60－R 003

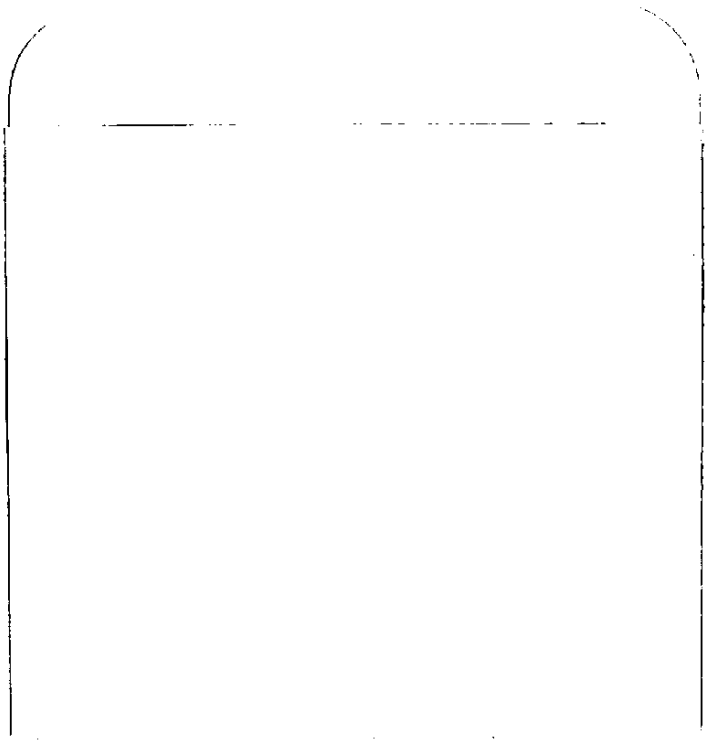
ソフトウェア開発・運用の高度化・効率化方法に関する調査研究報告書(Ⅲ)　－運用・保守－

昭 和 61 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60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，情報処理部門に対して，内外からさまざまなインパクトが強まっている。従って，このような状況から，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，このよう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，過去２年度に渡って実施したソフトウェアの開発計画段階およびソフトウェア開発段階の調査研究成果を踏まえ，ソフトウェアの運用・保守段階を中心として，コンピュータ・ユーザの運用・保守の実態に基づいてソフトウェア運用・保守の高度化，効率化，運用方法，保守技術・ツール，ソフトウェアの評価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，利用形態のいかん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・運用保守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６１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	興津勝	㈶地方自治情報センター研究開発部第１課課長
〃	菅野孝男	菅野技術士（情報処理）事務所所長
〃	妹尾稔	三井情報開発㈱技術開発部部長
〃	中島栄	総務庁行政管理局副管理官
〃	永渕寛幸	総務庁行政管理局副管理官
〃	前川征弘	外務省大臣官房情報管理室課長補佐
〃	政岡輝夫	通商産業省大臣官房情報管理課 電子計算機専門職
〃	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9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3システム部主任
〃	今木寿康	富士写真フィルム㈱システム管理部課長
〃	菅野孝男	菅野技術士（情報処理）事務所所長
〃	妹尾稔	三井情報開発㈱技術開発部部長
〃	落井徹	日本電信電話㈱データ通信本部 ソフトウェア開発部第4システム担当部長
〃	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

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最終年度として，初年度に実施したソフトウェア開発の計画段階，2年度目に実施したソフトウェアの開発段階に引き続き，ソフトウェアの運用・保守の段階に焦点をあ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，つぎのとおりである。

1. ソフトウェア運用・保守の高度化・効率化の考え方
2. ソフトウェア運用・保守の現状
3. ソフトウェア運用
4. ソフトウェア保守
5. ソフトウェア評価
6. ソフトウェアの今後の課題と展望

ソフトウェアの運用・保守は，ソフトウェアの開発にいたるまでの工程からみると，ソフトウェアの開発までが製品の製造の段階であり，運用・保守は，出来上がった製品の使用（利活用）の段階といった関係にあり，必ずしも軌を一にしがたい側面があるが，ソフトウェアのライフサイクルの面からは，開発→運用・保守→廃棄（処分）までを1つのサイクルとしてとらえられており，ソフトウェアの開発においては，運用・保守の負荷をいかに軽減することができるかについても開発設計段階において十分配慮されなければならない。

ソフトウェアの需要に対して，その供給が不足している，いわゆるソフトウェアの需給ギャップが増大している主たる原因は，ソフトウェア開発要員の不足のほか，ソフトウェアの保守（メンテナンス）作業に多くの工数を費していることが指摘されており，保守コストの低減，保守作業の効率化などが大きな課題となっているところである。

今回，民間企業のコンピュータ・ユーザ６００社に対して実施したソフトウェアの運用・保守に関するアンケート調査においても，過去に例をみないほどの高い回答率となっており，この問題に関し，多くの企業（コンピュータ・ユーザ）が強い関心を持っていることを示している。

今年度の報告書においては，運用・保守の高度化・効率化に関し，まず，その基本的な考え方を示したのち，ソフトウェアの運用，保守，評価の問題について順次述べている。これらのうち，とくに保守の問題について力点を置いてい

る。また，このほか，コンピュータ・ユーザ（民間企業）の運用・保守の現状についても紹介するとともに，ソフトウェアの今後の課題について考察しつつ，将来展望を試みている。

本報告書の各章ごとの内容について要約すると，つぎのとおりである。

(1) ソフトウェア運用・保守の高度化・効率化の考え方

この章は，本報告書の総論として位置づけられるもので，今年度調査研究のテーマである「ソフトウェアの運用・保守の高度化・効率化」に関し，それが必要とする背景と運用・保守の各々についての問題点を述べ，これらに対処していくための基本的な考え方を効率化の観点から取りまとめている。

すなわち，運用・保守の高度化・効率化を必要とする背景として，開発プロジェクトのバックログの増大，組織内における情報システムのインフラストラクチャ化，ソフトウェア資産の増大，運用・保守要員の増大とモチベーション，組織体制などの問題をあげている。つぎに，運用面での問題点として，システムの高度化，複雑化に伴って発生する問題，すなわち，通信と情報処理の境界領域が不明確な傾向が強く問題領域の特定化が困難となってきていること，運用トラブルの発生に対して回復作業の複雑化が増大していること，どこまでをシステム的にカバーするのか採算上の問題があることなど，また，運用要員サイドの問題点として，運用工程の作業領域，すなわち，運用という名前のもとでさまざまな業務が実施されているが，必ずしも同一土俵上で論じられないことから生じる問題，運用要員の教育の困難性やユーザサイドの要員問題，すなわち，稼動中にトラブルが発生した場合の即時対応の困難性の問題など，さらに，保守での問題点として，保守作業の分類に関するもの，保守作業工程上での問題や保守業務が固有に持っている問題点について論じている。そして，これらの問題に対処していくための基本的な考え方として，運用問題および保守問題の２つに分けて述べている。すなわち，運用問題解決のための基本的な考え方として，運用の無人化・自動化の推進，品質管理の導入や評価の実施を，また，保守問題解決のための基本的な考え方として，標準化の推進，ツールの活用，組織体制の確立，管理機能の強化，要員教育の充実などをあげている。さらに，運用・保守の共通的な対策として，外注政策の確立やシステム監査の導入の必要性を述べている。

(2) ソフトウェア運用・保守の現状

ソフトウェアの運用・保守に関し，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社を対象に，郵送調査および一部面接調査の方法により実施したアンケート調査結果（回答企業１６５社．回収率２８％）をもとに，ソフトウェアの運用については運用基準，システム監査，安全対策，運用の効率化対策などの現状を，また，ソフトウェアの保守については，総工数に占める保守作業の割合，保守体制，外注および保守対策などの現状やソフトウェアの評価の問題について取りまとめたものである。

(3) ソフトウェアの運用

ソフトウェアの運用の高度化には，効率性，信頼性，安全性が充分考慮された運用システムの確立が必要であることから，この章では，ソフトウェア開発後の運用の基本となる事項，すなわち，運用組織体制，運用基準の設定，安全対策，システム監査の実行について概略を論じ，最後にソフトウェア運用の効率化対策について述べている。

まず，運用組織体制に関しては，運用部門の要員と組織について，システムの運用形態やサービス形態に即した組織作りや要員管理（労働環境，健康保持，目標管理，外部要員の活用），ジョブ・ローテーション，要員教育の必要性を述べている。運用基準に関しては，運用に係わる作業規定項目，管理規定項目および運用で使われるドキュメントについて紹介している。

つぎに，安全対策に関しては，システムの運用における問題の存在を紹介し，それらに対する具体的な方策および安全対策基準について述べている。システム監査に関しては，システム監査の必要性，監査内容，監査の実行方法，留意事項などについて述べている。

さらに，システム運用の効率化に関しては，標準化，機械化の推進，外部資源の活用，ユーザ部門の活用などの方策について述べている。

(4) ソフトウェア保守

この章では，はじめに，ソフトウェアの保守の基本的な概念について説明し，ついで，保守管理の基本的事項，応急／日常保守作業の管理について説明したのち，外注開発する場合の留意点，保守に関する技術とツールについて述べている。

(5) ソフトウェア評価

この章においては，まず，「評価と改善の枠組み」において，評価を開発活動の評価と，完成したソフトウェアそのものの評価に分けそのコンセプトを示している。すなわち，開発活動の評価は，その結果をつぎの開発活動

にフィードバックさせ，より優れた開発のやり方を実現するために行うものであること，また，完成したソフトウェアそのものの評価は，その結果に基づき，ソフトウェアそのものを改良することおよびその結果をもたらした開発活動にその原因を探り，より優れた開発方法を策定することが評価のコンセプトであり，評価の結果のフィードバック機能の重要性を強調している。

つぎに，生産性の評価として，生産性の尺度，生産性に影響を与える要因などについて，また，信頼性の評価として，評価の観点，そのためのデータ収集の方法，分析方法などについて述べている。

このほか，処理効率の評価，使い易さの評価に関しても記述している。

(6) ソフトウェアの今後の課題と展望

ここでは，今後の課題として，ソフトウェア要員に関する課題（要員確保，組織，教育訓練），技術・ツールに関する課題（設計技術，製造技術，保守技術，生産管理，生産設備，生産支援ツール），ソフトウェア品質評価に関する課題（品質の造り込みと評価，ディグレードの防止）の3つをソフトウェアを取りまく大きな環境要素として取上げている。

また，以上の課題を解決する技術，方策などとして，標準化，再利用技術，ソフトウェア・パッケージの活用，移植技術，プロトタイピング，新高級言語，人工知能，ソフトウェアCAD／CAM，PWB，ソフトウェア・テスト，開発組織，QC活動，要員管理生産支援ツールなどに関し展望を試みている。

1. ソフトウェア運用・保守の高度化・高率化の考え方

1. ソフトウェア運用・保守の高度化・高率化の考え方

1.1 高度化・効率化を必要とする背景

情報化の進展とともに社会のさまざまな局面にシステム化ニーズが醸成され，しかもそれらが次々に新らしいニーズを要求し，丁度雪だるまが，どんどん雪によって大きくなるように肥大化してきている。これと同じように企業においても，コンピュータ技術やネットワーク技術の飛躍的な発展によって，システム化ニーズは増大の一途を辿っている。このような状況下において，人間だれしも，新らしくてしかも創造的な開発業務の方に目が向けられがちである。しかし，コンピュータ化の歴史が長い企業では，多くの蓄積されたソフトウェア資産を，長期にわたって運用・保守しており，その上さらに今後増大する多くのソフトウェア資産を運用・保守するためには多くの要員を投入しなければならなくなっている。

この事は，つまり，運用・保守段階での生産性の低くさが開発プロジェクトや保守要求に対するバックログ化を助長させる要因ともなっている。そこで本年度は3ヶ年計画の総決算ともいうべき運用・保守工程での生産性を向上させるために，運用・保守段階の高度化・効率化について研究することとしたものである。

システムは開発ニーズを凍結した段階から陳腐化が始まるといわれるように，システムを取り巻く環境は常に変化している。この変化についていけなくなり，必要な情報を提供出来なくなると，そのシステムは廃棄されることになる。

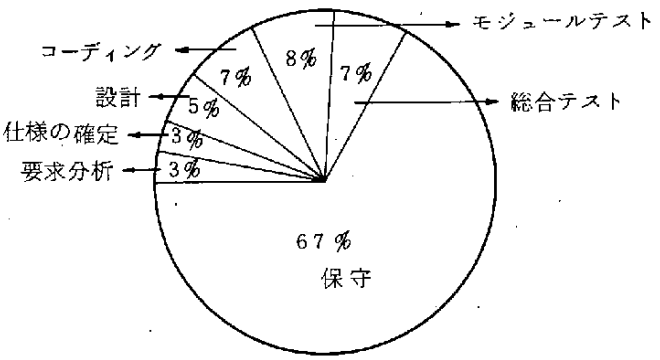
そこで多額な投資をして開発したシステムのライフを出来るだけ長期間化し，当初目論んでいた目的を常に最大限発揮出来るように修正を加え，利用者の必要とする情報が提供出来るようなシステムにする必要がある。

つまり外部環境の変化をシステムのなかに取り込み，使い勝手の良いシステムにレベルアップをしていくのがこの運用・保守段階の大きな役割でもある。

しかも，この運用・保守工程は，日常の業務活動の中で直接エンドユーザとの対話が比較的多く持たれる工程でもあるので，エンドユーザに対する対応いかんによっては，折角高まいな理想の下に構築されたシステムの評判を落としかねないので慎重な対応が必要とされる工程でもある。また，この工程は開発作業のように一過性で事が済む工程でなく，システムが完全に陳腐化してしまい，廃棄されるまでの長時間をめんどうをみる必要がある。そのために開発段階にお

いていくら生産性があがっても，運用・保守段階で多くの人手を要するようだと大きな問題となってくる。費用面に焦点をあてて，システムライフ全体でどの位の割合になっているか研究したプロジェクトがあるので参考までに掲げる（図表１－１）

図表１－１　大規模ソフトウェアのライフサイクルでの費用比率



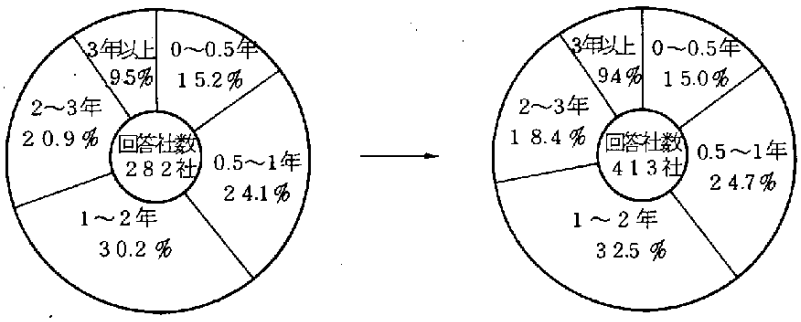
（メリーランド大学ビルコピッツの調査）

これでも分るように保守費用がシステムのライフサイクル全費用の約７割を占めており，この運用・保守段階の生産性向上が，強く望まれる理由でもある。このほかに，運用・保守工程はさまざまな問題点を持っているので，それらをつぎの５つの側面から眺めることにする。

(1) バックログの増大

コンピュータの先進国である米国においても２～３年のバックログがあるといわれているように，日本においても相変らず大きな問題となりつつある。このバックログ問題を日本では，日経コンピュータが２回にわたって調査を実施しており，非常に興味深い調査結果があるのでこれを参考にこの問題を眺めてみることにする。アンケート調査は，主に東証一部上場の企業を対象に，１９８２年１０月，１９８５年１０月の２回に分けて実施されている。

図表１－２　バックログの推移



１９８２年１０月のアンケート結果　　１９８５年１０月のアンケート結果

このアンケート調査結果は，各企業の必死の努力にもかかわらず，バックログ問題は，３年経過してもそれほどの改善がなされていないことを示している。しかも今後３年でのバックログ問題は，それほど大巾な改善が期待出来ない，変わらない，さらに増えるといったアンケート結果を合計すると約８割弱が悲観的な予想をしている。

つまりこの事は，生産性向上のために色々な方策（例えばツールの導入，標準化の導入etc）を行ったが，このペース以上にシステム化要求が増加し，相変らずバックログ化している様子を浮き彫りにしている。

この他の直接的な方策としては，人的資源の増強がある。しかし，一般の企業の場合，低成長期の時代にあってかなりの減量経営を強いられており，それほど多くの要員を確保することは困難になってきている。アンケート結果でもわずか１５.３％しか増加しておらず，システム化要求の案件の方がはるかに多く，バックログ問題の解消問題にはそれほど多くを期待出来ないと思われる。このような状況下においてバックログを解消するには，従来の延長路線上の対策では，それほど多くの事を期待出来ず，しかも急激な変化で動いている社会環境を考慮に入れると，２～３年間のバックログになっているエンドユーザは，コンピュータ部門の対応を待ち切れずに自分なりの対策を採り出すのではないかと思われる。このようになると，コンピュータ部門の主導性は薄れ，かなり大きなソフトウェア上の混乱が発生するのではないかと思われる。このような問題が発生する前にコンピュータ部門は，しっかりとした計画とフィロソフィとを持ってこのバックログ問題に対処していく

必要がある。そのために，従来あまり手がつけられていなかった分野への研究を積極的に進める必要がある。例えば

① データベースの利用部門への解放

② コンピュータ部門へのソフトウェア自動生産システムの積極的導入

③ プロトタイピングによるソフトウェア開発の導入

④ 保守部門の生産性向上策の確立

などが考えられる。又最近盛んに提唱されている情報センター（Information Center IC）などは，ソフトウェア問題を自部門内部だけで解決をはかるといった考え方でなく，エンドユーザ部門をも巻き込んで，ソフトウェア問題を共通の問題として，オープンにし，全社員一丸となって対処していくといった考え方である。コンピュータ部門は，このような動きに対して積極的に支援し，これを上手にコントロールすることによってバックログ問題解決方法の異なった糸口が見出せる可能性もあるので大いに活用するのも一つの方法と思われる。

(2) 情報システムのインフラストラクチャ化

最近のコンピュータ部門は，情報システム部あるいはシステム部といった名称に部門名を変更してきているが，このことは，単に名称を変更したというのではなく，このコンピュータ部門に寄せられる役割や企業内での部門としての位置づけが変ってきた結果として部門名称が変ってきたものと思われる。

コンピュータ部門の歴史は，経理部門や会計部門の片隅に一つの係や担当として設置され発展し，現在では企業での中枢的な役割を果たしている部門までに発展してきている企業もある。また，コンピュータ化の歴史もまだ２０～３０年の歴史しか存在していないが，企業の中で短期間のうちにこれほど急激な変化を強いられた部門はないといわれるほど，ますますその重要性が増大してきている。

アプリケーションの内容も，最近では省力化を目的とした業務処理の機械化というよりは，むしろ戦略的な経営を行うための基本的なツールとして活用されており，情報システム部門の強力な支援なしには企業経営が出来ないほど，面的な広がりと深化が行われており情報システムなしでは経営がなりたたなくなってきている。つまり，わずかなトラブルが，企業のライン活動を麻痺させるほどライン業務に密着してきており，この情報システムの良否が企業の競争力に大きな影響力を持ってきている。またこの情報システムの円

滑な運用・保守業務も同様に重要な役割を帯びてきている。そのために生産性や安全性，信頼性が重要な要件ともなってきている。

このように情報システムが，企業活動のある種のインフラストラクチャ化してくると，ちよっとしたトラブルが企業に莫大な損害を与える可能性があり，情報システムの運用や保守作業にミスが発生しないように万全の体制を敷くことが重要になってきている。

そこで運用工程や保守工程の中で人力に依存している作業に対しては品質管理の考え方や，作業の品質を高めるためのチェック機構を組み込んだり，要員教育に十分な注意を払う必要性が非常に高まってきている。

(3) ソフトウェア資産の増大

日経コンピュータが１９８５年１０月に東商一部上場企業などを対象にアンケート調査を実施しているが，このアンケート結果によると，一社当り平均約３００万ステップがソフトウェア資産として蓄積されており，しかも今後とも着実に蓄積されていく傾向が見られる。これに対してそれほど多くの要員増強が得られず限られた要員で，この増大傾向にあるソフトウェア資産を運用・保守していかなければならない苦悩が浮き彫りにされている。また要員問題では，バックログとの板ばさみにあってソフトウェア資産が増加したからといってそれほど多くの増強は期待出来ず，１人の担当者が運用・保守していく量は，年々増大していくものと思われる。このためにも運用・保守の生産性向上は必須要件となってきている。しかも企業が蓄積しているソフトウェア資産を１から開発し直すと平均6.5年の開発期間が必要になるであろうと日経コンピュータのレポートでは述べているが，再開発にはこのように莫大な年月を必要としており，そのためにもこの蓄積されたソフトウェアを最新の状態にして，いつでも利用出来るようにしておくのが，この運用・保守工程での重要な機能の一つでもある。さらにこの工程では自然災害や人災といった問題からも，このソフトウェア資産を守る義務があり，そのために多額の投資や安全対策を事前に構じておかなければならない。またソフトウェアの中味である機能的な面について眺めても，そのソフトウェアの置かれている環境やユーザニーズは常に変化しており，折角開発したソフトウェア資産を有効に活用するためには，この変化に矛軟に対応出来るようにしておく必要がある。蓄積されているソフトウェア資産が，それほど多くない時には，運用・保守といっても片手間に対処が出来たが，ソフトウェア資産が大

きくなってくると，しっかりとした組織体制と管理が重要となってくる。

つまり大規模なソフトウェア資産も，しっかりした運用・保守工程の管理の手に委ねなければ，「コンピュータ，ソフトウェアがなければただの箱」となってしまい，折角長い年月かけて開発してきたソフトウェアも宝の持ち腐れとなってしまう。そこでソフトウェア資産の規模が増大するにつれて，ますます運用・保守工程の確立と，生産性，信頼性，安全性などに十分な注意を払う必要がある。

(4) 要員の増大とモチベーション

運用要員は，ソフトウェア規模の増大の割には，オンライン化や運用の自動化，省力化のためにそれほど大きな増加は示していない。

しかし，システムが高度化，複雑化し，企業間システムや分散処理が進展してくると，システム間の接点であるインターフェイス部分が多くなり，その結果としてこの部分での円滑な情報交換や調整，制御のために徐々に人手が必要になってくるものと思われる。つまり，量的には幾分視点の異なった領域での増加や，技術力の高い質的な増強といった両面が要請されてくる。

一方，保守要員に関して眺めてみると，ソフトウェア規模の増大につれて，ハードウェアの変更，社会環境の変化，利用者ニーズの変化といった事態に対処するために，ソフトウェア規模いかんにもよるが，新規の開発が終了した段階から新らしく保守担当者を任命する必要がある。このようにして新規の開発が終了し，その都度，新らたな要員を追加していかなければならないとなると保守要員がますます増大してくることになる。しかし，片一方では開発ニーズの増大があり，これに充分な対応がとれずにバックログ化となっており，これへの対処のためにも保守要員だけを増強出来ない。そのためにも保守業務の生産性向上が大きな命題ともなってきている。

また，一般的な人間の行動傾向として，新らしくて創造性の高い開発業務につきたがる傾向があり，それに反して，他人の開発したシステムの保守業務につきたがらない傾向がある。そのために，運用・保守要員には比較的経験年数の低い要員や新人が割りふられる場合が多い。そのうえこの作業に従事している要員の心理状態には，人の嫌がつている作業に従事させられているといった考え方があり，しかも，評価面に関しても開発業務よりも一段と低くみられ，ちよっとしたミスでも大きく扱われたりして，精神的に安まらない環境状態にある。さらに，保守環境を眺めてみても開発環境と比較す

ると生産性向上のためのツールも十分整備しておらず相変らず人海戦術的な人手作業に依存している場合が多い。

また，企業側の姿勢としても，常日頃運用・保守の重要性をうたっている割には，運用・保守要員の処遇にその考え方が十分反映しておらず，理屈と行動が全く一致しておらず運用・保守要員のモチベーションを阻害する要因ともなっていると考えられる。

前述したように保守作業は人手に委ねられる部分が多く，いかにして保守要員にモチベーションを持たせて作業を行わせるかが，保守生産性の向上に大きな影響力を持つものと思われる。そこで保守の重要性を主張している企業として，その歌い文句どおりにスキルの高い優秀な要員を保守部門に重点配属したり，評価方法などを変更するといった抜本的な改革をはかることが保守要員のモチベーションを高くする要因にもなるので，企業としても十分この点に配慮する必要がある。

(5) 組織体制

開発，運用，保守に係わる組織体制については，色々な所でアンケート調査が実施されており，傾向としてコンピュータ部門の規模，蓄積されているソフトウェア規模といった規模の大きさによって変化が現われている。

蓄積されているソフトウェアの規模が小さい場合には，あらゆる作業を一人の担当者あるいはグループによって対処している場合が多く，逆にソフトウェア規模が大きい場合には，開発，運用，保守が明確に分離されている場合が多い。

つまりコンピュータ部門の歴史が浅く，規模も小さい場合には，要員もそれほど多くいる訳でもなく一人の担当者がすべての業務を行わなければならないのが実情である。これに対してソフトウェア規模や要員規模が大きくなると，生産性を向上させるためにそれぞれの役割や機能を明確化し，それを分担して行うようになるのが一般的である。このことは，組織体制問題を論ずるには，ソフトウェア規模，要員規模といった規模の問題を無視して議論することが出来ないことを意味している。ただし，規模のことを十分加味して，この問題を考える必要があるがつぎのようなことにも十分注意を払って組織体制問題を検討する必要がある。

① 個人がいくら優秀であっても，出来る作業には限界があり，注意を払える領域にも限界がある。

② 個人にすべてが委ねられて仕事をしている場合，外部からのチェックがないかぎりいくら意識が高くても最終的にはマンネリに陥いり，規則も十分遵守されなくなる。

このことは，例えば開発要員がシステム内容を十分熟知しているので，そのまま運用・保守を継続させた方が生産性が高いという考え方がある。確かに，この方法は引継作業を実施する必要もなく，しかも修正作業も迅速に出来るといったメリットがある。しかし，長期的な視点から眺めると，システムが属人化してしまい，システムが持っているさまざまな問題点が表面に出ずに潜在化してしまい，ある時期に一度に出てくるといった問題を含んでおり，ローテーションも十分行えず，人員の有効配置にも支障をきたすといった問題までも出てくる。

そのためにも，一過性の狭い視野で物事を判断するのでなく，長期的な観点からこの組織体制問題を考える必要がある。比較的コンピュータリゼーションの長い歴史を持っている企業が設定している機能分化の方式はそれなりの意味を持っているので参考にすべきであるが，しかし，企業にも独自の文化があるように自分の企業の体質に合った独自の組織体制を作りあげる必要がある。

1.2 運用での問題点

1.2.1 運用工程の作業領域について

システムのライフサイクルを眺めてみると，新らしくシステムを創造する開発工程と，開発終了後そのシステムが廃棄されるまでの運用工程とに大きく二分される。この2つの工程の中で運用工程は，システム化するために色々な約束事を取りつけて新らしいシステム環境を構築する開発工程と比べて，エンドユーザと直接接触する期間が長い。また運用領域には，かなり性格を異にした作業が混在している。そこで運用工程段階で発生する保守作業を除いたすべての作業を運用工程段階の作業領域とここでは定義することにする。この定義領域をさらに細かく分類するとつぎのように分類出来る。

① 狭義の運用：コンピュータを中心に円滑な運用をはたすための業務

② 広義の運用：アプリケーション単位の運用で，ユーザとの対応窓口を行う業務

この広義の運用の中には，エンドユーザ側にいて実際の業務運用に携わっ

ている要員もいるが，ここでは検討外にする。またコンピュータ部門の規模が小さい所では，一人で何ん役の業務をこなしているが，生産性といった観点から，運用・保守といった機能分担を意識して業務を遂行することが重要である。

つまり，高度化・効率化を目指すためには，一人の要員が例えあらゆる作業を実施していても，作業工程，作業内容，作業目的などを常に意識しながら作業を行っていると，問題点や改善すべき事柄などが自然に明確となり，生産性向上につながるといった基本的な考え方で，この運用・保守問題を眺めてみた。しかし，最近のようにオフコン，パソコンなどといったハードウェアの導入が活発化し，一方ソフトウェア技術やネットワーク技術の高度化によって分散処理が企業の中に多く採り入れられたりしており，運用工程も単純なパターンだけでは議論出来なくなってきている。そのうえ，システムの形態によっては，エンドユーザ自身がシステム運用の中に，直接参加するケースも出ており，運用領域や運用問題をより一層難しくしてきている。このエンドユーザが直接的に運用する場合，このエンドユーザの運用領域に対しては，操作性や容易性を高めるためにかなり注意が払われるようになっている。

1.2.2　システムの高度化・複雑化による問題点

情報処理技術とネットワーク技術を上手く融合させた大規模システムが社会のさまざまな局面に導入され，利便性に大きな貢献をしている。これと同様に企業内システムでも企業グループ，取引先を含くんだ縦系列の高度で複雑なシステムを作り上げつつある。このように複雑化したシステムであればあるほど，トラブルが一度発生すると，どの部分でトラブルが発生したかの特定化がなかなか出来ず，原因究明に多くの時間を要する場合がある。また開発時点では，想像だにもしなかったようなシステムの利用の仕方をしたためにシステムダウンにつながり大きな問題になるといったケースもある。

このことは，開発部門が新らしいものを創るといった意欲だけに燃え，実際にこのシステムを利用する側の環境条件を十分念頭に置かなかったために発生した問題と考えられる。このようにシステムが複雑化し，対象領域が広域化してくると運用段階での運用環境を事前に十分想定してシステムを構築しないと思いがけない問題が運用段階で発生してくるので注意する必要がある。

又企業活動の根幹をなすような高度化したシステムの場合には，トラブル

の発生によって致命的な打撃を受ける恐れがあるので，システムダウンが長期的になった場合には，そのシステムなしでも十分対処可能なような仕組や訓練を十分構じておく必要がある。勿論この場合，採算問題を無視しては考えられないが，そのシステムが企業内でどのような位置づけにあるかといったリスクマネジメント的な観点からも十分検討しておく必要がある。

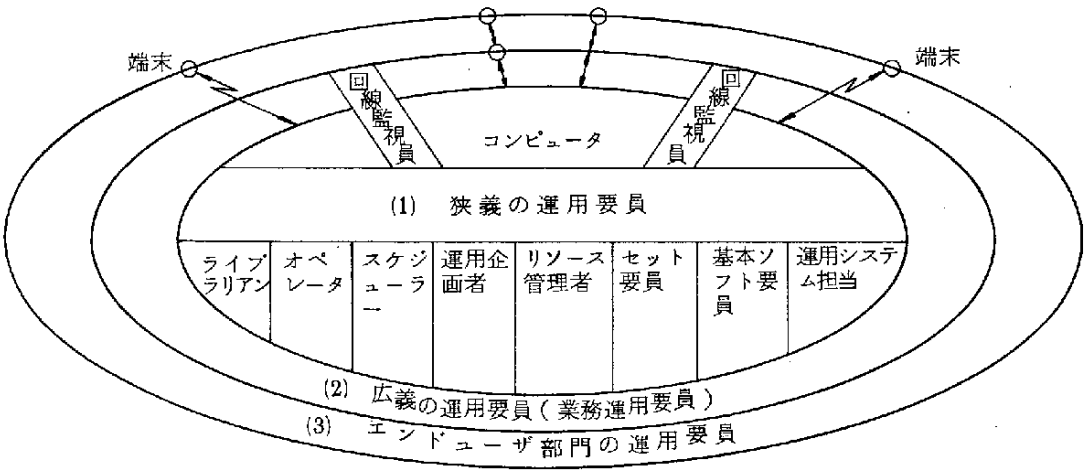
また，トラブルが発生すると，複雑化したシステムの場合には，単純には正常状態に復帰させることが困難である。復帰作業の過程でミスを行ってさらに状況を悪化させるケースなどもあり，出来るだけ復帰作業の自動化システムなどを事前にシステムの中に組み込んでおく必要がある。しかし，このように自動復帰的な機能を持っていたとしても，最終的な判断業務を行うのは運用要員であり，ますます運用要員の技術の高度化が要請されてきている。

また，トラブル対処のための作業指示などは運用マニュアルなどで十分ドキュメント化しておく必要がある。しかし，このドキュメント上だけでは十分表現出来ないような事象が，システムが高度化，複雑化すると一段と増加してきており，これが運用ノウハウとして運用要員に蓄積していくので，今後の問題として，このように運用ノウハウを豊富に持ち，しかも高度な技術力を持った要員の知識，能力をいかにして他の要員に技術移転していくかが大きな課題でもある。

1.2.3 運用要員の問題

“1.2.1”で述べたように運用領域には，性格を異にした色々な業務が存在しているにもかかわらず運用という名称のもとで統一的な業務を遂行しているように考えられがちである。しかしこの運用領域には，非常に単純なテープの出入庫を管理するライブラリアンからリソースの管理といった高度な技術力を必要とするような業務までを含くんでいる。そのためにこれらの問題点を同一の土俵のもとで議論できないのでそれぞれの領域毎に分けて，代表的な問題点について眺めてみることにする。それぞれの運用領域にどのような運用担当者がいるかを図化したのが，図表１－３である。

図表１－３　運用担当者の構成図



(1)　狭義の運用要員の問題

システムの高度化，複雑化が進むにつれてシステムそのもののブラックボックス化が進み，トラブルが発生しても運用要員では十分に原因がつかみきれなくなってきている。しかもトラブル原因も色々な要素が絡み合った結果として発生してくると，なお一層問題点が複雑化し，従来のように単にコンピュータを運転するだけの技術力を持っているオペレータでは，対処不能となってきている。そのために新らしく回線監視員といった職種が発生し，エンドユーザと直接対応し，それと同時にコンピュータの動きを常にウオッチしながら，エンドユーザに最新情報を提供し客先のトラブルによるいらだちの解消をはかるといった業務を行っている。

この要員には業務運用を経験したり，リソース管理，運用企画，基本ソフトなどの業務経験を持ったかなり高い技術力を必要としている。このような要員は，各企業にもそれほど多くなく，オンライン化やネットワークシステムの増大によってますます要求され，その意味において狭義の運用要員の中でも回線監視ができるような高度な技術者の早期の育成が強く要請されている。

１９８５年の産業能率大学のオンラインの導入実態調査によると７２％の企業でオンラインが導入されており，このためにネットワーク運用やオンライン運用と同時にバッチ運用の両面に精通した運用要員を育成する必要がある。

そのために運用企画者やリソース管理者は，バッチ業務やオンライン業務での運用特性を十分熟知しておく必要がある。

さらに前述した回線監視員はエンドユーザとコンピュータとの接点にあり，色々な問題に即座に対処出来る能力が必要とされ，しかも苦情処理的な窓口業務をも兼ねているので，従来の運用要員にはそれほど必要とされなかった営業的なセンスを持っていることも必要とされるようになってきた。つまりこの要員にはエンドユーザの業務内容がある程度理解でき，しかもコンピュータ技術（特に基本ソフトに近い領域の技術），ソフトウェア技術，ネットワーク技術などの高い技術力を持ち，しかも営業マン的なセンスといったかなりオールランドな能力を持った要員が必要とされ出してきている。そのためにはしっかりとした要員に対する育成計画を確立する必要がある。またこの回線監視は，かなり強い緊張を強いられたままで業務を遂行しなければならず，精神衛生的な面での配慮も今後重要な課題となってくるものと思われる。

(2) 広義の運用要員問題

ここでの作業は，エンドユーザと狭義の運用要員との接点にあって，業務の運用計画をたて，それに沿って正確な，高品質の情報をエンドユーザに提供する役割を負っている。この場合の運用要員の姿勢としてエンドユーザー側に立った物の見方をする必要がある。そうでないと，トラブルが発生しても，エンドユーザ側の立場にあまり目を配らずにコンピュータ側の問題をあまりユーザ側が理解できないような言葉を使って，無理やり納得させられる傾向があり，そのためにエンドユーザ側にいらだちやもやもやが残り，良い関係とならない。そこでユーザ側の立場にたって，ユーザの理解し易い言葉で話し，ユーザ側の痛みも十分理解できる運用要員が必要となってくる。その意味では，ユーザ側の業務内容を運用面から熟知し，コンピュータ，エンドユーザ両面から発生するトラブルを上手にさばきながら，最終的にはエンドユーザの満足する情報を届ける役割を負っており，ある面では営業的なセンスも必要となってくる。

また，エンドユーザとの対応窓口という点から，ここでの対応いかんによっては，折角多額の投資をして開発したシステムの評判を落とすことにもなりかねないので，その役割を十分認識する必要がある。また運用環境を眺めてみると，エンドユーザとの窓口ということでのストレスがたまり易く，しかもちよっとしたミスからでも大問題が発生する場合もあるので気

が安まらない，そのうえ上手く運用されて当り前，悪くいくと大きな減点になるといった管理姿勢に対する不満といったものが潜在化している職場でもある。

そこでリラックス出来るような環境や雰囲気作りが大切であり，運用業務は比較的属人化し易いが，トラブルが発生した場合，一人ですべてを対処させるのではなくグループで対応させる方法なども考える必要がある。

また，トラブルシューティングの時間もかなり人によって大きな開きがあり，トラブルシューティングの方法なども十分整理分析し，ノウハウ集の形で取りまとめておく事も技術力の平準化に効果があるので，運用要員全員をこの作業に参加させ一体感を持たせることも重要である。この事は運用要員の孤立感をなくすためにも重要と思われる。しかもこの作業は継続的に実施することが重要である。

(3) 管理者の問題

ソフト工学の考え方の導入により，開発工程ではかなりしっかりとした工程分けが行われている（但し工程分けの基準は，必ずしも統一化されておらず企業によってはかなり異なっているが）。そのために管理，品質保証，生産性といった側面にかなり大きな効果を発揮している。これとは対照的なのが運用工程であって，ここではこれといった工程分けが存在しておらず（昭和５６年に㈳日本情報センター協会が実施したアンケート調査結果にもあるとおり），この分野の未発達状況を示しているものと思われる。この工程での作業の高度化・効率化を達成するためには，どうしても作業手順を明確化し，誰れでもが行っても同じ様な品質の作業結果になるような仕組が必要になると思われる。

しかし，この工程を管理している管理者には一般的に言ってこのような考え方を持っておらず，担当者まかせのうえに，単に担当者の上に座っているといった色彩が強い。あるいはトラブルが発生した時に頭を下げに行くのが管理者の役割であり，これが唯一の仕事であるといった心得違いの管理者さえもいる。

そこで第一番目に実施しなければならないのは管理者の意識改革がある。これと同時にソフト工学的な考え方を運用工程にも導入し工程の明確化とそれに基づいたしっかりとした管理の実施が大切である。しかも品質を運用要員の能力に依存するのでなく品質管理的な手法を活用する必要がある。また，運用作業にも計画性を持たせ，運用要員に依存した受身的な管理でな

く能動的な管理へと管理の方法を転換する必要があると思われる。このことが運用要員の技術力向上につながり，ひいては管理者の管理能力の向上へと発展していくと思われるので，この基礎的事項の充実が重要と思われる。

また，情報処理の世界は特殊な世界であり，そこでの管理はかなり難しいと良くいわれている。しかし管理といった基本原則から眺めてみると他の分野での管理とは基本は全く同一と思われる。しかるにこの分野での管理者の多くがプログラミングやＳＥに従事していた者がある年限がたったのでその中から適当な要員を管理者として登用する場合が多い。そのために十分な管理者教育も受けておらず，しかも本来の管理者がいかなる役割を果たすべきかの役割意識を十分に持っていない場合が多くそのために運用工程に存在する多くの問題が未解決のまま放置されている場合が多い。

一般的には管理者の役割としてつぎのようなことがいわれている。

① 部下の能力開発
② 職場の統率
③ 職場の革新
④ 部門間の協力
⑤ 業務の達成
⑥ 戦略への参画

これらの役割はコンピュータ部門の管理者にも十分適用される事柄でもある。そこでこれらの役割が遂行出来るような管理者の育成が今後ますます重要となってくると思われる。

1.3 保守での問題点

多額の投資をかけて開発したシステムの延命化をはかるために，エンドユーザニーズやシステムの環境変化を取り込んで，保守作業を長期間にわたって継続的に実施する必要があった。しかし，従来の考え方は，エンドユーザが持っているシステム化ニーズに対して，いかに効率的，かつ，トラブルが無いようにスマートに開発するかといった点に力点が置かれ，保守工程は，開発作業の後始末的な立場に置かれていた。そのために保守部門がどれだけ苦労して開発の後始末を実施しているかなどには全然気にかけず，開発部門は作ることだけに専念している場合が多かった。

ところが，Boehmが予測した保守費用コストの増大，バックログ問題，ウオー

タフォールモデルによる保守工程の明確化などによって保守工程の重要性がかなり一般的にも認識されてきた。この保守作業の宿命として長期間にわたってシステム機能を維持していかなければならないが，保守期間が長期にわたってくると開発作業に携わらなかった要員が新らしく割り振られるケースが多くなってきた。その結果，他人が作り込んだバグを一生懸命おっかけ，問題点の解明に努力したりしている。また，保守作業で良く発生する緊急を要するバグ修正作業は，それほど十分な検討期間をかけないままにシステムの修正を実施しなければならないといった保守業務個有の性格を持っている。これらのことは，保守要員自身が受身的な仕事を強いられ，主導的，計画的な立場で作業を遂行できず，ある意味ではかなりストレスのたまる職場でもある。そのために出来ることなら保守業務は避けて，開発業務だけをやっていきたいと考えている要員が多い。

そこで，ここでは保守業務が持つさまざまな問題点を各工程毎に明確にし，それをいかなる方法で解決していくかが解明できるように整理してみた。

1.3.1 保守内容の分類

保守作業に関する明確な定義はないが，Riggs（１９６９年）は，保守作業をつぎのように定義している。「コンピュータ・システムをユーザの要求，データ処理を行ううえでのいろいろな操作，それに付随した事務処理的な働き，政府や他の行政機関からの外部的な要請といったものと継続的に合致するようにする一連の作業」。

しかしこの定義は保守業務をかなり包括的，観念的に表現しているので，保守業務が持つさまざまな問題点が明確に浮き彫りにされない。そこで保守業務の問題点を，保守作業の工程に沿って眺めると同時に，問題点の視点を次の２点に絞って眺めてみることにする。

① 保守業務内容に焦点をあて，しかも作業実績データの把握といった観点から

② 保守工程を明確化し，それに対応した管理の側面から

日本では，保守作業での体系化についてあまり十分な検討がなされていないので外国の例を参考にした。

㋑ Canning による分類（１９７２年）

対象プログラムを保守しなければならなくなった理由によって分類し

ている

- バグによる修正（プログラムが動作しなくなった）
- プログラムが間違った情報を作成するための修正（プログラムは正しく動作するが，アウトプットされる情報が正しくないため）
- 運用環境の変更による修正（運用環境が変化し，この変化に適応するようにプログラムを修正する）
- 機能拡張や最適化のための修正（既に開発済の機能に，さらに新らしい機能を追加したり，削除するため）

㈱ Monneyによる分類（１９７５年）

- リペア（修繕）
- リビジョン（改訂）
- エンハンスメント（機能拡張）

㈲ Boehmによる分類（１９７６年）

- Software update （ソフトウェアの機能上の仕様を変更する）
- Software repair （ソフトウェアの機能は，そのままにして修繕を行う作業）

などの分類がある。しかし，これらは，分類上の基準が必ずしもはっきりしていないので，分類上の基準について眺めてみる。

① 原因別による分類
② 時間経過による分類
③ 保守対象物による分類

などが考えられる。この中で，比較的適切で，日本でもかなり利用されている分類としてSwanson（１９７６年）の原因別による分類があるので，これを簡単に紹介する。また，ここではこのSwansonの分類に沿って問題点を整理してみることにする。

㈠ 修正保守（Corrective maintenance）

- 処理上の欠陥
- 性能上の欠陥
- 製造上の欠陥

㈡ 適応保守（Adaptive maintenance）

- データ環境の変化
- 処理環境の変化

㈣ 完全化保守（ Perfective maintenance ）

- 処理の非効率
- 機能の拡張
- 保守性の向上

また，保守問題を解決するために，国家的プロジェクトとして推進した協同システム開発㈱が，このプロジェクトを推進するに先立って実施したアンケート結果によると（これは基本的にはSwansonの分類をベースにしている）図表１－４のような結果が出ている。

図表１－４　保守作業の分析

保　守　作　業	比　　率
①機能の追加，変更，削除	５９％
②機種，ＯＳなどの変更による変換作業	１３％
③機能上の誤り修正	１１％
④性能改善	１０％
⑤性能上の誤り修正	４％
⑥その他	３％

このようなデータを分析することによって，開発段階での作業状態を推測でき，しかも，保守業務のなかでどのようなタイプの保守作業が多いかが把握できて，対策もたて易くなる。つまり，このように保守作業内容を多方面から分析し，類型化することによって，どのようなツールを整備したら良いか，どのようなドキュメントを用意しておけば良いかなどの対策をたてることが出来る。このような保守作業に関する各種のデータベースを構築することが，保守業務の高度化・効率化への道に通ずるものとして十分整備する必要がある。現状では，このようなデータを整備している企業は非常に少ないものと思われる。

1.3.2　保守作業上での問題点

保守作業も運用工程と同様に人に依存する割合が高く，保守作業のなかには工程といった概念が全く無く，どんぶり勘定的に保守作業と総称している

場合が多く，しかも，作業計画もあまりたてずになんとか期日に間に合うような形で作業を実施している場合が多い。

しかし，この作業を良く眺めてみると，期間的には，長期間必要とするものや短期間で済むものやいろいろあるが，開発作業と同じような工程をふんで保守業務を行っていることが分る。

つまり，保守作業が準開発とも称されるように，工程的には開発作業とかなり酷似しており，ソフトウェア工学で提唱している工程の概念を導入し易い環境にあった。

ただし，作業工程が，開発作業と非常に似いるといった理由だけから，開発工程での作業方法の考え方が，そのまま保守作業にもあてはまるかといえば，そうではなく，詳細レベルではかなり異なっているので，保守作業での特性（期間的には短かい場合が多いetc）などを十分考慮した独自の方法論を設定する必要がある。

例えば，開発作業の場合，無のものから新らしく有のものを作り出す場合が多く，作業姿勢としても能動的である。しかし保守業務の場合，対象となるソフトウェアが既に存在しており，これを無視して作業を進めることができない。そのために作業上いろいろな制約を受け，常に受け身の立場で作業せざるを得ないのが保守作業である。このことが保守作業にさまざまな影響を与え，保守作業固有の問題も生れてくることになる。

また，別の特徴として開発工程的な作業を保守作業では，極端な例として数分で実施し，正しい情報をユーザに提供しなければならない場合がある。この場合でも，作業としては当然開発工程の各フェイズを次々と頭の中でこなし，プログラムを修正し，コンピュータにかけるといった一連の作業が行われるが，作業的には工程といった考えが存在しておらず，どんぶり勘定的に頭の中で瞬時にいろいろなことが考えられ実施されてしまう。

しかし，いずれにしても保守要員は内外からの問題点の指摘や新らたな要請に対して，期間としての短期間といった問題はあるが，保守作業を実施しているので，下記の作業工程に沿って問題点を眺めてみることにする。それと同時に，保守作業に大きな影響を与えていると思われる文書化作業と管理業務についても眺めてみることにする。

① 分析作業

② 設計作業

③　プログラミング

④　テスト作業

⑤　文書化作業

⑥　管理作業

これらの作業についてさらに詳しく分析してみることにする。

(1)　分析工程での問題

保守作業の中でもこの工程が一番重要な工程であって，エンドユーザ側からの機能追加要求やシステム上の欠陥に対する指摘などをこの工程の段階でしっかりと受け止め，その内容を十分分析し，どのプログラムに影響を与えるか，あるいはどのプログラムに問題があるかなどといった特定化をはかる工程である。そのためにここでの分析が十分行われていないと，以降の工程での作業がどんなに正確に行われたとしても，ユーザ側が必要とする情報を提供できなくなる。

しかも，修正保守や適応保守の中には，何んといっても緊急性が大事であるといった要請があり，この場合には，それほど多くの時間をこの工程にも十分かけられず，その結果かえって大きな問題に発展するケースもある。その意味において，この工程での作業を効率的かつ正確に実施できるか否かが，この工程での重要な課題でもある。そのうえ分析作業を支援するためのツールなども十分整備されておらず，保守要員の経験や技術力に非常に依存している工程でもある。そのためにもすこしでも保守要員の作業効率を高めるようなツール開発がこの工程での重要な課題でもある。

また，ここでの要員は，業務知識やコンピュータ知識（ハード，ソフト両方）を持っていることが強く要請されているが，現状では，この種のタイプの要員が少なく，早急な養成が望まれている問題でもある。そのうえ保守作業は，創造的でなく技術力の向上がはかれないといった変な誤解があり，保守要員へのなり手が少なく，絶体数的にいっても要員不足が発生しており大きな問題になりつつある。

さらに，ここでの重要な作業にドキュメント作りがあり，このドキュメントの量的な面での過不足，内容面での過不足がこの分析作業に大きな影響を与えるので，バランスのとれたドキュメントの整備に心がける必要がある。

(2)　設計工程での問題

分析段階での検討結果をベースに，いかにして開発時の設計思想を壊さ

ずに，ニーズを組み込み再設計するかが重要な問題でもある。そのためにこの作業を担当する要員には，かなり高いソフトウェア技術力を持っている必要がある。この設計の仕方いかんによっては，作り出されるプログラムに，効率性の面や保守性の面に大きな影響力を持つことになるので十分注意する必要がある。プログラムは一度その構造がくずれ出すとつぎつぎに自壊作用をおよぼす傾向があり，しかも，これをもとに戻すためには大変な努力を必要とするので，１つ１つの保守要求を，設計思想をくずすことなく設計することがここでの重要な課題でもある。

また，継続性の観点からは，再設計の方針などについてドキュメントの形で残しておくことが，次回の保守に際しても重要な参考資料ともなるので必須の作業ともいえる。一般的には，この作業は緊急性という隠れみのに使われて，あまり残っていないので，管理者はこれらの作業結果をドキュメント・ベースに確認しておくことが重要な作業でもある。

(3) プログラミング工程での問題

保守作業は全般的にいって，開発作業のように分析者，設計者，プログラマなどといった職能分化が明確でなく，１人で何ん役もこなしている場合が多い。しかも，開発作業での品質を高めるために，開発工程での作業終了毎にレビュー工程を設定して品質を高めている。これに対して保守作業の場合，緊急性があれば，これが「錦の御旗」となり，保守工程でのオーソドックスなやり方が無視され，頭の中だけで，分析，設計，などが行われ，いきなりプログラム修正作業が行われる場合が多い。そのうえドキュメントも不十分なままに放置されてしまい，最終的には誰れからも見向きもされずに倉庫に仕舞い込まれてしまうことになる。

その結果開発段階で十分検討し設計されたプログラム構造がどんどん破壊され，つぎはぎだらけの非構造のプログラムとなり保守の生産性がどんどん低下することになる。そのうえ，この保守作業を担当する要員が，技術的にはそれほど高くなく，プログラムの劣化に一層の拍車がかけられ，保守生産性を阻害する悪循環の原因ともなっている。

またプログラミング作業は，プログラマの個性がかなり入り込む作業であるので，開発工程では，かなり厳格な標準化が設定され，遵守させる方向で作業が進められているが，保守作業での標準化がかなり遅れており出来上ってくるプログラム品質にはかなりバラツキがあり，長期的なスパン

から眺めると保守作業のダーティ化を一層助長させることになるので，今後はこのプログラミング段階を始めとして，それぞれの工程での標準化が重要な課題になってくるものと思われる。

(4) テスト工程での問題

この工程は保守作業結果の最終的な品質を保証することにある。対象案件が正しく処理されていることを確認するのは当然のこととして，この工程で重要なことは，プログラミング段階でいろいろと作業した結果が予想外の個所（設計段階での検討ミス，プログラムの修正ミスなどによって発生した）に影響を与えていないかを確認することである。急いで作業を実施し，部分的な確認しかしなかったために，実際の本番になった段階で，今までは正常に動いていた部分に影響をおよぼし，大きな問題になる場合もあるので，この工程では細心のチェックを実施する必要がある。

しかし，開発段階でのテスト工程ほど多くの時間を保守作業のテスト工程ではかけられないので，システム修正やプログラム修正による余波をどのように効率的にチェックするかがテスト工程での生産性向上や品質向上への重要な課題となる。しかもこの作業自身は，かなり保守要員の経験や勘が働く作業領域でもあるので，このようなトラブル事例集を作り，保守要員の教育などに利用することによって効果を発揮するものと期待される。

また，このテスト工程で良くおかすミスとしてテストデータの問題がある。つまり，本番データが手近かにあるために，安易な発想で本番データを使いがちであるが，本番データであるがゆえに，必ずしも発生しうるあらゆるケースに触れたデータが存在している訳でない。そのために本番データを使う場合には，その点を十分認識したうえで，目的を持った使い方をする必要がある。またこのテストデータを上手に作ることが，テスト工程での作業効率や品質を向上させることになるので，テストデータ・ジェネレータなどといったツールを上手に使うことも，この工程では大切な作業でもある。

(5) 文書化作業の問題

保守作業の中で必ず問題として提起される作業に文書化作業がある。この作業は，保守作業のなかでもその重要性について関係者間でも十分認識されているにもかかわらず，実際には，文書化作業が等閑視されている場合が多い。そのために問題が発生し，あわてて対処するといったことが繰

り返されているのが実情である。また保守業務に従事している要員の中には，極端な例ではあるが，かなり多数を占めていて「ドキュメント無用論」を唱え，プログラム・ジャーナルさえあれば十分だとも主張するものもいる。保守業務は長期間にわたって継続されるという特徴点から眺めてみると，このことは一過性の対処であって，保守業務の本質を十分理解していない，視野の狭い考え方でもある。しかもプログラム・ジャーナルさえあればといった議論は，保守作業の属人化を出来るだけ回避したいという方向性に対して，全く逆の属人化を助長させることになるので，文書化作業は，勿論程度問題はあるが，十分に実施する必要性がある。また別の問題としてドキュメントが最新状態になっていないので安心して利用出来ないといった不平をこぼしている場合があるが，これは最終的には自分自身が常に文書化作業を着実に実施していれば，そんな問題は発生しないはずであるし，保守業務のルールとしてこの作業が確立していないとこのような問題が発生してくる。つまり緊急を要する業務が多くてついついドキュメントを更新しなかったために，最新状態のプログラムとドキュメント内容とが不一致を起こしてくる大きな原因ともなっている。しかしこの緊急性を必要とする保守業務は，前述した協同システム開発㈱のアンケート調査によれば，機能上の誤りの修正作業が11％，性能上の誤り修正が4％となっており2割にも達していない。それに対して機能の追加，変更，削除が6割も占めており，緊急性を要する保守業務がそれほど多くなく，作業に十分計画性をもたせ，かつドキュメント化作業も実施し得る状態ではないかと思われる。要は適切なドキュメントがどんなものであるかはっきりしないこととドキュメントを作成するように訓練されてないことが問題のようである。また管理者自身も，担当者の言をそのまま信用してドキュメント作業の不十分さに対してある種のあきらめを持っていたが，文書化に対して明確な方針を持つ必要がある。このことが，泥くさくなかなか生産性が向上しない作業を着実に向上させる重要な手段の1つと思われる。この場合，どのようなドキュメントが保守業務に有効な働きをするかを十分研究し，それを習慣的に行うような仕組み作りも重要である。

(6) 管理者の問題

開発工程段階で作業した結果（これが意図的であるにしろ，無意識的であるにしろ）がそのまま保守工程に持ち込まれ，問題が発生すれば，保守

要員の責任において保守工程の中で対処していかなければならない。つまり開発工程段階の不具合がすべて持ち込まれることになる。このことに対し，一般的に管理者は無関心で担当者まかせにする場合が多い。

しかしこれでは問題点は一向に改善されないので，管理者は積極的にこれらの問題に突込んで問題解決にあたることが重要である。しかも，保守管理者は，開発済としてテイクオーバされるソフトウェアを保守の立場から全体像を眺めることができる立場にある。つまりどこに問題点があり，開発サイドで十分解決しなければならない問題を放置して保守工程にパスさせているなど，ソフトウェアにまつわるあらゆる問題点を見ることができ，ソフトウェア問題の解決をはかる提案ができる立場にある。そこで積極的に開発部門に問題点などをフィードバックする必要がある。このことが保守問題解決に対する重要な役割を果たすことになるので，管理者は問題点を担当者まかせにしないで，この問題に積極的に参与していく必要がある。

また，保守作業を担当者まかせにしている限りでは，それほど大巾な生産性向上は望めず個人の持っている能力によって生産性や品質が左右されてしまうことになるので，管理者は，保守業務がどのように行われ，しかも，いかなる保守要求などが多いかなどを保守情報データベースとして保持し，要員計画，作業計画などの計画をたてる時に積極的に活用していく必要がある。この意味において保守業務の中でも管理者が持つ役割は重要であり，管理者が活躍出来る領域はかなり広いものと思われる。

さらに緊急度の高い保守要求に対しても，担当者まかせにするのでなく，積極的に参与し，何故そのような問題が発生したか，それを解決するためにどの位の人間が，どの位の工数を必要とするかなどのデータをしっかりと把握する必要がある。これらの問題点を発生の都度，確実に解明しこれらを着実に積み重ねることが，保守問題の解決と生産性を向上させることになる。

つまり保守作業での管理項目を明確にし，それらを出来るだけ定量的に把握するように心がけ，保守情報のデータベースの充実化をはかることが，今後の重要な課題になるものと思われる。

1.3.3 保守作業固有の問題点

(1) システムの陳腐化

現在，一般的に行われている方法論からシステム開発とシステム環境を眺めてみると，システム環境は常に変化しているのに対してシステム開発の立場では，システム化ニーズや環境といったシステム環境を意識的に一時期止めて開発を進める方法をとっている。そのためにここに大きなシステムを陳腐化させる原因を含むことになる。

つまり現在行われているシステム開発の方法論ではシステム化ニーズを持っている利用者の要求仕様を開発時点のある時期にフィックスする必要がある。そのために開発期間が長期化すればするほど，要求仕様を凍結した時点から開発完了時点までの期間が長期間化し，その間にシステム環境やシステム化ニーズにも変化が発生し，システムの移行過程においてシステム修正を実施しなければならないといった事象が発生する。このようにシステム環境やシステム化ニーズは絶えず変化しており，システムの陳腐化を避けるためにも，保守段階の重要な作業として，いかに迅速にシステム修正を実施するかである。

そのうえ最近のようにシステム規模が大きくなり，利用者層も拡大してくるとシステムに対する修正要求は一段と多くなる傾向がある。それに対して保守要員はそれほど多くを期待出来る状況にない。そのために開発案件のバックログのみならず保守要求案件のバックログ化を助長し，今後大きな問題に発展する可能性がある。この意味において保守作業の生産性を向上させシステムの陳腐化を防止することが，今後の重要な課題になってくるものと思われる。

しかも，ソフトウェアの性質上，環境変化に対応した保守を正確に実施しておかないと，全く使い物にならなくなってしまうので，保守要求や環境変化の情報がスムーズにパスされ，システムの保守が行われる仕組を確立しておく必要がある。

それに対してハードウェアの場合は，法的に償却年限が決まっていたり，あるいは，年限が経過すると錆が出たり，型が古くなったり，性能が落ちたり，故障しがちになったりして，何んとなく感覚的にシステムライフがきていることが分ってくる性質がある。

これに反して，ソフトウェアの場合には，保守要員がしっかりとした保守を

やっている限りにはシステムライフがきたと明言出来るだけの材料がなく，いつまでも保守し続けることになる。その結果保守費用が増大してくることになる。ソフトウェア保守での問題として，どのような評価基準で，新らしいソフトウェアに置き換えるべきかが決定出来ない点である。そのために十数年前のシステムが相変らず動いていて，そのシステムから出力されるアウトプットが何んとなくエンドユーザに届けられ机の上に放置されるといった事態も発生することになる。

そこで保守作業での迅速な対応と同時にシステムの陳腐化基準を明確にし，いかなる状態になつた時に新システムに置き換えるべきかを判断する評価工程を運用工程の中にしっかりと確立し，システムのライフサイクルを通したシステムの経歴管理を確立しておく必要がある。

(2) 要員問題

① モチベーション

ハーツバーグはピッバーグ地方の知識労働者（会計士，エンジニア）を対象にいかなる要因が仕事を遂行する上での動機づけになるかの研究を行っている。彼の説によると知識労働者が仕事に満足を感じるのは，やる気といった意欲を増進させる促進要因とやる気の低下を防止する歯止めの役割をする衛生要因とがある。

㋑ 促進要因：仕事の達成，達成に対する評価，仕事に対する責任，昇進など

㋺ 衛生要因：会社の経営政策，管理・監督の仕方，対人関係，作業条件など

これらの中で，特に衛生要因は，不満足感を一時的に防止するが，しかしこれは必ずしも長期的な動機づけにつながらない要因として位置づけている。

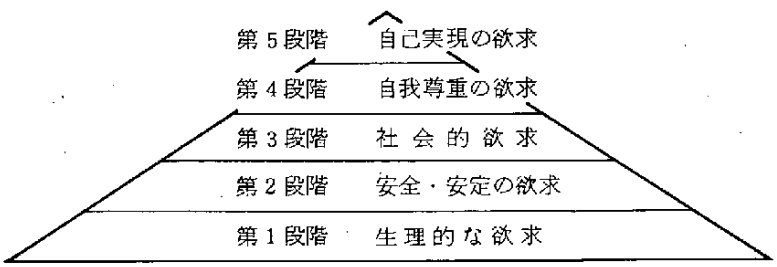
そこでこのハーツバーグの理論をベースに保守要員の置かれている状況を眺めてみると，企業側がいくら保守業務の重要性を力説したとしても，企業側が無意識的にとっている開発業務への傾斜度を保守要員は敏感に感じとってしまい，二義的な効果しか持っていない衛生要因までも駄目にしてしまっているのが実情である。

つまり保守要員のモチベーション問題を解決するためには，まず第一番目に不満足感を助長させている衛生要因をしっかりと充足させること

が重要であり，それと同時にモチベーションを非常に高めるといわれている促進要因（例えば仕事の達成に対する評価を公平に行うこと）を高めるような施策をうつことが，保守要員のモチベーション問題を大きく解決する基礎的な手段と思われる。

またマズローは，「人間を行動にかりたてるもの」として要求の5段階説を唱えている。この中でマズローは，「人間は誰れしも人から認められたい，人からほめられたい」という欲求が強く，これをかなえられると人間は，非常にやる気を出して働くとしている。この説をかりるならば，保守要員でも開発要員と同様に自己の行った仕事に対しても十分な評価をしてもらいたいと思っているにもかかわらず，保守要員が行う作業に対しては，比較的に冷淡で，「うまくいってあたりまえ下手すると，徹底的に文句を言われる」といった行動を管理者にとられる場合が多かった。このことは第4の要求である「自我尊重の欲求」をとことん傷つけることになり，大きくモチベーションを阻害していたことになる。この問題の解決は，前述したように保守要員の行う業務に対して，開発業務よりも比重を高くした評価を行い，彼らの行う作業自体を日の当る場所に引き出してやることが重要である。参考までにマズローの要求5段階説を図表1－5に掲げる。

図表1－5　マズローの欲求5段階説



② ローテーション

保守業務には，ソフトウェア，ハードウェアに関する高度な技術や知識が必要とされている。しかし実際には，技術力のあまり高くなく，しかも業務知識に乏しい，比較的若い人が振り向けられる場合が多い。そこで，一人前の保守要員にするために長期間をかけOJTで教育する場合

が多い。その結果投入当初は，ミスが頻発し問題になるケースが多い。このように保守業務を習得させるのにＯＪＴが重要な手段として採用されている割には，開発要員の教育ほどには，保守要員の教育には，いかなる体系で，いかなる教育をするかが明確になっていない。

そのためにモチベーション問題の部分的な解決を兼ねて，開発要員をそのまま保守要員として残こし，そこに新らたな経験の浅い要員を投入してＯＪＴ的に教育する場合が多い。

このように保守要員を一人前にするには，かなりの年月を必要とするので，一旦保守要員として養成してしまうと，管理者はリスクを恐れて，システム規模が大きくなればなるほど，核になる保守要員のローテーションには消極的となる傾向が強い。

このローテーション問題を解決するために，何よりも大事なことは，保守技術の明確化と保守要員育成のためのカリキュラムを確立し計画的な教育が必須条件と思われる。さらに，これに加えて，保守業務に必要な業務知識の吸収を計画的に実施しておく必要がある。

つまり，管理者が安心してローテーションに踏み切れるような仕組みを企業の中で組織的に確立しておくことが，ローテーション問題を解決する鍵となるものと思われる。

1.4　運用・保守の高度化のための基本的な考え方

1.4.1　運用問題解決のための基本的な考え方

運用という作業は，一般的には軽く考えがちであり，しかもこの考え方の背後には，運用は単純な作業であって，それほどたいした技術力は必要としないであろうという考え方が隠されているためと思われる。このことは，確かにある一面では，そう考えても良い業務もある。しかし前述したように運用業務の領域は，かなり巾広く，しかも最近ではかなり技術力を必要とする領域も出現しており，運用問題はそれほど簡単には考えられなくなってきている。

つまり，昔の情報システムは，省力化を主目的にした現状の処理の置き換え的なシステムが多かった。それに反して，最近のシステムは，規模的にも大きく，複雑化しており，全く新らしいシステムの創造となっている場合が多い。このようにシステムが持つ優劣や情報システム部門の体力が，そのまま

企業の競争力を左右するといったように，企業経営の中枢的な役割を果たすようになってきている。そのためにシステム運用上のトラブルが企業の死活問題にまでに発展する要素を含くんでおり，それだけ一層運用効率が高く，信頼性，安全性の面に十分注意が払われた運用を要請されてきている。

そのうえ従来のシステムでは，対象範囲が自社内だけにとどまっている場合が多かった。ところが，最近の著るしいソフトウェア／ハードウェア技術，ネットワーク技術の進展によって，企業間システム，あるいは垂直統合的な流通システムなどによって，色々な企業の要員が同一システムの運用に関係したり，色々な階層の人が運用要員としてシステムの系の中にとり込まれてきている。この場合，システムの内容がいくら複雑であってもシステム運用の安全性，信頼性を増加させるために出来るだけ運用の現場はシンプルにする必要がある。

その一方，コンピュータ室内部の運用を考える場合，バッチ主体型の運用か，オンライン主体型の運用かによって解決方法が異なる。オンライン主体型の運用の場合には，自動化をシステム自体のなかでかなり構えているので，省力化がかなりはかられているが，この場合といえども，バッチ処理業務をオンライン化の支援作業として夜間などに稼動させる必要があり，この部分の効率化が大きな課題となっている。

そこで運用の高度化・効率化をはかるための基本的な考え方であるつぎの項目について詳細に検討してみる。

① 運用の無人化，自動化
② 品質管理の導入
③ 評価の実施

(1) 運用の無人化，自動化

無人化を達成するための一つの手段として，システムのオンライン化があり，しかもユーザの利便性という観点からますますオンライン・アプリケーション・システムは拡大の一途を辿っている。一般的にオンライン・システムは，その設計思想としてセンター・マシン側での人手の介入を出来るだけ排除するといった考え方で構築されており，その意味において運用の効率化・高度化にかなり役立っている。

しかし，オンライン・システムが企業の中にかなり浸透していっても，オンライン・システムを陰で支えているシステムとしてバッチ・システムが相変ら

ず残っており，しかも業務内容によっては，オンライン・システムよりもバッチ・システムの方が，はるかに採算的にも良い場合があるので，バッチ処理自体の割合はかなり低下していくものの，今後かなりの部分残っていくものと思われる。

そこでこの人手を比較的多く必要とするバッチ・システム運用に対して，人間でしか作業出来ない領域を残して，他を出来るだけ機械化するように，自動化，無人化の考え方が運用工程の色々な所に導入されてきている。

この無人化，自動化を進めるためには，コンピュータ室の業務分析や，対象となるアプリケーションのＩ／Ｏ要件を分析し，事前に業務処理の標準化を実施しておく必要がある。

つまり無人化や自動化を推進するためにはエンド・ユーザや開発要員を含くめて，運用問題を十分認識させて，標準的な運用方法に対するコンセンサスを得ておく必要がある。

また運用領域というのは，非常に泥くさく自動化用の各種のハードウェア，ソフトウェアを導入したからといって，大巾な省力化が達成出来るのではなくて，しっかりとした方針のもとでステップ・バイ・ステップ方式で運用要員を新らしい環境に徐々に慣れさせながら，着実にレベルアップをはかっていく方法が大きな混乱なしに実現出来ていくものと思われる。この自動化，無人化をはかる場合の一般的な手順としてつぎのような方法が考えられるが，運用要員の技術的なレベル，エンドユーザのシステム化レベル，ハードウェア環境などといった環境を十分認識して，自社の環境に合致した手順を確立する必要がある。

① 運用環境の整備：業務運営の標準化，例外処理のルール化など
② 省力化を目的としたハードウェア，ソフトウェアの導入：ＭＳＳ，ＬＰオートカッター，ソフトウェアの導入など
③ 防災，障害管理の確立：障害パトロール機，障害情報の自動収集など
④ リスタート機能の確立：自動ＩＰＬなど
⑤ 網管理の確立と自動化ＩＰＬの作成：自動運転用のプロトコル，停止予告など
⑥ 統合化：最終的には無人化を指向する。しかし，この場合といえども算について十分考慮する必要がある（人間を直接残しておいた方が良い場合もある）

(2) 品質管理の導入

運用作業の場合，比較的個人的な作業活動が多く，どうしても運用ノウハウが個人に蓄積し，属人化する傾向がある。そのために管理者の参画が困難となり，個人にまかせがちになってしまう。その結果個人の能力や性格が運用作業に大きな影響を与えてしまい，ケアレスミスが頻発したり，同じような誤りを繰り返しておかしたりして，大きな問題にまで発展する場合がある。

そこで運用作業にも品質管理の考え方を適用し，安全性，信頼性，正確性といった運用品質に関係深い要素の高度化をはかる必要がある。つまり運用作業といった個人的な色彩の強い作業に対して，工程分けの概念を導入し，それぞれの工程でいかなる作業を実施するかを明確にし。その作業目的は何か，他工程との関連はどうか，工程内での順序づけはどうかなどを明確にし，しかもそれぞれの工程内の作業を実施するにあたっても，計画，実施，評価といったデミングサークルを廻し，最終的にはより質の高い運用が出来る仕組を構築する必要がある。

また，運用工程を明確にし，しかもＰＤＣＡといったデミングサークルを廻すことによって管理者もこの運用業務に参与することができ，徐々に運用品質を高めるように管理の側面を強化していくことが出来るようになってくる。勿論，運用工程の中には遂次処理的な作業ばかりがある訳でもなく，どちらの作業を先行しても構わない作業も多い。しかも同時期に併行的に実施しなければならない作業も存在している。この場合は，とに角，理論的根拠がないので，感覚的に決定してもそれほど問題はない。要は，作業の位置づけや目的が明確になっていれば良く，あまり細かいレベルにこだわって議論をしていると，運用品質をいかにして高めるかという本来の目的を見失う恐れがあるので，ある程度のみきりが必要となってくる。

このようにして作られた運用工程，ＰＤＣＡを廻転させるための仕組み，運用情報のデータベースの構築などを活用することによって人手で行っている運用領域の運用品質が徐々に高まり，これが完全に業務の中に根づくことにより運用の高度化・効率化が達成出来るものと思われる。

(3) 評価の実施

ソフトウェアの品質を高めるために，開発工程では，工程の終了段階毎にレビューを実施し，出来るだけ次工程に問題を持ち越さないような努力

を行っている。運用工程でもこのような考え方を採用して，運用によって作り出される情報の品質を高めるために品質管理の考え方の導入を進めた。

しかし，これらの考え方は開発，運用にしろ出来上って運用されているアプリケーション・システムが当然有効なシステムであるといった前提で議論が展開されていて，このアプリケーション・システムが本当に有効であるか否かについては触れていない。

そこでちょっと議論の視点を変えて，アプリケーション・システムの有効性に関する評価について眺めてみることにする。つまりここで述べている評価とは，運用段階にあるソフトウェアが実際に利用者に役立っているか否かを探るためのものである。

従来のシステムは，どちらかというとコンピュータ部門主導型で作られており，しかも多額の投資の割にはそれほど実効が上がっていないのではないかと経営者も内心判然としないながらもシステム化の推進に承認を与えてきていたのが一般的な姿ではないかと思われる。

しかし，最近のように戦略的な情報システムが経営の中枢の中に入り込んでくると，今までのように良く分らないのでコンピュータ部門にまかせておけば良いといった考え方の経営では，経営が成り立っていかなくなってきている。そのために経営者の方も提案されたシステムが企業経営の運営の中でどれだけの効果を発揮しているかを知りたいといった要望が強くなっている。つまり，経営者側も開発後の評価実施にかなり強い関心を持つようになっており，そのうえこの評価を実施することによって，どこに問題点があり，それを解決するにはどうしたら良いかなどが明確になり，これが開発部門のノウハウとして蓄積され，技術力向上の源泉となってくる。これらの情報を開発部門にフィードバックし次回の開発に役立てることが出来るので，この評価工程は，システムのライフサイクル論の中でも重要な役割を果たすことになる。

評価作業の実施にあたって重要なことは，評価工程をソフトウェアのライフサイクルの中に明確に位置づけて，これを一過性的に実施するのでなく継続的に実施することである。確かに評価方法に対する方法論が確立しておらず，評価の実施はかなり難かしいのが実情であるが，自社の環境条件に合った方法論を策定し，試行錯誤的に実施し，とにかく評価作業自体

を定着化させることが重要である。また，運用段階では各種の情報を収集することができ，この評価工程では，運用情報のデータベースを多方面にわたって分析，検討し，管理や改善のために活用する必要がある。またこの場合，大事なことは，構築されているデータベースは必ずオープンにし関連部門にフィードバックしてやる仕組を作ることが重要である。

1.4.2 保守問題解決のための基本的な考え方

開発の生産性を向上させるために，古くからさまざまな試みがなされ，新しい技術や考え方が導入されている。しかし，そのわりには，まだまだ不十分な点が開発に必要な各種の技術に散在している。とはいうものの，開発問題に関してはかなりの研究開発がなされ，かなり充実化されてきていると思われる。これに比べて保守環境は，開発環境のようにほとんど整備され体系化されていないのが実状である。例えば市販ツールにしてもほとんどが開発用ツールであって，保守用ツールはそれほど提供されていない。

このことは，ソフトウェアの開発工程に対して目が注がれ，保守工程にはそれほど目が向けられなかったためと思われる。しかしソフトウェアのライフサイクル全体を通してのコストを眺めてみた時に，保守コストが全体の費用の6割を占めているといった論文が公表されるにおよんで，保守の生産性向上の必要性が強く叫ばれるようになってきた。

我が国でもこの保守問題の重要性に着目して，昭和56年から5ヶ年計画でソフトウェア保守技術開発計画（SMEF）プロジェクトが通商産業省支援のもとに協同システム開発㈱を中心に進められている。

このプロジェクトの目的が，保守のための実践的，工学的方法論および総合的な保守支援環境の構築技術を確立することとしており，その中で保守技術の確立，各種の保守支援ツールの提供，保守環境の整備などが重要なテーマとなっている。このSMEFプロジェクトによって，多くの注目が保守工程に注がれ，保守問題におけるさまざまな問題が解決されていくものと思われる。

この場合，単に保守工程だけの生産性向上を狙うのでなく，開発段階から保守工程までを含んだ，ライフサイクル全体の生産性向上をはかる必要がある。

本来，上流工程でしっかりとした作業が実施されていれば，保守工程でそれほど多くの問題点は発生してこないはずであるが，しかし現実には，開発

段階での問題点がそのまま保守工程に持ち込まれる場合が多い。

そこで保守問題を検討するには，理想論をベースに検討するのでなく，現状をある程度是認した上で，保守問題の解決に取り組む必要がある。そのために開発段階でとられる生産性向上などと基本的には同様のつぎの諸施策を保守工程でも採用する必要がある。

① 標準化の推進と教育の充実
② ツールの活用
③ 組織体制の確立
④ 管理機能の強化

これらの施策について順次概説する。

(1) 標準化の推進と教育の充実

緊急を要する作業の発生によって，保守作業の実行過程では，工程といった概念が全く打ち捨てられ，どんぶり勘定的に作業が行われる場合が多い。

しかし，保守作業を原因別に分けた統計資料を眺めてみると保守作業全体の中で，緊急性を要する作業は，わづか2割にしかすぎない。このことは，保守作業の属人化を利用して，保守要員が保守作業の緊急性をイメージ化させ標準化ルールとして設定されている基準を遵守しない理由づけに悪用しているきらいがある。

すべての作業に対して絶体に標準化ルールを守るべきだといった固定的な考え方に固執している訳ではないが，原因別作業量の統計資料から保守作業を分析してみると，標準化に沿った作業を実施する時間的な余裕は十分あると考えられる。

しかし，現実には，標準化ルールがあまり遵守されていないのが実情であり，このことがさらに保守生産性の向上を阻害する大きな要因ともなっている。

そのために保守工程での標準化推進の重要なキーポイントは保守要員の意識改革と思われる。またこの意識改革と同時に，標準化を遵守しなければつぎの作業工程に進めないといった仕かけを作ることも重要な課題でもある。

また，保守工程の標準化のなかで一番遵守されない作業の1つとして，文書化作業があるが，最近では，ドキュメントの自動生成ツールなどが市販され省力化がはかれるようになっている。しかし，ここで大事なことは，

保守用のドキュメントとして何が一番有効な働きをするかをじっくり見定めることである。それによって，市販のツールを採用するか，自社独自で開発するかの意思決定を行う必要がある。また，さらに保守用ドキュメントとしてどの程度が必要であるかを自社の技術レベルなどを参考にして設定する必要がある。

つぎに，保守段階での標準化推進での重要な作業項目について記すと，

① 担当者教育
② 保守支援ツールの整備
③ 標準化ルールを遵守させるための仕かけ
④ 標準化遵守状況のチェック

などがある。

(2) ツールの活用

前述した日本の国家的プロジェクトであるSMEF プロジェクトではツールの体系をつぎのように設定して開発を進めている。

① 性能評価支援系ツール：性能モニタリング，信頼性分析，性能余波分析など
② 分析支援系ツール：要求分析，ソースプログラム分析など
③ 変更支援系ツール：再構造化，コードオーディター，コードコンパレーターなど
④ テスト支援系：テストデータの生成，管理，テスト結果分析など
⑤ 文書化支援系ツール：保守作業用ドキュメント，プログラムドキュメントなど
⑥ 変換支援系ツール：プログラム変換，ファイル変換など
⑦ 管理支援系ツール：製品監査，配布管理，改訂管理など

このプロジェクトでのこれらのツールは，あくまでも保守支援ワークステーション上におけるツール群として持つべき機能を一覧表の形式に取りまとめたものである。またこれら個々のツールは，独立のパッケージとして存在するのでなく，オンライン会話型での保守環境においてそれぞれのツールを単純化してツールキットの形で保持していて，必要な時にそれらのツールを適当に組み合せて，より複雑な作業が実行出来るような仕組にしている。これが完成した段階ではツールとしてもかなり使い易いものになると思われる。

現在市販されている多くの保守用ツールは，保守段階で大きな問題となっているドキュメントを自動作成するツールが多い。つまりこの事は，考え方の発想を逆にして，保守作業の実行中に十分なドキュメントを作成する余裕がないならば，保守作業終了後ソースプログラムから逆にドキュメントを自動生成させることによって大巾なドキュメント化作業の軽減化をはかることを目的としている。この種のツールの需要が高く，それに対応してメーカ各社からつぎのようなツールが開発，販売されている。例えばCOBOL 用として日立ソフトウェアエンジニアリング㈱のＨＩＤＯＣ,富士通㈱のＳＮＯＴＥ,日本電気㈱のＣＯＢＯＬ／ＳＤＡ，ＦＯＲＴＲＡＮ用としてＣＲＣのＳＵＦＯＲＴなどがある。

一般的にいって，ツールの活用は当該業務の生産性向上に大きな貢献をするが，これには大きな前提条件があって，自社の環境に十分マッチすることが必要となってくる。しかも保守環境は開発環境と同様に各社毎にかなり異なっているので，自社環境にマッチするツールを選択することが重要である。

あるいは，自社の保守工程の中でどこの工程が一番問題であるかを明確化し，自社の保守環境に合致したツールを開発し，活用することも重要である。

そのうえツールを購入したり，開発する場合単にある工程だけの省力化を目的にするのでなく，保守工程全体を見透し，全体的な生産性向上につながるようなツールにする必要がある。しかしその中で，特に保守作業では，作業結果が直ちに本番作業に影響をおよぼす場合が多いので，作業結果の品質を保証するような，分析支援系やテスト支援系などのツールの提供が強く要望されている。

(3) 組織体制の確立

日本システミックス㈱が，昭和５９年に実施したアンケート調査によると，開発部門と保守部門の分離状態はつぎのようになっている。

システム開発を担当するグループと別組織になっている	18社	18%
システム開発を担当するグループと別組織になっていない	79社	79%
その他	3社	3%
	100社	100%

またシステムの保守を担当するグループに対して組織的な面でどのような方法で改善していくかのアンケート調査を実施している。

組織体制＼将来方向	現　状	将　来
保守と開発を分離する	18%	34%
保守と開発を一本化する	79%	31%

つまり，現時点では開発グループと保守グループがそれほど明確に分離されていないが，今後の方向性としては開発と保守を分離する傾向を示している。しかしこれだけからでは，開発グループと保守グループの分離が，コンピュータ部門の生産性向上につながるか否かは明確ではない。

ただし，一般的には，ソフトウェア規模が大きくなってくると，分離して保守業務を専門的に行った方が要員の効率的な活用という観点から考えると望ましい方向といえる。また，保守グループを独立させることによって保守問題で発生する諸々の問題が保守部門自身にふりかかる問題として，保守の生産性向上に真剣に取り組まざるを得なくなってくる。

さらに重要なことは，開発部門と保守部門を明確に分離したあと，両部門のコミュニケーションをどのように緊密にし，協力体制を築いていくかである。良く発生する事象として両部門の分離によって，セクショナリズムが台頭し開発されたシステムが色々な問題を含くんでいるために宙ぶらりんになって，どちらの部門が保守するか不明確なまま開発担当者が苦労するといったケースもある。この場合は，明らかに自部門のエゴのみをぶつける結果発生する問題であって，コンピュータ部門全体の生産性向上とい

う視点にたてば，自から解決出来る問題であって，その意味では開発部門と保守部門との間の情報をスムーズにパスさせる仕組みの構築が重要となってくる。

(4) 管理機能の強化

ソフトウェア開発は，同一人あるいは同一チームによって，保守期間に比較すると比較的短期間で開発が行われる。これに対してソフトウェアの保守期間は段々と長期間化し，その間には，担当者が変わったり，大巾な修正が加えられたりして，開発段階でのソフトウェアと大巾に機能面で異なったりしてくる。このようにソフトウェア環境は流動的であり，しかもこのソフトウェアの持つ機能レベルを継続的に維持あるいはレベルアップしていかなければならない。そのために，ソフトウェアの保守を担当者まかせにするのでなく，管理という側面からしっかりと管理を実施する必要性が生じてくる。

つまり，保守作業自体の性格上かなり保守担当者だけのクローズドな世界で対処可能であるが，この場合，保守要員の能力がそのまま作業結果に現われ，信頼性，効率性に直接影響をおよぼすことになる。そこで要員に出来るだけ依存しないで，ある程度均質な作業結果になるような仕組みが組織上必要となり，しかも，適切かつきめの細かい管理が要請されてきた。

また，当該ソフトウェアに対する保守作業の長期間化は，保守要員の記憶の不正確さを増大させ，そのうえ担当者のローテーションによる技術移転の減少化を招き，効率性を低下させることになる。このような問題の解決手段の1つに文書化があり，保守担当者の作業の中で遵守されにくいこの文書化をしっかりとした管理業務のなかに組み込む必要がある。

つまり保守業務の効率性，信頼性，正確性を向上させるためには，つぎのような管理業務を継続的に実施，強化していく必要がある。

① 保守作業そのものの管理
② 保守工程の管理
③ 受入れ検査
④ 改訂管理（ドキュメント，ソフトウェア）

これと同時に重要なことは，保守工程にかかわる各種の情報を収集し，蓄積しておく必要がある。このデータベースを十分分析し問題点を明確に

し，それらの情報を開発部門や運用部門にフィードバックしてやることが，保守部門だけの効率化にとどまらず，ソフトウェアのライフサイクル全体を見透した，高度化・効率化につながるので，今後この保守業務に関連する保守用のデータベースの充実が大きな課題になってくるものと思われる。

1.4.3　運用・保守の共通的な対策

情報化の進展につれて，ソフトウェアニーズは増大の一途を辿っている。しかしこれに対して，この需要を消化すべき開発部門の生産性は，それほどの向上を示していない。このような状況を踏まえ通商産業省では1990年には，約60万人のソフトウェア技術者が不足するだろうと予測している。このソフトウェア・クライシスと呼ばれる大巾な需要と供給のギャップを埋めるべき国家的なプロジェクトとしてソフトウェア生産工業化システム（通称シグマ・プロジェクト）が昭和60年度から5ケ年計画で開始された。このようにソフトウェア産業全体で眺めてみても，ソフトウェア技術者の不足は，かなり深刻な問題となっており，一般企業の場合にはもっと悩みが深い。これを解消する方策として，従来から外注化が行われてきた。しかし従来の外注化業務は，自社要員の比較的やりたがらない業務や，一時に多人数の要員を必要とする場合の業務などを外注化するのが一般的な姿であった。

これに対して外注先自身も需要の増大に対応して，仕事の遂行をとおして出来るだけ良い成果が自社に蓄積出来るような仕事を選ぶといった，仕事の選択の時代になっており，外注化も必ずしも簡単ではなくなってきている。また，このような需要と供給のアンバランスのなかで，バックログを解消するためには多くのソフトウェアを出来るだけ短期間で開発していかなければならず，開発されるソフトウェアの品質面に粗製濫造的な面が見られ，一方今後ますます高度化・複雑化し，社会活動のあらゆる局面に重要な役割を果たすソフトウェア開発をしなければならないといった方向性との間に，ある種の自己予盾が生じてきている。そこでこの予盾を出来るだけ正しい方向にただし，高品質で信頼性の高いソフトウェアを提供出来るようにするための1つの手段としてシステム監査がある。この保守工程だけでなくソフトウェアのライフサイクル全体を通した品質保証をするために，最近ではシステム監査に注目が置かれ，通商産業省の認定試験のなかにこのシステム監査がとり入れられるようになってきており，しかも金融業では，かなりの企業がシステ

ム監査を実施するようになってきている。

また，保守や運用業務の生産性向上を従来と全く異なったアプローチで解決をはかっているケースがある。

つまり，コンピュータ部門の要員増がそれほど期待できず，例え増員したとしても開発要員に振り向けられる場合が多く，そのために保守業務自体にもバックログ化する傾向が出てきている。そこで，これを解消するためにコンピュータ部門は，プロしか出来ないようなスキルの高い業務だけに特化し，エンドユーザでも十分使えるようなツールや環境を整備，提供し，他の業務を出来るだけエンドユーザ自身に肩代りしてもらい，企業全体のソフトウェアに対する処理能力（開発，運用，保守，コンサルティング）を全体として向上させるような方策を採用するようになってきた。

そこで，運用・保守業務の共通的な効率化対策の中で今後比較的重要になってくると思われるつぎの3種類の方策について概説することにする。

① 外注政策の確立

② システム監査の導入

③ ＩＣの導入と４ＧＬの導入

(1) 外注政策の確立

前述したようにソフトウェア技術者の不足はかなり深刻化しており，従来のように自社要員の下働き的な作業だけを発注するような方法は採れなくなると思われる。しかも傾向として下働き的な作業である単純作業などは，ツールの導入や自動化などによって段々と減少してきており，領域としてもかなり狭ばまってきている。そのために運用工程や保守工程でもかなり高度な判断力がいる作業や技術力のいる作業領域が人手作業として残るようになる。

そのために受注者側も単に経験の浅い要員を派遣するといったやり方では，発注先の需要を満足することができなくなり，要員に対する自社内での十分な教育の実施や，長期的な視点にたった育成計画などが必須になってくるものと思われる。

一方発注者側は，供給側の能力を十分評価出来るだけの体制や技術力を確立する必要がある。このためには，発注者側も従来よりもまして要員の育成計画や人事ローテーション計画を綿密に確立する必要がある。これらの計画を踏まえて，自社ではソフトウェアに関連する業務（開発，運用，保

守，コンサルティング，管理，企画など）のなかでどの機能を受け持ち，どの業務を外注に委かせるといった機能面からの分業化政策といった外注政策を確立する必要がある。

これによって外注先は，これらの機能分担と保有している要員の技術力などを勘案してどこに魅力があるかなどを判断して，より前向きなパートナーとして参画出来るような仕組みを期待してくるものと思われる。

そこで発注者側も，責任範囲を明確にし，仕様書や契約条件などを明示して発注する必要がある。つまり，このような仕組みを作ることが発注側や外注側の技術力を高めることになり，ひいては外注先も含くんだ全体としてのソフトウェアに係る生産性の向上につながっていくものと思われる。

(2) システム監査の導入

㈱日本情報処理開発協会発行のシステム監査基準解説書の中でシステム監査についてつぎのように述べている。「システム監査は監査対象から独立した監査人が情報システムを総合的に点検，評価し関係者に助言，勧告するものであり，セキュリティ対策の実効性を担保し，費用効果の高いセキュリティ対策を可能にする上で極めて有効な手段である」としている。また，さらに「システム監査は情報システムの品質を保証し，かつ経営の観点からみて，十分効率的であるかを評価する機能を担うものである」としている。ここでも述べられているようにシステム監査の重要なポイントは，監査対象から独立した監査人が監査を実施する点である。

つまり，従来は経営者があまりコンピュータ業務内容を理解出来ないために，ある程度コンピュータ部門に"委せておけ"的な放任主義の結果，かなり閉鎖的な世界を作り出し，一般の人々を近づけにくい場所にしてしまった。そのために，しっかりとした管理の習慣が出来ず担当者まかせになりがちであった。このことが属人化傾向を助長させ，品質面がなおざりにされる結果となってしまった。

しかし，経営に密着した情報システムが構築されるようになってくると，コンピュータ部門の位置づけがかなり高まり，経営の中枢機能を果たすようになってきた。この結果，従来のようにコンピュータ部門がブラックボックス的な存在ではあり得なくなり，企業経営の中心に引き出され，コンピュータ部門自体の活動に企業全体からの注目を浴びるようになってきた。

そこで，コンピュータ部門のアクティビティを詳しく監査し，しかも，情報システムをより有効なシステムにするためにシステム監査の手法が導入されるようになってきた。特に運用や保守工程では，日常の忙しさを理由に本来やるべき手続きを踏まずに作業を実施している場合が多く，この事が結果として最終的に品質や生産性の向上に悪影響をおよぼし，信頼性や安全性を低下させる原因ともなるので，第三者によるシステム監査が，これらの問題点の歯止め防止の有効な手段になるものと思われる。アメリカでは既に１９８１年からシステム監査の試験制度がプライベートであるが導入されており，しかも発注先の契約条件の中にシステム監査の実施を義務づけている企業もあり，今後一般企業や情報産業を含くめてシステム監査の実施が日常的なものになってくると思われる。

(3) 情報センター（ＩＣ）と第４世代言語の導入といった組織体制の変化

前述したように，開発ニーズや保守ニーズの増加によって，開発や保守のためのバックログが２～３年にもなってくると，エンドユーザ自身がそんなに長期間待っていられなくなり，コンピュータ部門の手を借りずに自分自身で何んらかの対策をとるようになってきた。そのなかの１つがパソコンの導入であり自分たちで積極的にプログラムを作り，自分の必要とする情報を自分の手で作るようになってきた。この場合，問題となるのは，コンピュータ部門といった何十年の経験を持ったプロの手から統制がはずされてくると，初期の段階でコンピュータ部門が散々経験してきたいろいろな問題をエンドユーザ自身が味わうことになる。

これでは企業体としては必ずしも効率的でないので，エンドユーザ自身が自分の手で自分の必要とする情報を作りたいという要望をある程度満たしながら，一方ではコンピュータ部門のコントロール下に置くための仕かけとして，情報センターという組織をコンピュータ部門の他に設置して，運用，保守，開発を含めた生産能力の増強策が導入されるようになってきた。この情報センターの役割は，あくまでもコンサルティングに特化し，エンドユーザが自分自身の手で開発，運用，保守業務を実施するように指導する役割を負っている。

このように，エンドユーザが自分自身の手で開発，運用・保守する場合，コンピュータ部門に依頼して開発するより，かなり少ない要求で満足する傾向がある。そのうえ，エンドユーザの要求を自分自身で解決するために，木

目の細かい開発が出来，効率面からいってもかなり高い生産性を発揮することが出来る。

つまり，コンピュータ部門だけでは，もはや対処不可能になってきたこの段階において，エンドユーザ部門をもこのような情報処理のなかに取り込むことによって，かなりの情報処理作業をエンドユーザ自身に移管できるようになり，しかも，ある程度組織的な対応が出来るために，ソフトウェア問題に対するコントロールが出来るといった一石二鳥的なメリットを持っている。

ただし，この場合といえども単に組織として情報センターを設置すれば事足りるという訳にはいかず，この部門の運営を円滑にするためには，コンピュータ部門の十分な強力と第4世代言語といわれるようなツールを整備，提供することが必須条件となってくる。

これと同時にツールの利用や運用方法に対する教育を何ん度も根気良く繰り返して実施する必要がある。新人やローテーションによって職場の構成員は変化しており，少こしでも手抜きをすると折角のツールも台無しとなるので，継続的に教育をすることがこのような仕組みを定着させる重要な要件ともなるので，インストラクターの養成と同時に継続的な教育実施が今後の重要課題となってくる。

参 考 文 献

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

2）「自動運転システムの実態調査報告書」㈶日本情報処理開発協会，昭和５６年３月

3）「ソフトウェア保守技術研究会報告書」協同システム開発㈱，昭和５６年３月

4）妹尾稔「メンテナンス・ツール"ＭＳＳ"について」共立出版bit臨時増刊，１９８２，２

5）妹尾稔「応用ソフトウェア保守問題」共立出版bit臨時増刊，１９８２，2

6）「システム監査基準解説書」㈶日本情報処理開発協会，１９８５，８，３０

7）宮本勲「ソフトウェア保守の管理」ＴＢＳ出版会，１９８４，４，３０

8）「第2回バックログ／ソフトウェア生産技術利用実態調査」日経マグロウヒル社，１９８６，１，６

2. ソフトウェア運用・保守の現状

2. ソフトウェア運用・保守の現状

この章ではシステム運用・保守の現状をみるために，コンピュータ・ユーザのコンピュータ部門におけるシステム運用・保守の現状と課題について，調査結果に基づいて述べることとする。

調査はアンケート調査，企業への訪問ヒアリングによった。アンケート調査はコンピュータ・ユーザ600社を対象に60年９月１日から10月18日までの期間で行ない，１６５社（回収率２８％）より回答を得た。ただし個々の項目については回答数が異なるので，その数字を各図表に示した。訪問ヒアリングは調査終了後，その補足調査としてアンケート回答企業の中から選んで実施した。

2.1 ソフトウェア運用について

2.1.1 コンピュータ部門の規模について

(1) 平均要員数（６０年８月１日現在の状況）

コンピュータ部門の要員については派遣要員が男子社員の約半分を占めている。また運用担当者についても同じ比率を占めている。一方運用担当者がコンピュータ部門の要員に占める比率についても約半分と，その役割は大きな比率を占めていることがわかる。（図表２－１）

図表２－１　平均要員数

区分	社員（人）		派遣社員（人）		合計（人）		派遣社員が占める割合(%)	
	男	女	男	女	男	女	男	女
コンピュータ部門の要員	59.9	24.1	25.6	5.2	85.5	29.3	29.9	17.7
運用担当者	29.3	10.5	15.9	3.0	45.2	13.5	35.2	22.2
運用担当者が占める割合（%）	48.9	43.6	62.1	57.7	52.9	46.1	—	—

(2) 要員数の分布

要員数の分布をみると，まず，コンピュータ部門の要員数から述べると１１人～２０人が全体１６５社のうち３７社（２２.４％）で一番多く，次に２１人～３０人が２２社（１３％）となっている。全体の５４.３％が１

人～４０人の範囲に入っている。一方派遣要員からみると，１人～１０人が比率としては断然多く，６７社（４０.６％）を占めている。この他は１人～２０人が１７社（１０.３％）程あるほかは殆んどバラついており，外部要員については１０人どまり，多くても２０人どまりという企業が約半分を占めている事がわかる。（図表２－２）

図表２－２　要員数の分布（コンピュータ部門）

区　分	社　員	派　遣	区　分	社　員	派　遣
1～10人	15社(9.1％)	67社(40.6％)	81～90人	5社(3％)	0社(0％)
11～20	37 (22.4)	17 (10.3)	91～100	3 (1.8)	1 (0.6)
21～30	22 (13.0)	5 (3.0)	101～150	13 (7.9)	6 (3.6)
31～40	17 (10.3)	6 (3.6)	151 以上	21 (12.7)	7 (4.2)
41～50	10 (6.1)	6 (3.6)	0 人	－	12(7.3)
51～60	7 (4.2)	4 (2.4)	無回答	－	32(19.3)
61～70	10 (6.1)	2 (1.2)			
71～80	5 (3.0)	0 (0.0)	合　計	151(100)	151(100)

つぎに，運用担当のみに限って同じようにみると，次の図表２－３に示すようになる。傾向としては社員については１～１０人が６０社（３６.４％，１１～２０人が２７社（１６.４％）で，両方で全体の５２.８％を占めている。派遣についてみてもほぼ同様な傾向を示し，１～１０人が６０社（３６.４％），１１～２０人が１８社（１０.９％）で，両方で全体の４７.３％を占めている。

図表２－３　要員数の分布（運用担当）

区　分	社　員	派　遣	区　分	社　員	派　遣
1～10人	60社(36.4%)	60社(36.4%)	81～90人	2社(1.2%)	1社(0.6%)
11～20	27 (16.4)	18 (10.9)	91～100	5 (3.0)	0 (0.0)
21～30	15 (9.1)	2 (1.2)	101～150	6 (3.6)	2 (1.2)
31～40	16 (9.7)	6 (3.6)	151 以上	8 (4.8)	3 (1.8)
41～50	9 (5.5)	3 (1.8)	0 人	5 (3.0)	19 (11.5)
51～60	2 (1.2)	5 (3.0)	無回答	4 (2.4)	41 (24.9)
61～70	3 (1.8)	4 (2.4)			
71～80	3 (1.8)	1 (0.6)	合　計	165 (100)	165 (100)

(3)　運用担当者のローテーション状況

運用担当者のローテーションは図表２－４に示すように全体の６５％が行っていると答えている。

図表２－４　運用担当者のローテーション状況

区　分	件　数	割　合
行っている	106	65.0 %
行っていない	57	35.0

N＝163

2.1.2　運用基準について

設定している運用基準についてみると，名前のつけ方の規定９５.１％，オペレーション手順を規定８５.２％，ジョブの運用を規定８８.３％，媒体・ファイルの管理を規定８４.６％などは比較的多くのユーザで実施されている。一方比較的実施が少ない項目としては，メンテナンス方法の規定４７.５％や稼動実績の収支管理の規定４１.４％などである。全般としては比較的運用規準は整備推進されているといえよう。（図表２－５）

図表２－５　運用基準について

設定している運用基準	件数	割合%
名前（利用識別名・ファイル名・プログラム名等）の付け方を規定している。	１５４	９５.１
ジョブの運用（ジョブ制御言語・ジョブ入出力ジョブの実行等）を規定している。	１４３	８８.３
オペレーション手順を規定している。	１３８	８５.２
媒体・ファイルの管理を規定している。	１３７	８４.６
運用形態・運用時間を規定している。	１３０	８０.２
プログラムの管理方法を規定している。	１２８	７９.０
スケジューリング・手続き等を規定している。	１１８	７２.８
ドキュメント管理を規定している。	１１７	７２.２
各資源（CPU，メモリ，周辺装置，ファイル等）の配分や限度を規定している。	１０６	６５.４
利用者管理（機密保護，アクセス権，パスワード等）を規定している。	１０６	６５.４
本番受入れ基準を規定している。	８３	５１.２
メンテナンス方法を規定している。	７７	４７.５
稼動実績の収支管理を規定している。	６７	４１.４
その他	２	１.２

N＝１６２

2.1.3　システム監査について

システム監査は比較的未実施のユーザーが多い。図表２－６がアンケート調査の結果である。これによると全く行っていないユーザーが４３.１％と全体の半分近くを占めている。システム監査は将来必要だとする回答も６７.５％を占め，いずれにしても今後の大きな課題である。なお社内の専門家が定期的に監査しているユーザが１０.６％となっており，次第に定着をしつつあるといえる。

図表２－６　システム監査について

項　　目	件　数	割合%
外部のコンサルタント（専門家）が定期的に監査している。	１４	8.8
内部（社員）の専門家が定期的に監査している。	１７	１０.６
外部の専門家が不定期に監査している。	２４	１５.０
内部の専門家が不定期に監査している。	２０	１２.５
システム監査は全く行っていない。	６９	４３.１
システム監査は将来必要と思う。	１０８	６７.５
システム監査は必要と思わない。	３	1.9

N＝１６０

2.1.4　安全対策について

(1)　通商産業省の安全対策基準について

通商産業省の安全対策基準の認定を受けているのが１１.３％となっており，あとは受けていないものが大部分を占めている。受けていない理由としては「必要を感じない」が３９.４％と一番多く，またこれと殆んど同じ割合で「実施にはお金がかかりすぎる」３８％がある。また「知らなかった」が１０.６％もある点も考慮すると，今後一層ＰＲし徹底していく必要性が感じられる。（図表２－７）

図表２－７　通商産業省の安全対策基準実施状況

通商産業省の安全対策基準の認定を受けている	１８件	１１.３％
〃　受けていない	１４２	８８.７

N＝１６０

受けていない理由	件　数	割合%
必要を感じていない。	５６	３９.４
実施にはお金がかかりすぎる。	５４	３８.０
知らなかった。	１５	１０.６
その他	１７	１２.０

N＝１４２

⑵ 現在安全対策で最も不安に感じている点について

一番不安が高いのは災害対策であり，これについては約半分の４５.６％が何らかの不安を感じている。つぎは障害対策２６.３％，機密保護２１.９％となっている。（図表２－８）

図表２－８　現在安全対策上で最も不安に感じている点

項　目	件　数	割合％
災害対策	７３	４５.６
障害対策	４２	２６.３
機密保護	３５	２１.９
防犯対策	１６	１０.０
その他	２６	１６.３

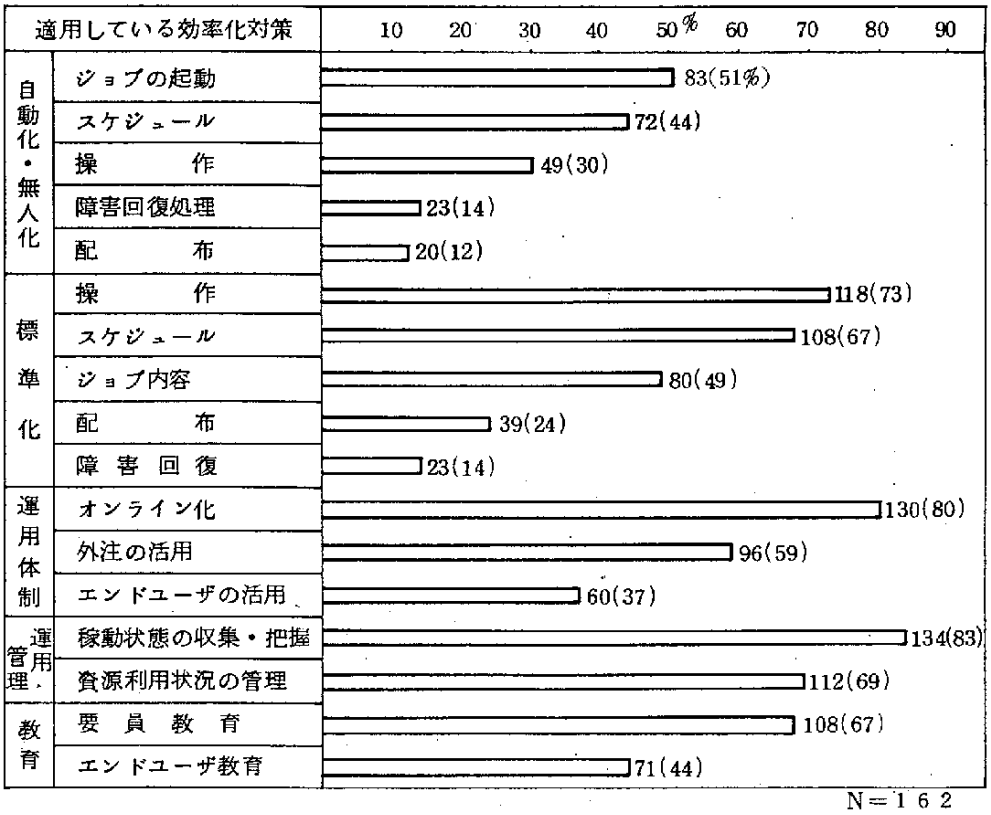
N＝１６０

2.1.5　運用の効率化について

⑴ 適用している効率化対策

運用管理の面では比較的効率化対策が進んでおり稼動状態の収集・把握は１３４社（８３％）と一番多く実施しており，また資源利用状況の管理も１１２社（６９％）となっている。つぎに運用体制の面ではオンライン化および外注の活用がすすんでいる。標準化の面については操作，スケジューリング，ジョブ内容の標準化が比較的進んでいる。教育の面では要員教育について１０８社（６７％）が実施しており，またエンドユーザ教育について７１社（４４％）が実施している。自動化・無人化については比較的少なく，ジョブの起動，スケジュール以外はまだ対策が進んでいない。（図表２－９）

図表２－９　運用の効率化対策



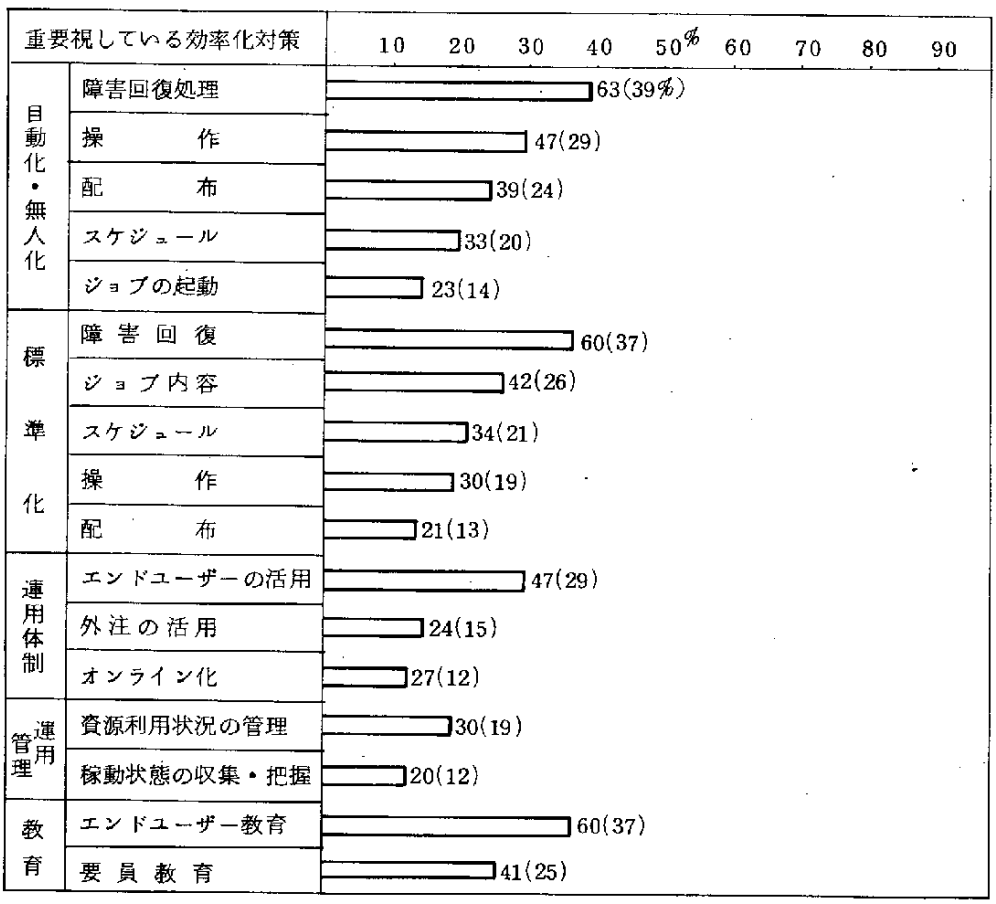
その他運用を効率化するためにとられている方策として，若干ながらつぎのようなものがみられた。

- 利用部門でシステム詳細設計およびメンテナンス作業を行えるような体制
- 作業指示書の標準化の徹底
- 本番ファイルと開発ファイルの分割
- 開発・保守・運用３機能の組織，役割の明確な分離

(2) 重要視している効率化対策

重要視する面からみると自動化・無人化では障害回復処理，操作などが多く，標準化では障害回復処理，教育ではエンドユーザ教育などが重要視されている。運用体制，運用管理などは現在実施済が多いため，ほとんど重要視されていない。（図表２－１０）

図表２－１０　重要視している効率化対策

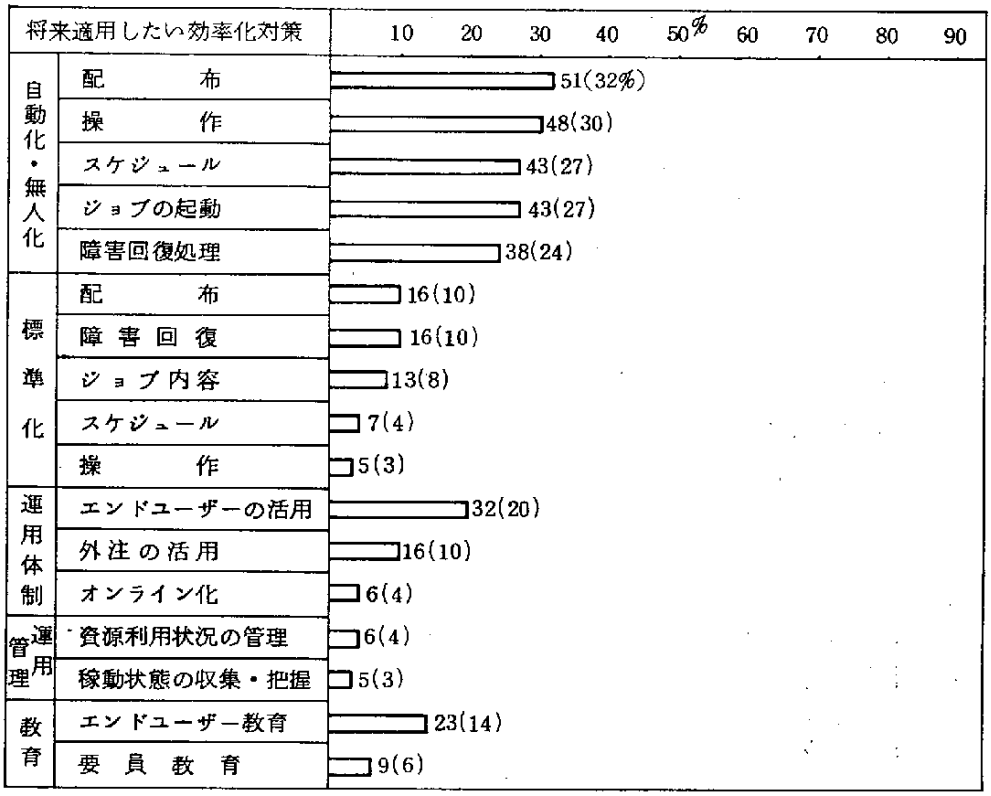


N＝１６２

(3) 将来適用したい効率化対策

将来適用したい対策としては自動化・無人化が多い。また運用体制としてはエンドユーザの活用，さらに，教育についてもエンドユーザ教育が将来適用したいと希望している。（図表２－１１）

図表２－１１　将来適用したい効率化対策



N＝１６２

(4)　現在の運用状況について

コンピュータ稼動時間からみると，１時間～１５時間（１日当り）が一番多く３１.９％となっている。１～２０時間が２５.６％，１～２４時間も２５.６％で，比較的長時間使われている。（図表２－１２）

図表２－１２　コンピュータ稼動時間

コンピュータ稼動時間／日	件数	割合%
１　～　５　時間	３	1.9
～１０　〃	２４	１５.０
～１５　〃	５１	３１.９
～２０　〃	４１	２５.６
～２４　〃	４１	２５.６

N＝１６０

平均稼動時間　１３.５時間

オペレーションの体制からみると二直体制，一直体制，三直体制の３つがほぼ同じ比率となっている。またオペレーションは２人で行うのが多い。（図表２－１３）

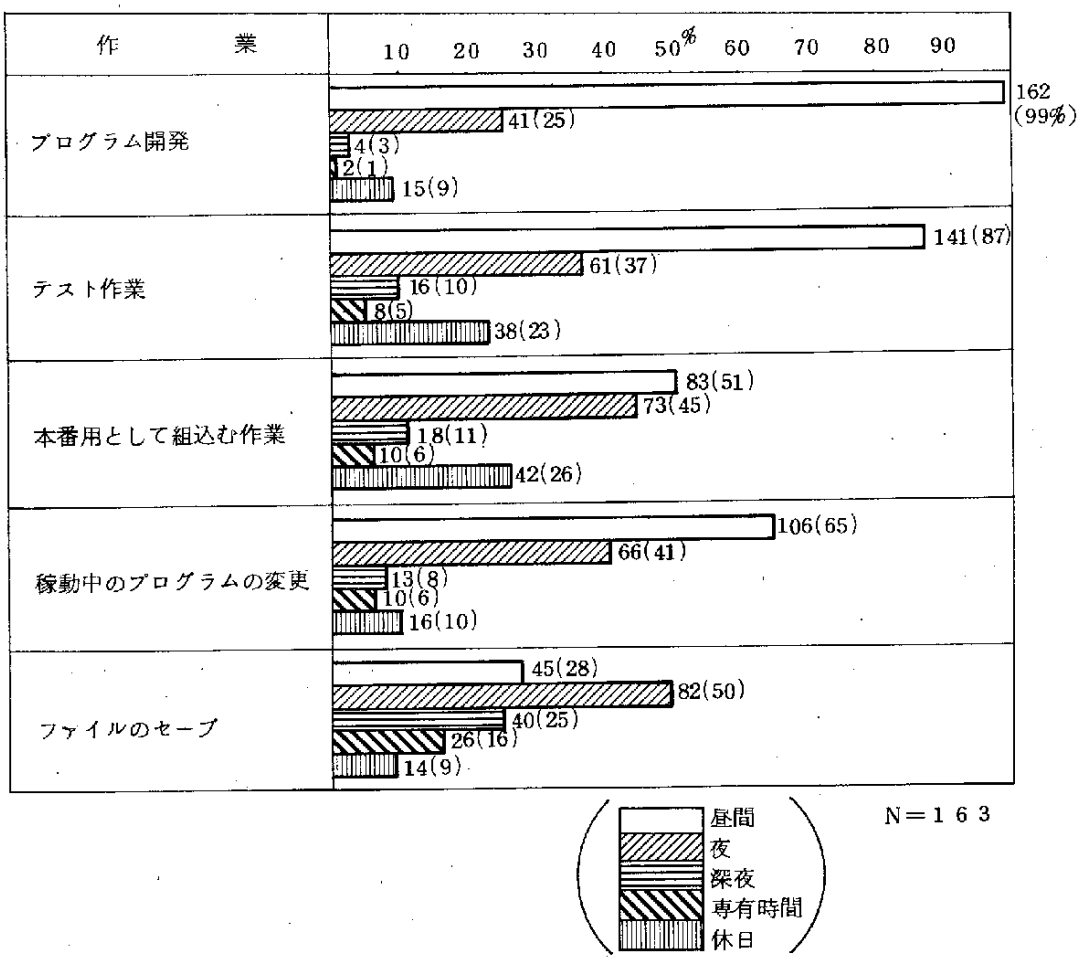
図表２－１３　オペレーション体制

オペレーション体制	件数	割合%
一直体制	４９	３０.４
二直体制	５３	３２.９
三直体制	４４	２７.３
オペレーションは１人で行う。	２４	１４.９
オペレーションは２人で行う。	５９	３６.６
オペレーションは〔３～５〕人で行う。	２５	１５.５
〃　〔6~10〕〃	１０	6.2
その他	１０	6.2

N＝１６１

コンピュータの利用時間帯でみるとプログラム開発は殆んどが昼間に実施されている。テスト作業もほぼ同様に昼間が多い。一方ファイルのセーブなどは夜間に実施される比率が高い。（図表２－１４）

図表２－１４　コンピュータ利用の時間帯



(5)　運用全般についての問題点，意見

今まで述べてきた質問項目以外に運用全般についての意見を求めた結果つぎの図表２－１５のような意見があった。

図表２－１５　運用全般についての問題点，意見

運用全般についての問題点，意見
オペレーションを１００％外注したが，障害時の対応や社員の能力低下が心配。
運用に費用がかかり過ぎている。
オンライン業務の拡大にともない稼動時間帯の拡大が望まれている。早朝化休日稼動をやがて実施する。省力化，勤務条件の改善を目的として自動起動自動終了をはじめとしてM／Tハンドリング，用紙ハンドリングの省人化を行いたい。
２４^{H}×３６５^{日}稼動のため，トラブル時の対処が大変である。 大規模ネットワークのため，障害問題判別に時間を要する。

2.2　ソフトウェアの保守について

2.2.1　保守作業の割合および内訳

(1)　職種別割合

職種別の総工数の中で保守作業が占める割合をみると，ＳＥおよびプログラマについては保守の比率が高く，平均５３.８％にもなっている。特に全体の３８％のユーザは総工数の７０％以上を保守作業に向けており，このことがバックログを累増させる原因の一つであることを示している。一方管理者およびオペレータについては保守の比率が低くなっているが，管理者でさえも４１％の人が保守作業に１０％占有されている。（図表２－１６）

図表２－１６　職種別の総工数の中で保守作業の占める割合

割合(件数) / 職種	10%	20	30	40	50	60	70	80	90	100	平均
ＳＥ及びプログラマ	11	9	21	15	15	23	26	10	15	6	53.8
オペレータ	51	25	13	2	1	3	1	2	1	1	13.1
管理者	62	27	10	6	2	2	4	1	0	0	13.8

Ｎ＝１５１

(2)　保守の種類別内訳

保守の種類からみると完全化保守の割合が高くなっている。この値は平均５３.３％である。修理保守および適応保守は比較的少ない。（図表２－17）

図表２－１７　保守の種類別内訳の割合

保守の種類＼割合(件数)	10%	20	30	40	50	60	70	80	90	100	平均
修理保守	71	32	19	5	7	6	2	3	1	1	21.4
適応保守	71	24	10	7	7	5	3	4	1	0	19.0
完全化保守	14	13	13	11	18	15	24	17	18	4	53.3
その他	27	3	3	2	1	0	1	0	1	1	4.4

N＝１４７

(3)　工程別の作業工程の割合

保守作業の工程別割合からみると，プログラムの修正，プログラムの追跡，プログラムのテストなどが保守の比率が高くなっている。逆にドキュメントの作成，ドキュメントの調査では比較的作業工程が少なくなっている。（図表２－１８）

図表２－１８　工程別の作業工程の割合

作業工程＼割合(件数)	10%	20	30	40	50	60	70	80	90	100	平均
ドキュメントを調べる	77	31	21	10	3	2	0	0	0	0	16.9
ドキュメントを作成する	86	40	13	2	0	0	0	0	0	0	12.7
プログラムを追跡する	40	45	37	14	6	2	5	0	0	0	23.9
プログラムを修正する	36	53	39	11	6	5	0	2	0	0	22.3
プログラムのテストを行う	43	52	42	9	5	1	0	0	0	0	21.4

N＝１５２

2.2.2　保守体制の現状について

(1)　プログラム保守の実施者

プログラム保守を誰がやるかについては「プログラムを開発した社員が保守を行う」が一番多く５８.９％であった。また「その時点で余力のある社員がドキュメントを見て行う」が４０.４％あった。複数回答が可能なのでいくつかの方法を併用しているユーザも多い。保守専門要員（社員）が全ての保守を行っている場合も２５.２％あった。メーカーと共同開発のプ

ログラムについては何等かの形でメーカーが保守に参画しているのが２９.４％あった。（図表２－１９）

図表２－１９　プログラム保守の実施者について

項　　目	件　数	割合%
保守専門要員（社員）が全ての保守を行っている。	４１	２５.２
そのプログラムの開発を手がけた社員が保守も行っている。	９６	５８.９
その時点で余力のある社員がドキュメント等を見ながら保守を行っている。	６６	４０.４
そのプログラム等を作成したメーカ等に全て任せている。	１１	６.７
そのプログラム等を作成したメーカ等と共同で行っているが，作業は主に社員が行っている。	２０	１２.３
そのプログラム等を作成したメーカ等と共同で行っているが，作業は主にメーカ側が行っている。	１７	１０.４
その他	１８	１１.０

N＝１６３

⑵　保守の方針の現状

部門にシステム保守に関する方針が５０％以上はあると答えている。ないと答えたのは３４％である。方針があると答えた場合，責任と義務が明示されている場合は３９％で，明示されていない場合は２０％となっている。またプログラムを修正するか，再度新しく書き直すかの判定のガイドラインについては約８０％がないと答えている。また応用ソフトの変更のために関係部門に費用を請求する手順についても，確立されていると答えた割合は３０％に満たない。ただ管理者が方針の実施にあたり口頭と行動の両方で努力しているかどうかという点では５０％以上が「努力している」，と答えている点，また「部内の予算の一部を保守用ツールの開発や購入のために充当している」，が約５０％を占めるなど，保守に関する認識は高まりつつある。（図表２－２０）

図表２－２０　保守の方針の現状

項　　目	10　20　30　40　50%　60　70　80　90
部門にシステム保守に関する方針がある。	85(55%) 53(34) 17(11)
方針がある場合，それにはそれぞれの責任と義務が明示されている。	60(39) 31(20) 64(41)
上位の管理者は方針の施行にあたり，口頭と行動の両方で努力している。	86(56) 19(12) 50(32)
プログラムを修正するか，再度新しく書き直すかの判定のガイドラインはある。	22(14) 119(77) 14(9)
保守の予算は〝ゼロベース〟になっており毎年の保守を正当化できる。	45(29) 60(39) 50(32)
保守の予算化は保守の型（例　保守，改造）ごとに行われている。	31(20) 80(52) 44(28)
応用ソフトの変更の際，関係部門に費用を請求する手順が確立されている。	36(23) 95(61) 24(16)
部門の予算の一部は，保守用ツールの開発や購入のためにあてられている。	72(46) 46(30) 37(24)

（――ある　＝＝ない　－－－－どちらでもない　以下図表２－２４まで同様）　N＝１５５

(3)　保守体制の現状

「システム保守を管理するための組織を持っている」と答えているのが４０%強，「作られていない」が５０%強となっており，保守の組織づくりは進行途上にあるといえる。つぎに部内にエラー分析者または保守分析者は２０%強しか設置していない。また保守管理者はプログラムのすべてのアクティビティについて５０%以上が「知っていない」と答えている。このようなことから，保守についてはいまだ体制づくりが進んでいないといえよう。（図表２－２１）

図表２－２１　保守体制の現状

項　　目	10　20　30　40　50%　60　70
システム保守を管理するための組織がつくられている。	64(42%) 78(51) 10(7)
部門内にエラー分析者または保守分析者が設けられている。	33(22) 96(63) 23(15)
上位の管理者はシステム保守の管理がうまくいっているか否かを定期的にチェックしている。	74(49) 59(39) 19(12)
保守管理者はプログラムのすべてのアクティビティについて知っている。	30(20) 86(57) 36(23)

N＝１５２

(4)　保守のライフサイクルの現状

保守のライフサイクルという概念はあまり一般ユーザーに浸透しておらず，７１％がこの概念を使っていない。また「管理上のチェックポイントが保守のライフサイクルの中に含まれている」，が１０％以下，同時に「ライフサイクルの中にチェックポイントは実際上使われている」なども１０％以下となっている。（図表２－２２）

図表２－２２　保守のライフサイクルの現状

項　　目	10　20　30　40　50%　60　70
保守のライフサイクルという概念は使われていない。	102 (71%) 36(25) 6(4)
保守のライフサイクルには十分な管理上のチェックポイントが含まれている。	12(8) 84(58) 48(34)
ライフサイクル中のチェックポイントは実際に使われている。	12(8) 80(56) 52(36)
応急の処理がなされた時，速やかにライフサイクル上の手続きに戻るようになっている。	18(12) 73(51) 53(37)

N＝１４４

(5) 保守の手順・手続きの現状

保守の手順・手続きの方法が確立しているかどうかについてアンケート結果を分析してみる。「ソフトウェアの変更要求の提出と承認の正式な手順が設定されている」が約８０％，「口頭で変更要求が示された場合文書を改めて書くようになっている」が８０％である。「プログラム／データが変更された時，必ず文書も更新するような手順が確立されている」も６０％以上となっている。一方実施状況が比較的少ないのは「必要なデータを収集し分析するような手順が確立されている」，また「保守作業の際，どのコマンドを使って行うかの指示を示した作業手順がある」，および「変更の際，分析者／プログラマが他に与える影響を調べることを強制する手順がある」などであるが，全般的に手続き・手順の確立は不充分であるといえる。（図表２－２３）

図表２－２３　保守の手続き・手順の現状

項　　目	10　20　30　40　50%　60　70　80
不必要ならプログラムをソースコードのライブラリから消す手順が確立されている。	103(65%) 42(26) 14(9)
プログラマが名前付の規則に従っているか否かを確める手順が確立されている。	82(52) 54(34) 23(14)
エラーが発見された時，他のシステムにも共通な影響を与えるか否かを調べる手順がある。	58(36) 76(48) 25(16)
保守作業と対象ソフトウェアから必要なデータを収集し，分析するような手順が確立されている。	32(20) 92(58) 35(22)
ソフトウェア変更要求の提出と承認の正式な手順が設定されている。	124(78) 24(15) 11(7)
オーソライズされたロードモジュールのみが実行可能になるような手順が確立されている。	84(53) 50(31) 25(16)
口頭で変更要求が示された場合，文書を改めて書くようになっている。	127(80) 23(14) 9(6)
保守作業の際，どのコマンドを使って行うかの指示を示した作業手順がある。	25(16) 99(62) 35(22)
システム分析者／プログラムからの変更要求はユーザのそれと同じ変更管理手順を踏むようになっている。	80(50) 54(34) 25(16)
プログラム／データが変更された時，必ず文書も更新するような手順が確立されている。	107(67) 45(28) 7(5)
変更に際し，それによって影響を受ける人やグループに対し通告をする手順がある。	97(61) 40(25) 22(14)
保守作業の間バックアップのソースコードを保管しておくことを強いる手順がある。	82(52) 56(35) 21(13)
不必要になったプログラムをその時点でコードモジュールのライブラリから消去する手順が確立されている。	79(50) 57(36) 23(14)
変更の際，分析者／プログラマが他に与える影響を調べることを強制する手順がある。	27(17) 101(63) 31(20)

N＝１５９

(6) その他

変更の際，「ロードモジュール・レベルでの変更が不可能」が約６０％，「保守活動に対し統計が採られていない」，「保守は型ごとに分類されて行われていない」などが５０％以上を越えている。一方「プログラムは必要に応じて保護されている」については８０％以上が保護されていると回答しており，ソフトウェアの保護に関心が高いことを示している。

（図表２－２４）

図表２－２４　その他の実施事項

項　　目	10 20 30 40 50% 60 70 80 90
プログラムは必要に応じて保護されている。	127(83%) 24(16) 2(1)
保守は型ごとに分類されて行われている。	45(29) 79(51) 29(19)
保守活動に対し統計が採られている（例・頻度，型，プログラム工数，テストケース数等）	55(36) 79(52) 19(12)
変更の際，ロードモジュール・レベルでの変更が可能である。	42(27) 90(59) 21(14)
複数バージョンのプログラムを識別することができるようになっている。	57(37) 71(47) 25(16)

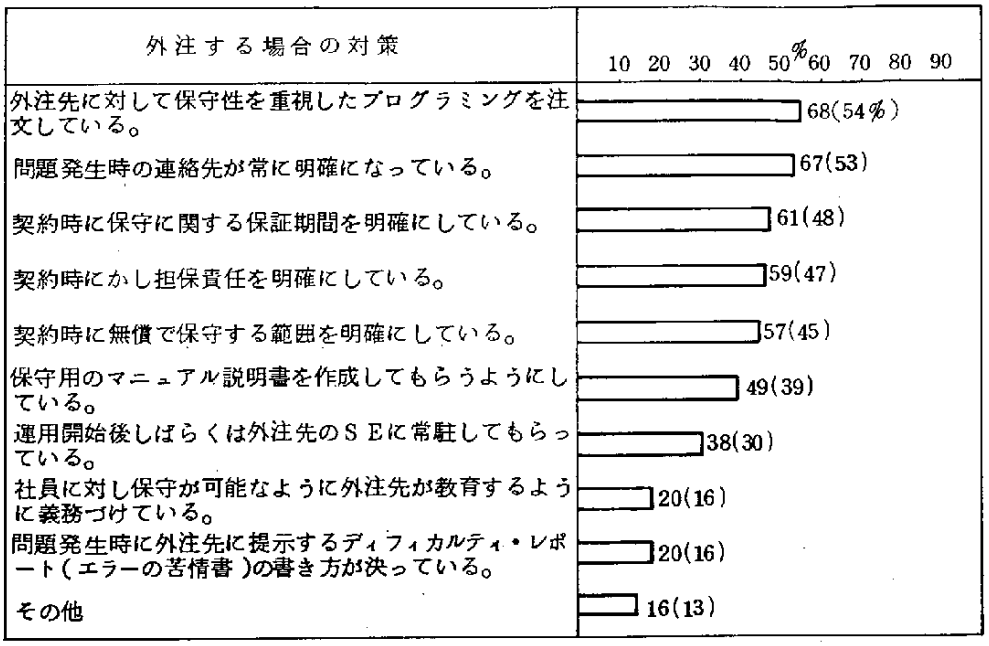
N＝１５３

2.2.3　ソフトウェア開発の外注および将来の保守の対策

ソフトウェア開発を外注する場合の対策としては，⑴外注先に対して保守性を重視したプログラミングを注文する，⑵問題発生時の連絡先が常に明確になっている，⑶契約時における明確化の項目として①かし担保責任②保守に対する保証期間③無償保守の範囲などを約５０％が実施している。　一方⑴問題発生時に外注先に提示するディフィカルティ・レポート（エラーの苦情書）の書き方が決っている。⑵保守が可能なように外注先が社員に教育することを義務づけている。⑶運用後しばらくは外注先のＳＥに常駐してもらっているなどは，実施されている比率が小さいものとなっている。

（図表２－２５）

図表２－２５　ソフトウェアを外注する場合の対策



Ｎ＝１２６

2.2.4　ソフトウェア保守に係る現状の問題点

(1)　要員の問題

ソフトウェア保守に関する要員の問題は保守要員の交替である。きわめて深刻と答えたものが１４.３％，かなり深刻が３０.４％，やゝ深刻が２４.８％で合計で６９.５％が深刻な問題としてとらえられている。また保守要員の数についても「きわめて深刻」は8.1％ではあるが「やゝ深刻」と答えたものは３６.６％と多い。保守要員の技術レベルおよび保守要員の動機づけについては深刻な問題ではないとする傾向が強い。（図表２－２６）

図表２－２６　要員の問題

問題点	きわめて深刻		かなり深刻		やや深刻		問題ではあるが深刻ではない		全然問題ない	
		%		%		%		%		%
保守要員の交替	23	14.3	49	30.4	40	24.8	35	21.7	10	6.2
保守要員の技術レベル	4	2.5	32	19.9	48	29.8	55	34.2	21	13.0
保守要員の数	13	8.1	40	24.8	59	36.6	51	31.7	6	3.7
保守要員への過度の負荷	5	3.1	32	19.9	53	32.9	55	34.2	11	6.8
保守要員の動機づけ	2	1.2	20	12.4	44	27.3	63	39.1	19	11.8

N＝１６１

(2)　エンドユーザ関連の問題題

エンドユーザに関しては約半数が問題としてとらえている。特に大きいのはエンドユーザ組織の交替であり，これは「極めて深刻」とするのが5.1％，「かなり深刻」が１２.７％，「やや深刻」が２３.６％となっている。また「エンドユーザが応用システムを理解していない」という点も問題としてあげられている。（図表２－２７）

図表２－２７　エンドユーザ関連の問題

問題点	きわめて深刻		かなり深刻		やや深刻		問題ではあるが深刻ではない		全然問題ない	
		%		%		%		%		%
エンドユーザが応用システムに興味をもっていない	4	2.5	17	10.8	35	22.3	69	43.9	29	18.5
エンドユーザが応用システムを理解していない	6	3.8	27	17.2	47	29.9	57	36.3	17	10.8
エンドユーザの非現実的期待	3	1.9	24	15.3	41	26.1	64	40.8	20	12.7
エンドユーザのトレーニング不足	3	1.9	24	15.3	58	36.9	57	36.3	10	6.4
エンドユーザ組織の交替	8	5.1	20	12.7	37	23.6	66	42.0	18	11.5

N＝１５７

(3)　応用システム関連の問題

応用システムの問題としてはまずシステム設計文書の品質の問題がとり上げられている。「極めて深刻」が5.1％，「かなり深刻」が１５.８％，

「やゝ深刻」が３４.２％と，深刻な問題としてとらえられる比率が５０％を越えている。この他の問題は比較的深刻でないという位置づけとなっている。（図表２－２８）

図表２－２８　応用システム関連の問題

問題点	きわめて深刻		かなり深刻		やや深刻		問題ではあるが深刻ではない		全然問題ない	
		%		%		%		%		%
システム設計文書の品質	8	5.1	25	15.8	54	34.2	61	38.6	9	5.7
システム開発時の品質管理	2	1.3	30	19.0	44	27.8	68	43.0	13	8.2
システムの動作エラー	2	1.3	14	8.9	38	24.1	76	48.1	21	13.3
システム・プログラムのストレージ上の制約	1	0.6	11	7.0	27	17.1	64	40.5	48	30.4
システム・プログラムの処理時間上の制約	3	1.9	14	8.9	37	23.4	66	41.8	32	20.3

N＝１５８

(4)　その他の問題点

保守に関する問題点としてはまず第１に保守作業の効率が悪い点であり，「かなり深刻」が２３.３％，「やゝ深刻」が３０.８％である。「きわめて深刻」の4.4％を加えると，深刻な問題としてとらえるのが５８.５％となっている。また保守工数の見積りの問題も深刻であり，システム拡張へのユーザからの要求も問題としてとらえられている。比較的問題点として少ないのはハードウェア／ＯＳの信頼性やその変更といった面であり，この面では少しずつ問題は排除されつつあることを示している。（図表2－２９）

図表２－２９　ソフトウェア保守に係るその他の問題点

問題点	きわめて深刻		かなり深刻		やや深刻		問題ではあるが深刻ではない		全然問題ない	
		%		%		%		%		%
ハードウェア／ＯＳ等の変更	7	4.4	16	10.1	38	23.9	68	42.8	26	16.4
システム拡張へのユーザからの要求	2	1.3	33	20.8	53	33.3	52	32.7	17	10.7
保守工数の見積り	5	3.1	23	14.5	50	31.4	61	38.4	10	6.3
保守作業の効率	7	4.4	37	23.3	49	30.8	57	35.8	4	2.5
予算上の制約	7	4.4	17	10.7	43	27.0	58	36.5	24	15.1
ハードウェア／ＯＳの信頼性	2	1.3	1	0.6	19	11.9	75	47.2	54	34.0

Ｎ＝１５９

2.3　ソフトウェアの評価について

2.3.1　開発されたソフトウェアの評価

開発されたソフトウェアの評価を行っているかどうかについてみると，「評価を行っていない」が多く４４.４％，「評価を行っている」のが１１.９％となっている。（図表２－３０）

つぎに評価を行っていない理由をみると，一番多い理由が「評価の方法および項目の設定が困難であるから」で６６.７％，つぎに「評価は困難」と「評価部門がない」がそれぞれ２９.２％となっている。（図表２－３１）

図表２－３０　ソフトウェアの評価の実施状況

評価を行っている	１９件	１１.７％
部分的に評価を行っている	71	43.8
評価を行っていない	72	44.4

Ｎ＝１６２

図表２－３１　評価を行っていない理由

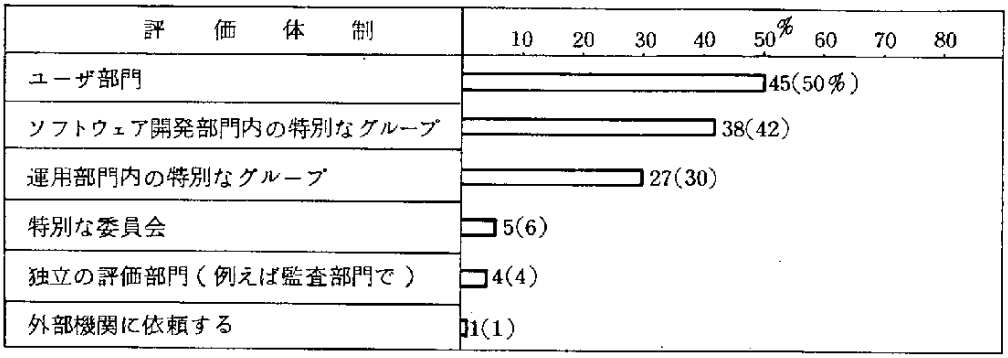
評価の方法および項目の設定が困難であるから	４８件	６６.７％
ソフトウェアの評価は非常に困難であるから	21	29.2
評価部門がないから	21	29.2
評価しても意味がないから	8	11.1
その他	2	2.8

Ｎ＝７２

2.3.2 評価の体制

評価の体制として「ユーザ部門が評価する」が一番多く５０％となっている。つぎに多いのが「ソフトウェア開発部門内の特別なグループ」で４２％，「運用部門内の特別なグループ」が３０％となっている。（図表２－３２）

図表２－３２　評価の体制

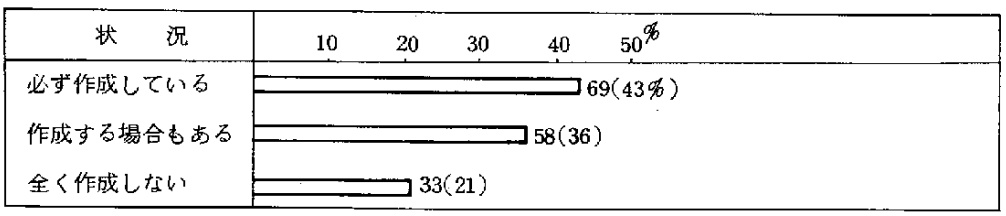


N＝９０

2.3.3 開発完了時点での報告書の作成

報告書の作成については「必ず作成している」が４３％，「作成する場合もある」が３６％となっているが，「全く作成しない」が２１％あり，今後の課題となっている。（図表2－３３）

図表２－３３　開発完了時点での報告書作成状況

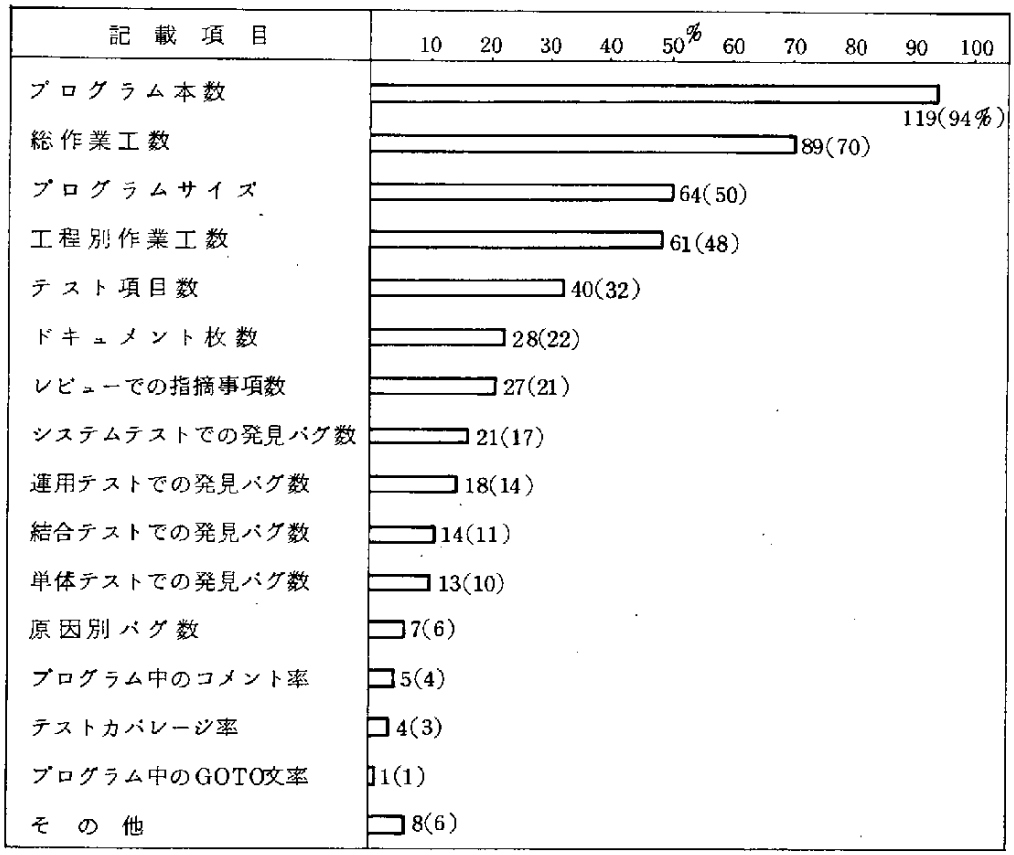


N＝１６０

つぎに報告書の記載項目をみると，「プログラム本数」が一番多く９４％以上が記載している。つぎに「総作業工数」「プログラムのサイズ」「工程別作業工数」などを半数以上が記載している。一方プログラム中の「GOTO文の比率」を記載しているのは１件のみ事例があり，「テストカバレージ率」「プログラム中のコメント率」「原因別バグ数」などが極く少ない。「テス

ト項目数」「ドキュメント枚数」「レビューでの指摘事項数」「システムテストでの発見バグ数」などは比較的記載される回数が多い。（図表２－３４）

図表２－３４　報告書の記載項目

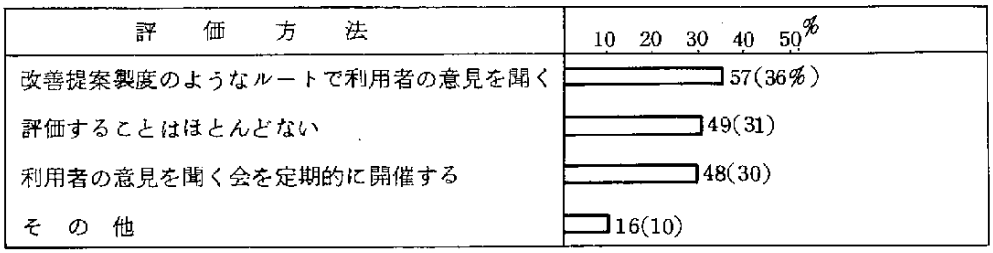


N＝１２７

2.3.4　運用開始後のソフトウェア評価法

運用開始後におけるソフトウェアの評価方法については，「評価することはほとんどない」という回答が３１％で，「何らかの形で意見を聞く」のが６６％となっている。この内訳として「利用者の意見を聞く会を定期的に開催する」のが３０％，残り３６％が「改善提案制度のようなルートで利用者の意見を聞く」という回答であった。（図表２－３５）

図表２－３５　運用開始後のソフトウェア評価方法



N＝１６０

2.3.5　生産性についての評価状況

生産性についての評価状況をみると「ほとんど評価していない」が３０.４%で，「常に評価している」が１２.４%となっており，「評価する場合もある」が５７.１%で，今後は評価を行うユーザがふえていくものと思われる。（図表２－３６）

図表２－３６　生産性についての評価状況

常に評価している	２０件	１２.４%
評価する場合もある	９２	５７.１
ほとんど評価しない	４９	３０.４

N＝１６１

つぎに生産性の基準値についてみると９０.３%が「基準値がない」と答えており，「基準値をもっている」がわずか9.７%である。（図表２－３７）

図表２－３７　生産性の基準値について

あ　る	１５件	9.７%
な　い	１３９	９０.３

N＝１５４

またプログラム改造の場合の生産性の基準値についてみると，「改造の場合の基準値を持っている」ユーザは少なく，５.１%に過ぎない。（図表２－３８）

図表２－３８　プログラムの改造の場合の基準値について

ある	8件	5.1％
ない	１４９	９４.９

N＝１５７

基準値があると答えた場合の基準値の内容をつぎの図表２－３９で示す。

図表２－３９　プログラムの改造の場合の生産性の基準値

プログラムの改造の場合の生産性の基準値
プログラムの組み方（標準か非標準），I／Oの増減数，修正内容により標準化してあるステップ数を計算し，１ステップ当りの標準時間をかけて求める。
時間当り変更ステップ数
保有S／W　１KL当りの保守工数
人数算定の場合，10万step／人月　を基準にしている

さらにプログラム再利用の場合の基準値については「持っている」ユーザが3.2％とほとんどが持っていないことを示している。（図表２－４０）

図表２－４０　プログラム再利用の場合の基準値について

ある	5件	3.2％
ない	１５０	９６.８

N＝１５５

再利用の場合の基準値があると答えた中でその基準値の内容を示したものをつぎの図表２－４１にあげる。

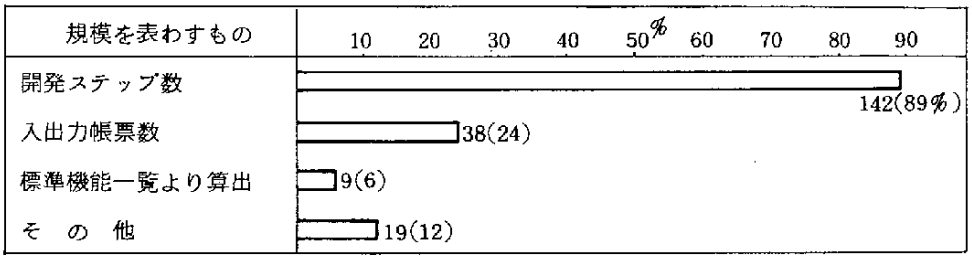
図表２－４１　プログラム再利用の場合の基準値の内容

プログラムの再利用の場合の基準値
機能が等しい場合
流用化率…… $\frac{\text{システム総ステップ数} - \text{正味製造ステップ数}}{\text{システム総ステップ数}}$
ＳＴＥＰ／時間

2.3.6　プログラム開発の規模の表示

プログラム開発の規模を示すものとしてどのようなものをベースにしているかをみると，圧倒的に多いのは「開発ステップ数」であり，８９％がこれを用いている。「入出力帳票数を使っている」は２４％，「標準機能一覧表より算出している」はごく少なく６％，以上３つに入らない「他の方法を用いる」が１２％である。（図表２－４２）

図表２－４２　プログラム開発の規模を表わすものについて

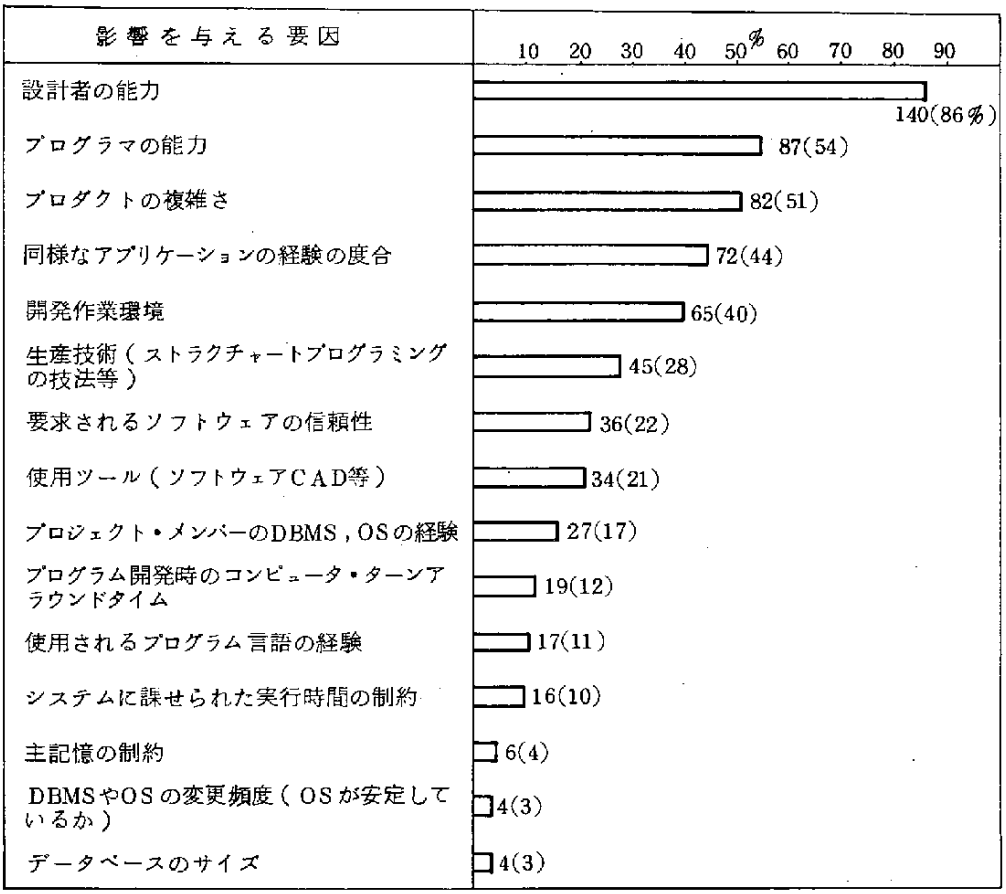


N＝１６０

2.3.7　生産性に影響を与える要因について

生産性に影響を与える要因についてみると一番大きな影響を持つと思われるものが「設計者の能力」で８６％がこれをあげている。つぎに「プログラマの能力」「プロダクトの複雑さ」がそれぞれ５０％程度となっており，つづいて「同様なアプリケーションの経験の度合」，「開発作業環境」などである。少ないものとしては「データベースのサイズ」「主記憶の制約」「ＯＳの安定度」などである。（図表２－４３）

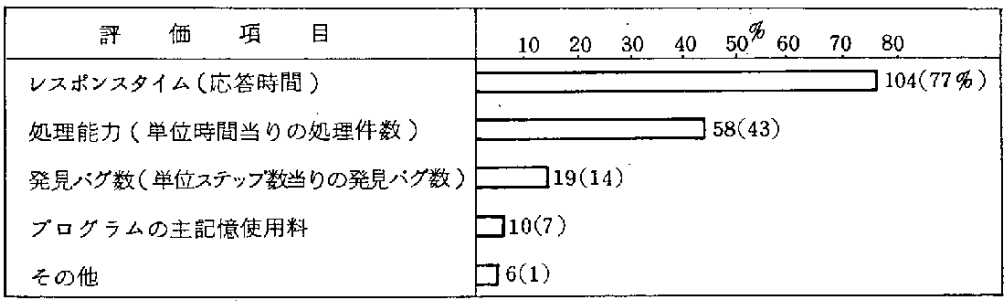
図表２－４３　生産性に影響を与える要因について



N＝１６２

また生産性以外の評価法についてみると「レスポンスタイム」（応答時間）が一番多く７７%，次に「処理能力」（単位時間当りの処理件数）が４３％となっており，その他の項目については比率は非常に少ない。（図表2－44）

図表２－４４　生産性以外の評価方法について

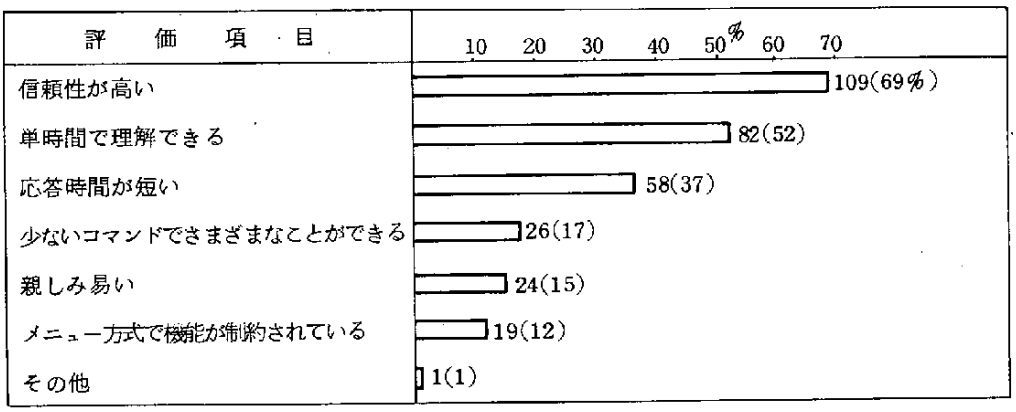


N＝１３５

2.3.8 アプリケーションプログラムの使い易さについての評価

アプリケーションプログラムの使い易さについての評価をみると，まず，「信頼性が高い」ことを６９％があげている。つぎに「短時間で理解できる」ことを５２％が，「応答時間が短かい」ことを３７％があげている。また「少ないコマンドで多くの機能が果せる」が１７％，「親しみ易い」が１５％などとなっている。（図表２－４５）

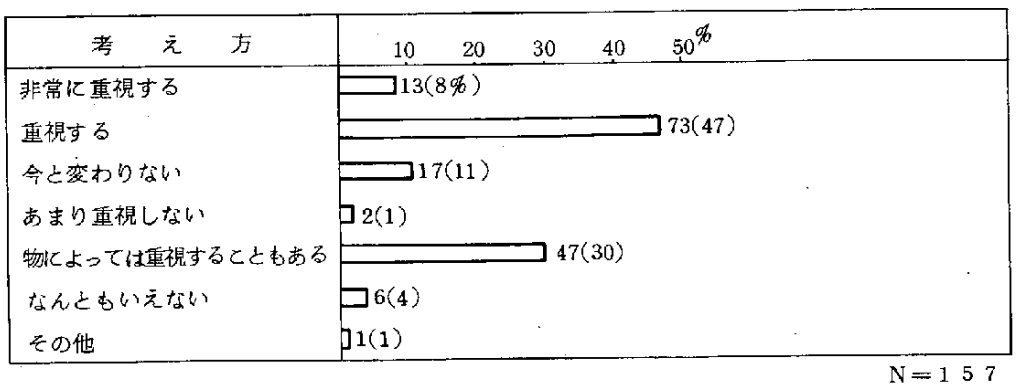
図表２－４５　アプリケーションプログラムの使い易さについての評価



N＝１５７

最後に評価についてどう考えているかについては「重視する」が４７％，「物によっては重視することもある」が３０％となっており，「なんともいえない」「あまり重視しない」は両方で５％に過ぎず，評価が重要事項であることは認識されている。（図表２－４６）

図表２－４６　評価についてどう考えているか



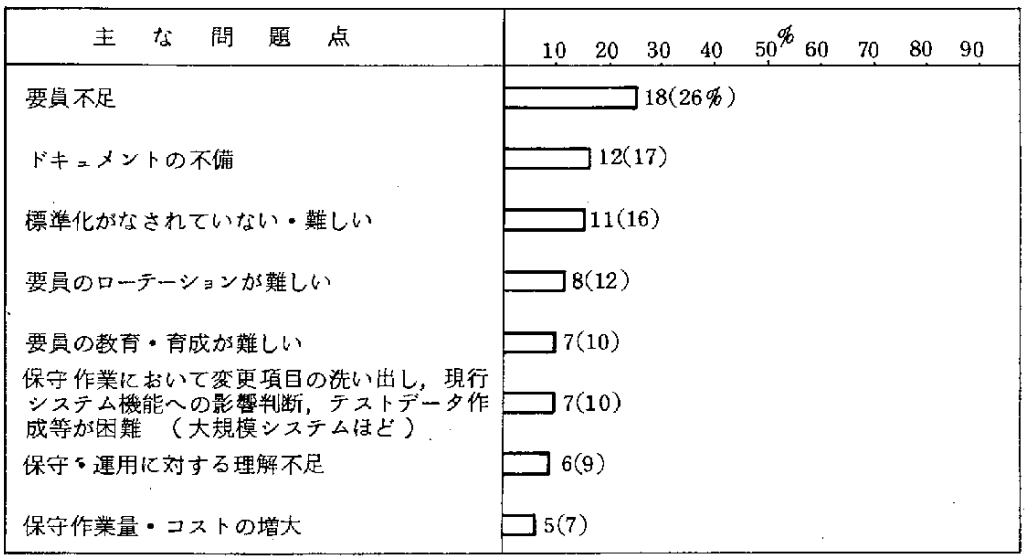
N＝１５７

2.4　ソフトウェア運用・保守の問題点と今後の課題について

2.4.1　ソフトウェア運用・保守に関する問題点

ソフトウェア運用・保守に関する問題点についてアンケートした結果をつぎの図表２－４７に示す。

図表２－４７　ソフトウェア運用・保守に関する問題点



N＝６９

上図に示した以外の問題点についてはつぎに示す諸点があげられている。

- 全てが今後の課題となっている
- ノウハウが個人に帰属し，システム部門全体の財産になっていない
- 開発者と運用者との引き継ぎが悪い
- 開発担当者が保守作業に携わっているため，新規システム開発が計画どおり進まない
- システム監査，安全対策
- 障害対策
- 意図する tool がまだ実現していない
- 現在使用していない言語の保守が困難
- テスト環境
- 新技術の調査導入

2.4.2 ソフトウェア運用・保守に関し期待する新技術

ソフトウェア運用・保守に関し期待する新技術についてのアンケートの回答をつぎの図表２－４８に示す。

図表２－４８　ソフトウェア運用・保守に関し期待する新技術

ソフトウェア運用・保守に関し期待する新技術	件数
プログラミングの自動化技術　（保守も含めた）	5
ドキュメンテーション作成の自動化技術	5
ソフトウェアＣＡＤ　（ＡＩによる）	4
データディクショナリソフトウェアの開発	3
ＲＤＢ	2
非手続言語　（日本語での表現方法）	2
他言語への自動変換プログラム	1
生産性向上ツール　（ex NECのＳＥＡ／Ｉ・etc）	1
自動運行システム	1
プログラム解析・修正支援プログラム	1
テストシミュレーションツール	1
プログラマーズワークベンチツール	1

上図表以外にはつぎのような意見があった。

- 種々のツールが出されているが，いづれも高価であり，電算資源を使いすぎる，または，サービスレベルに影響する程負荷が高い。現在開発されているものをもっと安価に簡便な操作で使用できる様改良されていくべきである
- 期待する新技術は特に見当らない。ソフト開発時に仕かけ（開発ツール）を用意し，徹底して標準化する以外にないと考える

3. ソフトウェア運用

3. ソフトウェア運用

システムの大規模化，複雑化に伴い，ソフトウェア運用の高度化もまた重要な課題となっている。

ソフトウェア運用の高度化には，効率性，信頼性，安全性が充分考慮された運用の仕組の確立が望まれる。

この章では，ソフトウェア開発後の運用の基本となる事項すなわち運用組織体制，運用基準の設定，システム監査の実行，安全対策についての概略を紹介し最後の効率化対策について述べることとする。

3.1 運用要員と組織

コンピュータ部門での組織体制は，開発部門，運用部門，そして保守部門が大きな作業単位であるが，実際の組織は共通して存在したり，さらに別な観点（利用部門毎，システム別，職務別など）から分かれている。

そして，それぞれが抱えている問題は，昔も今も，部門を共通して，

① 要員不足（特に保守要員に顕著）
② スキル不足
③ ノウハウの蓄積と継承
④ ローテーション
⑤ 運用技術の特殊性
⑥ 主に忙しさからくる問題

などがみられる。

3.1.1 利用形態と運用体制

システムの利用／運用形態が，オンライン化されたこと，大規模に集中化されたり，逆に分散化の方向へ進むケースもある。システムの利用者低辺は増々拡大しており，端末機，パソコン，ワークステーションなどの密着した機器が広く設置されている。さらに，これだけ，多様化，複雑化してきているシステムに一担災害がふりかかったり，障害が発生した場合，その影響も重大なことになってきている。

このような状況に合った運用形態や，サービスを考えていかなければならない。

詳しくは，3.2運用基準で述べている。

3.1.2 要員管理

ここでは，コンピュータやソフトウェアの運用要員に対する管理について考えてみる。

運用要員の管理では，特に目標管理と一体化された内容で行われることが望ましい。

(1) 労働環境

コンピュータ自体の適用範囲が拡大してきており，それに伴ない，運用時間数も同じように増加している。このような状況から，オペレータなどの運用担当者にとっても，長時間残業や，交代勤務体制が必要になる。

要員の対応は，簡単かつ早急には出来ないため，自動化／無人化運用への要求も強まるであろう。

(2) 健康の保持

夜間作業や長時間作業が続いた場合の健康管理には注意が払われていなければならない。

また，ＶＤＴ（ディスプレィ画面）を長時間，見つめながら作業する場合には，適当な休憩と時間数の上限を設けるべきであり，反射防止対策なども積極的に取り入れる。

ＶＤＴ作業の労働安全対策の事例を招介する。（別に作業環境，定期検診についても定めている。）

① 連続作業１時間につき，１０－１５分間程度の休止時間（労働省）

② １日最長４時間以内。５０分毎に１０分以上の休憩（総評），１時間毎に１５分の休止（同盟）

③ 連続２時間作業後１５分休憩（アメリカ）

(3) 目標管理

運用部門では，新しい業務の導入や現在稼動しているシステムの運用・改善作業が絶えず発生してくるので，担当者でも，新しい技術や知識の収得には機会を与えること，自己啓発への手助けをすることが必要である。

(4) 外部要員の活用

要員不足や労働環境の悪化，経費の節約，外部ローテーションの問題などの理由から外部要員の導入を真剣に考え，また導入している企業が多い。

これは，最近のアンケートや，各種の調査で共通していえることである。

3.1.3 ジョブ・ローテーション

担当する職務を変えることにより，組織や個人の活性化はかなり図れる。

コンピュータ部門では，その知識や技術が特殊であるとの解釈から，ジョブ・ローテーションについては，部門内にとどまっていたり，行われても少数であったりしている。

しかし，最近では，業務知識に詳しいことがシステムの開発や運用担当者に求められる能力や知識であると考えられてきている。つまり，コンピュータ専門の知識よりも業務知識のほうが，開発／運用担当者にとって大切であるとの認識になってきている。

このことは，より効率良く，効果的なシステムを開発するためにも，コンピュータ関連部門と，他の実務やスタッフ部門との人的交流が定期的に積極的に行われる必要がある。

コンピュータ部門内での，より上級の職務へのローテーションはもとより大切であるが，外部とのローテーション，それも出来る限り入社後早い時期に定期的，かつ意識的に慣習化されるようにすべきである。

では，ソフトウェアの運用から見たジョブ・ローテーションについて考えてみる。

ソフトウェアの運用・保守要員にとって，既に出来上がっているプログラムを運用し，保守していくことは，新規開発に比べると，比較的単調かつ目標が見つけにくい作業である。教育面での対処は別途必要として，やはり，同じ作業を長期間担当させることなく，内部／外部とのローテーションを積極的に推し進めなければならない。

その場合，問題となる主な点は，

① 運用や障害発生時の対応力が弱体化すると考えられるため引継ぎが難しい
② 専門の知識が必要であり，資料として残されていないため，引継ぎが難かしい
③ 他部門とは独立した人事体系になっている
④ 上司が移ることに抵抗している

などであるが，いずれの項目・内容とも，ジョブ・ローテーションの重要

性からみて，不可能な理由ではない。

3.1.4 要員教育

コンピュータ・システム，ソフトウェアを効率よく，確実に運用していくには，運用を担当する要員，保守を担当する要員にも，必要な知識が要求され，それらは経験だけでは身につかないことも多く，また満足のいくスキルを持つまでには長い期間や個人差が生じてしまう。

開発部門やエンドユーザも含めて，コンピュータ・システムに関連している人々に対する，効果的，計画的な教育体制やカリキュラムは是非とも検討され，用意されなければならない。

社内で独自のコースを設けることが，実務とも結びついたり，実機で訓練できるため，大変効果的であるが，社外やメーカ・ディーラ主催の講習会を有効に活用することも必要になる。

(1) 教育項目

各職能（管理者，システムエンジニア，プログラマ，オペレータなどにも分けれる。）毎に教育項目や習得目標の設定を行う。また，利用者や開発者，運用／操作者など，立場によっても相応に必要な教育項目があげられる。特に，前述したジョブ・ローテーションの問題もからみ，向上心や好奇心をわきたたせる魅力ある教育コース，厳しくても実績と評価の高い教育コースなど，特徴をもった内容も盛り込まれていることが必要である。

(2) 教育手段

職場教育，企業・組織内教育，外部教育などに分けられるが，さらに付け加えて，自主的な研修，グループ研究会や勉強会，自己学習，他システムの視察，発表会・講演会への参加，国家試験・資格試験への受験なども重要な教育・育成手段である。

さらに検討してほしいことは，教える側になる機会を是非作ってほしい点である。

3.2 運用基準

ソフトウェアの運用に際して，まず必要なことは，運用組織，運用基準をきめ，しっかりした運用体制を整えることが大切である。

本項では，ソフトウェア開発後の運用に係わる作業規定項目，管理規定項目および運用で使われるドキュメントの紹介を行う。

3.2.1 運用基準とは

システム開発およびシステムの運用を円滑に，ミスなく実施するために取り決める基準のことであり，運用の責任範囲およびシステムの効率的な運用を行うための管理項目と管理方法を明確にするものである。

運用基準の中味は運用体制，作業基準，管理基準および，運用のためのドキュメントより構成される。

運用基準の決定は，運用時の操作性，処理効率，システム全体のスループット効率に大きな影響を与える。運用基準設定にあたっては以下のことを留意する必要がある。

① 運用のし易さ
② 利用部門へのサービス向上への寄与
③ ミス防止への役立ち
④ 無人化・自動化の方向への整合性

3.2.2 運用基準の種類

ソフトウェア開発後の運用業務の流れは図表3－1に示すとおりであり，これらの運用業務に合わせて，運用基準が設定される。

(1) 運用全般

コンピュータ資源の有効活用と利用部門のサービスの向上にむけて，コンピュータ・システム全体の運用方法を明確にすることを目的としている。上記に盛り込む事項は以下のとおりである。

① 運用形態

オンライン，TSS，バッチなどのタスクの割当て方法を明確化する。

② 稼動時間

オンライン，ＴＳＳ，バッチごとにサービス稼動時間を明確化する。休日における稼動時間も明確化する。

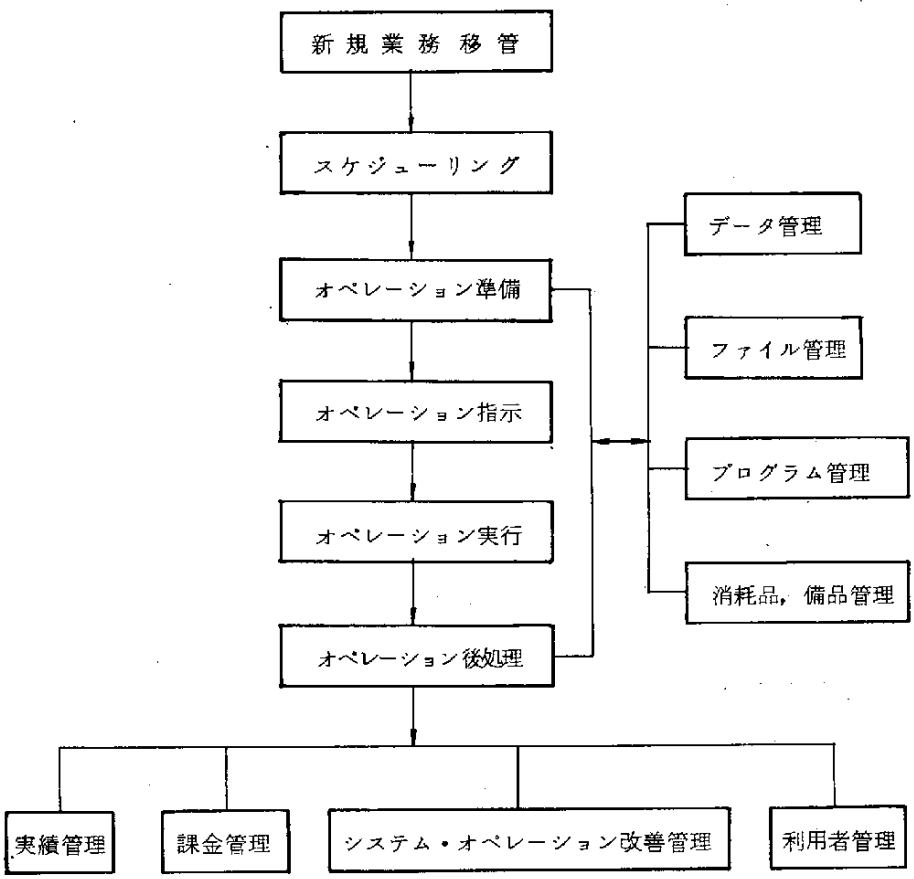
③ 資源配分

コンピュータ資源のショブへの割当て方法を明確化する。

④ システム全体の運用体制

システム全体運用のための体制を明確化する。

図表３－１　運用工程



㋑　現在稼動中の業務システムを円滑に運用するための利用部門，運用部門に対する調整窓口の明確化

㋺　ソフトウェア開発後の本番移行のために窓口の明確化

㋩　スケジューラーの明確化

㋥　ハードなど資源管理者の明確化

㋭　運用支援ツールの開発など，オペレーション効率化のための運用サポート部隊の明確化

㋬　一直，二直などのオペレーション体制の明確化

(2)　新規業務移管

本番移管後のシステム運用が正しく行われるように，本番実施に先がけて

行われる本番移行審査の手順，審査項目，審査ドキュメントなどを明確にすることを目的としている。上記に盛込む事項は以下のとおりである。

① 審査項目

本番移行に際し，具体的に何を審査したら良いかを明確にする。審査のポイントとなる事項は以下のとおりである。

㋑ 審査に必要なドキュメントはそろっているか

㋺ スケジュール，資源の利用方法，トラブルの対応方法など，コンピュータ運用の基準に合致しているか

㋩ 運用の自動化，無人化の方向に逆行していないか

② 審査ドキュメント

審査に伴って，システム担当者から提出されるドキュメントは以下のとおりである。

㋑ 新規業務移管審査依頼書

業務名，本番開始時期，スケジュール，トラブル対応などが記入されている。

㋺ 開発ドキュメントの運用部分

㋩ 運用に必要なドキュメント

ＪＣＬリスト，配布ガイド，新規作成ファイル情報記入シート，リラン対応手順書，テスト結果報告書など。

③ 審査方法，時期，体制など

上記の各項目について，いつ，誰が，どのような手続きで管理していくかを明確にする。ここでは最低限，手続書および，本番化の規定書を作成する必要がある。

(3) スケジューリング

スケジュールもれ，データ未着による混乱，スケジュール遅れなどを防止し，スケジュールの平滑化，効率化をはかれるように，スケジュールの立案方法，変更方法，作成すべきドキュメントを明確にすることを目的としている。上記に盛込む事項は以下のとおりである。

① スケジュール立案要件

スケジューリングに必要な要因を明確にする。

㋑ スケジューリング期間

スケジューリング対象とする期間を明確にする。

㈡ 使用予定業務の明確化

本番稼動の業務およびテストなど本番業務以外でのコンピュータ使用予定を明確にする。この明確化の中には以下の項目を含む。

- 入力データ作成時期
- 出力の納期
- 処理時間
- 処理順序

㈢ コンピュータ稼動日の明確化

休日，祭日，P.M などでコンピュータを使用できない時間，日を明確にする。

② ドキュメント

スケジューリングに必要な参照ドキュメントと作成すべきドキュメントを明確にする。

㈠ 参照ドキュメント

業務ごとの標準処理日程表，インプットデータ一覧表，アウトプットデータ一覧表など。

㈡ 作成すべきドキキュメント

週単位または月単位の稼動計画表を作成する。

③ スケジュール立案方法

上記の立案要件，参照ドキュメントに基づき，稼動計画表を作成する手順を明確にする。

ここで留意すべき事項は業務の優先度，多重度，スケジュールの平準化などについて充分考慮しておくことである。

(4) オペレーション

オペレーションが正確に，かつ迅速に行えるように，オペレーションの準備作業，オペレーション作業，オペレーションの事後作業について，作業手続き，方法，ドキュメントを明確にすることを目的としている。

① オペレーション準備作業

当日のオペレーションを中心とした作業予定の立案方法，媒体，データの取り扱い方法，オペレーション実行時にトラブルが発生した場合の対処方法，システム・端末・周辺装置の立ち上げ方法を明確化する。

② オペレーション作業

オペレーション時の順守事項，注意事項を明確化する。

㈠ コンソールなどコンピュータ・システムの稼動状況の監視方法

㈡ コンソール，ハードウェアの操作方法

㈢ 異常時の対応方法（リカバリー方法，緊急時の連絡方法など）

㈣ ファイル，媒体の取り扱い方法

③ オペレーション事後作業

オペレーションの事後処理としては，ジョブ実行状況の確認，出力帳票・データの送付，媒体保管，システム・端末・周辺装置の停止業務があり，これらについての確認方法，手続きを明確化する。オペレーション終了後の確認事項は以下のとおりである。

㈠ 予定ジョブの終了確認

㈡ 出力帳票・出力ファイルの確認

㈢ 出力データの送付状況の確認

㈣ 媒体の安全性の確認

㈤ システムの停止状況の確認

④ ドキュメント

これらオペレーション業務の基本となるオペレーションマニュアルの構成要素について列挙すると，

㈠ 各種機器の操作法

㈡ 電源のＯＮ／ＯＦＦ

㈢ 空調管理

㈣ 障害，災害発生時の対応

㈤ コンソールシートの取り扱い

㈥ オペレータコマンドの取り扱い

㈦ 入出力データの取り扱い

㈧ システムの立ち上げ，終了方法

㈨ オペレーションの準備作業

㈩ 媒体の取り扱い

などである。

(5) コンピュータ使用

本番稼動以外の業務でコンピュータを使用する場合の基準を明確にすることを目的としており，オペレータへのテスト依頼運用基準，ＴＳＳ運用

基準，緊急時のマシン使用運用基準，利用部門使用運用基準などがある。

① テスト依頼運用基準

オペレータに依頼して，テストを実行してもらう場合の基準で，ジョブ実行クラス，ジョブ名のふり方，実行時間ファイルなど資源の使用方法，ライブラリーの使用方法，実行結果の配布方法を明確化したものである。

② ＴＳＳ運用基準

ＴＳＳ端末機より，プログラマーがコンピュータを使用する場合の基準で，ＴＳＳのサポート時間，機密保護方法，ジョブの実行方法などについて明確化したものである。

③ マシン使用運用基準

完全なマシン室のクローズ制をしいていないコンピュータ部門で，プログラマーが，緊急時にコンピュータを使用する場合の基準で，マシン室の入室手順，サポート時間，ファイル，媒体などの利用方法，ジョブの実行クラスなど，ジョブの実行方法について明確化したものである。

④ 利用部門使用運用基準

利用部門の人がコンピュータを使用する場合の，テストオーダ，ＴＳＳ，マシン使用および利用手続きについて利用者の制限，ファイルの提供などの運用方法を明確化したものである。

⑤ ドキュメント

①から③までのコンピュータの使用形態にあわせて，各々コンピュータ使用申し込み書が必要となるが，ここではテスト依頼とＴＳＳ利用の場合のドキュメントについて紹介する。

㋑ テスト依頼書

依頼者名，業務名，処理予想時間，必要資源，他業務との関連などを記入したものである。

㋺ マシン使用申請書

ＴＳＳでマシンを使用する場合にオペレータにあらかじめ必要な資源を準備してもらうための申請書で，依頼者名，業務名，マシン使用日時，必要資源（使用ソフト，データセット，媒体など）を記入したものである。

(6) ファイル管理

ファイルのスペースの有効活用，ファイルの障害，破壊防止，Ｉ／Ｏ環境の維持・向上をはかれるようにファイルの管理方法を明確化することを目的としている。

① ファイル作成基準

新規業務などでファイルが新たに作成される場合のファイル新設の手続き方法と作成基準を明確化する。

㋑ 手続き

この手続きは申請手順と申請内容の審査方法を明確にするもので，審査時のポイントとしては，媒体の確保に問題がないか，他のシステムへの影響はないのか，ファイル作成基準に合致しているのかなどがあげられる。

㋺ 作成基準

ファイルの命令方法，ボリウムの割当て方法，スペースパラメータの指定方法，カタログ方法，用途別の保存期限などについての基準を明確化する。

② ファイルの保管規定

ファイルの利用目的，保管目的，ファイルのボリウム，重要性を考慮して，保管媒体，保管場所などを明確化する。また世代ファイルについて何世代まで保管するかの基準も明確化する。

③ ファイルのスペース管理規定

このスペース管理規定はスペースの有効活用とファイルのオーバーフロー防止のために，ファイルの増加状況，各ファイルごとのスペース状況などのファイルの現状把握とその対策方法，保存期限を過ぎたり，必要以上にスペースをとったり，その他データ作成基準からはずれた基準外ファイルの削除方法，ファイルのコンプレス方法を明確化する。

④ ファイルの機密保護規定

ファイルのアクセスに対し，ファイル更新，ファイル参照を可能とする範囲を規定すると共に，機密保護の方法を明確化する。

⑤ ファイルの災害，障害対策規定

災害，ハードウェア障害，オペレーションミスなどによりファイルが破壊されないように，また破壊されても対応ができるようなバックアップ体制を明確化する。破壊対策としては二重保管，ディスクボリウムセ

ーブ，磁気ファイルのプロテクトリングの取りはずしなどが考えられる。

ファイルの重要性，リカバリーに許容される期間，オンライン，または，バッチファイルなどファイルの種類ごとに障害対策規定を設定する必要がある。

⑥ ボリウムの割り当て基準

ファイルの使用目的，I／O環境，スペースの空き状況により，ボリウムの割り当て方法を明確化する。

⑦ ファイルの利用規定

複数アプリケーションから，同一ファイルを共用する場合，コンテンションや，間違った更新が行なわれないようにする使用制限，オンラインファイルに対するオンライン稼動中のオンライン以外でのファイル使用制限，本番ファイルのテストでの使用制限などを明確化する。

(7) プログラム管理

開発されたプログラムの紛失，破壊を防止し，必要な時に必要なプログラムが正しく使えるように，ライブラリの利用方法，プログラムの登録，メンテナンスの方法，プログラムのバックアップ方法を明確にすることを目的としている。

① ライブラリ規定

ライブラリの種類，利用方法，格納媒体について明確化する。

② メンテナンス規定

プログラムの名前のつけ方，プログラムの登録，変更，削除を行う場合の手続き方法，登録すべきライブラリ，変更時の世代管理，実施時期などを明確化する。

③ バックアップ規定

誰が，いつ，どのプログラム・ライブラリーをどういう方法で，どの媒体にバックアップをとるかを明確化する。

(8) 実績管理

処理時間の短縮，作業時間の平準化，リスポンスタイムの向上，リラン時間の減少などコンピュータの安定稼動がはかれるように，センターおよび端末の運用状況，オペレーション状況を的確に把握し，運用上の問題点の解決をはかることを目的として，実績データの管理項目，収集方法，評価方法を規定したものである。

① 実績データの管理項目

実績データの管理項目としては以下の項目があげられる。

㋑ ジョブごとの稼動状況（ＣＰＵ使用時間，エラップスタイムなど）

㋺ ジョブごとの資源の使用状況（ＣＰＵ，メモリ，Ｉ／Ｏなど）

㋩ ジョブごとの多重度状況，待ち状況

㋥ ジョブごとのＩ／Ｏデータ量

㋭ 回線の負荷状況

㋬ 端末機の稼動状況

㋣ トラブル状況（端末機，センター，ソフトウェア）

② 実績データの評価方法

各実績データの収集より，各資源の使用率の高いところ，リスポンスタイムの悪いところ，資源利用に片寄りのあるところ，ジョブ待ち状況の多いところ，Ｉ／Ｏ回数の多い周辺装置などを把握し，その原因を見つけ出し，資源の増設，ジョブの組合せの変更，ファイルボリウムの位置替え，ソフトウェアの修正による改善などを行う。

実績管理の運用に際してはデータ収集，分析のためのツールの導入がポイントとなる。

(9) 利用者管理

コンピュータ業務の拡大，オンラインシステムの普及，コンピュータ利用の大衆化が進むにつれ，ファイルの破壊，データの粉失・改ざん・漏えい，コンピュータ資源の過大使用などはコンピュータ・システムの運営に大きな混乱を招くと共に，コンピュータ費用の増大をもたらす。以下の事項を明確化し，上記事項が発生するのを防止することを目的としている。

① プログラム開発，ジョブの実行などのコンピュータ利用規定

② コンピュータ室の入退室規定

③ ファイルの提供規定

④ システムの利用者登録規定（センタ利用者コード，端末機利用者コード）

⑤ 入出力データの管理規定

⑥ 利用部門への課金規定

(10) 課金管理

利用者へのシステム運用経費の公平な課金を目的として，課金情報の収集方法と費用の配賦方法を明確にするもので，課金の基礎情報としては，以下のものがあげられる。

① ジョブ別のエラップスタイム
② ジョブごとのＣＰＵ時間
③ ジョブごとのインプットデータ件数
④ 出力ページ数
⑤ ジョブごとの媒体利用スペース
⑥ 利用者ごとの端末機利用台数

しかしながら，利用者別ジョブ別に正確な情報を収集するには多大な工数がかかり，仮に細かく集めたところで，労多くして，それ程の効果は期待出来ない。ある程度の公平感が確保出来れば，データ収集が容易で，配賦計どおりに復旧させなければならない。

3.3 安全対策

コンピュータ・システムが，社会や，企業，組織にとって，その中枢部分にあり，重要な役割りをになっているケースが多くなっている。個人にとっても，その恩恵を受身で得るのではなく，生活に密着した形でかかわり合っている最近では，コンピュータ・システムが災害や事故によって機能を止めたり，異常な動作をすることは許されない状況にあり，例えそのような事態におちいった場合は，すみやかにかつ完全に元どおりに復旧させなければならない。

個々のシステムによって，要求される度合いはまちまちであり，追って，そのことについて述べていくが，以前は専門家だけが叫けんでいたシステムの安全対策の重要性が，広く利用者までも含めて声高く求められている。

3.3.1 システムへの脅威

コンピュータ・システムの運用における災害を考えると，それらは以下の4つに分類することができる。

① 自然による災害（火災，地震，雷，風水害など）
② 環境異常や設備・機器の不良による災害（温度，湿度，塵，電源・空調の異常，漏水，ノイズ，通信回線異常，鼠害など）
③ 人為的災害（窃盗，詐欺，破壊活動など）

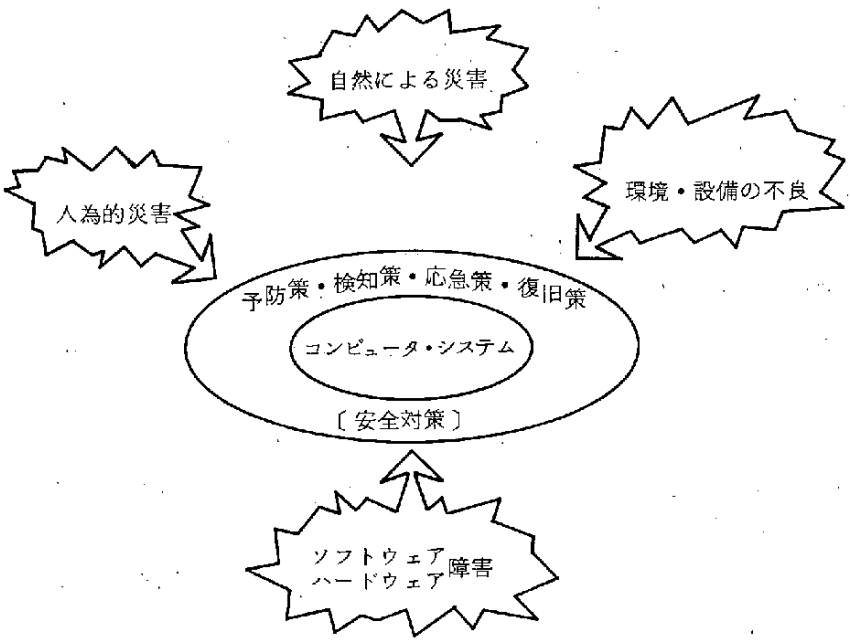
④　コンピュータ自体の問題からくる障害（ハードウェア機器の障害，ソフトウェアのミス，操作ミス）

また，コンピュータシステムの運用にとって脅威となっているこれらの災害が，具体的に影響をおよぼしている対象が存在しており，それらを守る対策が講じられなければならない。

①　運用要員
②　データ（重要性，機密性，再生不能など）
③　記録媒体
④　ハードウェア
⑤　ソフトウェア
⑥　通信系統
⑦　付帯設備
⑧　システムの利用者

安全対策を考えるためには，これらのシステムを正常に運用することを脅かす原因を十分に洗い出して，その発生頻度や可能性，発生した場合の影響度を十分に分析して把握しておく必要がある。これによって，効果的，効率的な安全対策の検討が可能となる。

図表３－２　コンピュータ・システムへの脅威



3.3.2 システムの安全対策

危険の洗い出しと定量的な分析・把握が出来た後は，安全度についての分析も行い，具体的な安全対策の選択を行う必要がある。また，その段階では効果の度合いとともに，効率面やコスト面への影響も同時に検討されなければならない。

以下に，システム運用設計時点で考慮すべき安全対策について，運用中に考えなければならない安全対策面の各種項目について述べる。

(1) 安全度の分析

システムに対する脅威について，具体的に出来る限り定量的に分析したことに対応して，システムの安全性についても具体的かつ定量的な分析を行うようにしたい。

定量的な分析・調査を行うには，チェックリストを詳かく作ることである。

チェックリストは，新らたに作っても良いし，行政面でのガイドラインとして発表されているもの（下記参照）を利用したり，コンピュータ・メーカや施設関係の企業，関連する提供マニュアルなどを参考にすることも可能である。

① 通商産業省 ・電子計算機システム安全対策基準（3.3.3にて簡単に紹介）
・システム監査技術者試験

② 郵政省 ・データ通信ネットワーク安全・信頼性基準

③ 大蔵省 ・金融機械システムの安全対策

④ 警察庁 ・コンピュータ・システム安全対策研究会（情報システム安全対策指針）

(2) 安全対策の選択

対策の実施によって，どこまでの安全を達成すればよいのかを明らかにする必要がある。

この安全目標の設定については，個々の脅威に対して，許容できない危険と許容できる危険，また，対策の有効性について検討し，具体的対策を選択する。

① 災害対策（地震対策，火災対策，災害発生時の対策，発生後の対策，復旧対策）

② 環境・設備対策（温度，湿度，空調設備，塵，電源，静電気，ノイズ，塩害，鼠などへの対策，発生時対策，復旧対策）

③ 人為的災害対策（防犯対策，機密保護，不正アクセス防止対策，破壊活動への対策，発生時対策，発生後対策）

④ ハードウェア障害対策（ハードウェアの保全，二重化対策，オンライン対策，発生時対策，復旧対策）

⑤ ソフトウェア障害対策（障害原因究明支援機能，障害の検出／通知と対応策，復旧対策）

⑥ 媒体障害対策（予備系保有，二重化，内容の保全，復旧対策）

⑦ 操作ミス対策（スキルアップ，モラル向上，記録，健康管理）

(3) 安全対策導入と効果

選択された安全対策と見送られた安全対策も含めて，実施の優先度，コストなども加味した実施範囲を決める必要がある。

対策の中には，同種同一の機能によっても同等の効果が得られるもの，別々の対策を二重，三重に講じることによって，より万全な対策とする場合も検討する。

特に安全対策は，その機能が役立つことがないことが望まれ，かつ珍らしい事であるため，一般に無駄な対策であり，経費であると思われ易いため，日常的な啓蒙活動や認識を高めること（教育，訓練，改善案検討など）を忘れてはならない。

(4) 安全対策の維持改善

新技術の開発や導入，確信的犯罪からの防護，定常的な監査と評価，そして，これらのことから要求としてあげられてくる改善策への検討と実行を絶えず行っている体制が敷かれなければならない。

(5) 安全対策と人間関係

安全対策を高度化，高コスト化している原因は，人間対策の部分が大きい，自然災害や環境・設備災害，障害対策などは，未だ高い費用を要するケースも有るが，確実な解決方法があったり，解決方向に向かうことができる。しかし，人間の問題（人為的災害）については，これらとは違った性格をもった災害であり，また安全対策も難かしいといえる。

人間の悪意や不注意を防ぐための対策を考えた場合の問題点は，安全対策の究極的な問題点でもあるといっても過言ではない。そのためにも，人

間関係や要員管理を円滑にしていることは，それだけでも安全対策のウェイトを大きく軽減することになる場合も多い。

3.3.3 安全対策基準

コンピュータ・システムの安全対策基準として，通商産業省が策定し公表している内容を紹介する。

基準は，対策内容ごとに，最高度の安全性，信頼性が必要とされるシステムにおいて実施することが望ましい対策としての強化基準（A基準），一般にシステムを運用する上での最低限の安全性，信頼性を確認するために不可欠と考えられる対策としての基本基準（C基準），これらの中間的な対策としての標準基準（B基準）の3段階に分類されている。

対策の具体的な内容は，設備面の対策（１５２項目），技術面の対策（13項目），運用管理面の対策（６３項目）に分けられている。

(1) 設備基準（１５２項目）

コンピュータ・システム，コンピュータ室，データ保管室などを火災，地震などの自然災害，不法侵入者による破壊行為などのあらゆる危険から物理的に保護するための対策である。

Ⅰ 建 物

1. 立地および環境
2. 建物の位置，周囲，利用形態
3. 構造
4. 開口部（出入口，窓，非常口など）
5. 内装等

Ⅱ 電子計算機室およびデータ等保管室

1. 位置および配置等
2. 開口部
3. 構造内装等
4. 設備
5. 什器，備品等

Ⅲ 電源室および空気調和機械室

1. 位置および配置
2. 開口部

3. 構造

4. 設備

Ⅳ 電源設備

Ⅴ 空気調和設備

Ⅵ 監視制御

(2) 技術基準（１３項目）

コンピュータ・システムの安全性，信頼性などの向上をシステム自身においてハードウェア的，ソフトウェア的に解決するための対策である。

Ⅰ 信頼性向上機能

（代替機器，縮退・再構成機能，障害検出・切り分け機能，リスタート機能，復旧機能など）

Ⅱ データ保護・不正使用防止機能

（パスワード，識別機能，アクセス制御機能，暗号化，監視・記録機能，不正使用・更新防止機能など）

(3) 運用基準（６３項目）

コンピュータ・システムの運用管理などを充実させることにより，システムの安全性，信頼性などの向上を図るための対策である。

Ⅰ 管理体制

Ⅱ 入退管理

1. 入館・入室資格の付与

2. 入退館管理

3. 入退室管理

Ⅲ コンピュータ・システムの運用管理

1. 標準化

2. 運転および確認

3. 管理

Ⅳ データおよびプログラム（ドキュメントを含む）の保管管理

Ⅴ 電源設備，空気調和設備，防災設備および防犯設備の管理

Ⅵ 監視

Ⅶ 外部委託

Ⅷ 教育訓練等

Ⅸ システム監査

3.4　システム監査

大規模ソフトウェアの開発，通信の高度利用など，コンピュータの利用は高度化，複雑化，かつ広範囲に利用され，コンピュータ利用がもたらす影響が非常に大きくなってきている。これにつれて，システム監査への関心も高まってきている。当委員会が実施した「ソフトウェア開発・運用の高度化・効率化方法に関するアンケート調査」でも，「システム監査を必要と思わない」と答えた企業は１６０社中わずか３件（ 1.9％）であった。

本項ではシステム監査の実行に焦点をあてて，システム監査の必要性，監査内容，実行方法，実行上の留意事項について述べる。

3.4.1　システム監査の定義

システム監査の定義については，我が国では統一見解が定まっていないので，ここでは(財)日本情報処理開発協会が７６年に提唱したものと，通商産業省が８４年に出した産業構造審議会情報産業部会で答申した定義について紹介する。

前者は「システム監査とは，監査対象から独立した客観的な立場で，コンピュータを中心とする情報処理システムを総合的に点検，評価し，関係者に助言・勧告することを言い，その有効利用の促進と弊害の除去とを同時に追求して，システムの健全化をはかるものである」と定義している。

後者は「システム監査は，コンピュータ・システムの効率性，信頼性，安全性を確保するため，監査対象から独立した監査人（システム監査人）が一定の基準（システム監査基準）に基づいて，コンピュータ・システムを総合的に点検・評価し，関係者に助言・勧告するものである」と定義し，より具体的にシステムの効率性，信頼性，安全性の３つの確保の観点から，システム監査を実行するよう促している。

3.4.2　システム監査の実行

(1)　システム監査の対象部門

情報システム部門，データ発生部門，データ入力部門，アウトプットの活用部門すべてを対象とする。オンラインシステム，分散処理システムの普及により，情報システム部門のみのシステム監査では，運用の効率性，信頼性，安全性を確認するのは困難になってきている。

(2) システム監査の対象業務

通商産業省より８５年１月に出されたシステム監査基準によれば，システム監査の対象業務はコンピュータ・システムの企画段階，開発段階，運用段階のすべてを対象としている。

各々の段階におけるシステム監査のポイントと監査対象業務は以下のとおりである。

① システム企画段階は，全社的な立場から，コンピュータ・システムをいかに管理，運営していくかという観点から監査がなされ，以下の各々の業務について，システム監査が実行される。

- 経営方針
- 組織計画
- 要員計画
- 監査計画
- 評価計画

② システムの開発段階は採算性，信頼性，安全性，生産性，運用性の幅広い観点から監査され，運用段階に入ってからの効率性，信頼性，安全性が確保されるようにする。以下の各々の業務について，システム監査が実行される。

- 予備設計
- 基本設計
- 詳細設計
- プログラミング
- システムテスト

③ システムの運用段階は安全性，信頼性，機密性の観点から監査され，以下の各々の業務について，システム監査が実行される。

- 入力プロセス（現場処理，センタ処理）
- オペレーション（マシンオペレーション，ライブラリ）
- 出力プロセス（端末出力，センタ出力）

(3) システム監査実施基準

システム監査の実施基準については，業務全般についての一般基準とシステムの品質に適用する品質基準とがある。

一般基準

- 準拠性
- 採算性
- 適時性
- 生産性

品質基準

- 安全性
- 信頼性
- 機密性

システム監査の実施基準と対象業務の関係を示すと図表３－３のようになる。

図表３－３　システム監査の実施基準と対象業務

システム監査の内容 ＼ 対象業務の範囲			システム監査の実施基準						
			一般基準			品質基準			
			準拠性	採算性	適時性	生産性	安全性	信頼性	機密性
企画レベル	経営方針		○	◎	◎				
	組織計画		◎		○				
	要員計画		◎		○				
	監査計画		○	○	○				
	評価計画		○	◎		◎			
開発レベル	予備設計		◎	◎	○				
	基本設計		○		○			○	
	詳細設計		○		○			◎	◎
	プログラミング				○	◎		◎	◎
	システムテスト				○			◎	
運用レベル	入力プロセン	現場処理	○					◎	
		センタ処理	○	○	○		○	◎	
	オペレーション	マシンオペレーション	○		○	○	◎	○	
		ライブラリ	○				◎	○	○
	出力プロセス	出力管理			○			○	◎

◎　最も重要　　○　重要

（出典：コンピュータ運用管理入門（ＮＥＣ））

具体的にシステム監査を実施する時は(2)の対象業務の項でのべた対象業務の各項目ごとに監査実施内容を示したチェックリストを作成しておくと良い。通商産業省のシステム監査基準における実施基準では企画業務で２２項目，開発業務で３１項目，運用業務で５２項目に分けている。

3.4.3　システム監査実施上の留意点

前節でシステム監査の実施方法についてのべてきたが，本節では効果的なシステム監査を行うための留意点について述べる。

(1)　トップ・マネージメントの理解

日本では，システム監査を実施している会社が少なく，この普及のためにはまずトップ・マネージメントにシステム監査の重要性を十分に認識してもらうことが重要なこととなる。トップ・マネージメントの理解なくしては，効果的なシステム監査はまず実行されない。

(2)　システム監査人の養成

システム監査に取り組もうとする場合，まず最初に壁にぶつかるのは，システム監査人がいないということである。システム監査人は，監査技術の他にデータ処理システムの仕組やシステム開発に関する知識を有していることが最低限必要となる。この監査人の養成には監査技術の習得よりも，コンピュータ関連知識の習得に時間をとられる傾向があるため，コンピュータ専門家に監査教育を実施する。

(3)　システム開発過程の標準化

システム監査を企画業務，開発業務，運用業務すべてに実行する場合には，システム開発過程の標準化がなされていないと，監査ポイント，監査タイミングが不明確になり，効果的な監査の実行が不可能になる場合が多くなる。従って，開発工程，ドキュメントを標準化し，システム監査人が容易に介入できるようにすることが大切である。

(4)　システム監査対象業務の明確化

システム監査を企画業務，開発業務，運用業務をどのような業務に細分化して実施するかはその企業，その企業の環境条件，コンピュータの活用状況，運用形態で変ってくる。そこでどの程度に細分化したら良いかを決定する必要がある。

(5)　システム監査への期待の明確化

効率性，信頼性，安全性のすべての分野に重点をおいて行うのか，特定の分野においてシステム監査を実施するかは，その企業のシステム監査人の育成状況，また重要性の位置づけにおいて決めていくのが良い。始めからすべての分野について行うにはかなりの体制整備が必要となる。それよりもとにかく一歩一歩システム監査の実施に向けて歩きだすことが重要である。

(6) システム監査の実施主体

システム監査の実施主体は，企業内にシステム監査人を養成するのが究極的には好ましいが，過渡期的には外部へ依頼して監査を実施するという方策も考えられる。

(7) システム監査人の所属部門

システム監査人の所属部門はコンピュータ部門から独立していることが望ましいと思われるが，以下のようなパターンが考えられる。

① 監査室などコンピュータ部門から独立

② コンピュータ部門に属する

③ 監査部門からの派遣

3.5 システム運用の効率化

ハードウェアの性能向上，低価格化，システム機能の充実によって，大規模システムはもとより，小規模なシステムにおいても，ぼう大なデータやソフトウェア資産を抱えている。さらに，開発したくても着手できないでいる要求(バックログ)も依然として大量である。

また，多様化された処理形態や大量の業務処理要求にも対応していかなければならないのに，運用要員不足やスキル不足などの原因から思うように解決できない状態にある。

そして，今後これらの状況が好転していく見込みがないばかりか，端末機やパソコンの急激な普及によって，エンドユーザが拡大してきたためさらに悪化していくと予測されている。

このような状況下では，何らかの手を打っていかなければ事態は深刻化していく一方であるが，要員増強といった直接的な手が昨今の企業環境では望めなくなってきており，運用の効率化や省力化によって乗り切らなければならない。

自動運転，さらには運用を無人で行い，夜間も運用出来れば省力化と処理能力の実質的向上が図れるし，運用業務の一部に外部の力を活用することも考え

られる。エンドユーザ機能を充実させることによって，直接システムの運用をまかせられれば運用部門への負荷も軽くなる。運用方式の標準化や基準化は，運用効率化に大きく寄与するし，ミスの根絶にも役立つ。これら運用効率化の具体的方策について以下に説明する。

3.5.1 利用者部門の活用

システムを運用していく作業や，改善・修復，最適化といったシステム保守作業は，本来システムの利用者のための作業であり，その利用者から要求されて作業する一時的な作業も多い。であるならば，より利用者に近いところでシステムの運用作業が可能であれば運用部門にとっては効率化が，利用者にとっても同様に効果的な作業と，要求に対する速い回答が得られるはずである。

つまり，運用部門にまかせっきりのシステム運用方式では，運用関連部門の軽量化が図れず，利用者にとってもシステム改良の可能性に乏しい陳腐化したシステムになってしまう恐れさえある。

(1) エンドユーザ機能の充実

エンドユーザ機能にも種々あるが，どのような機能システムにどう適用するかは，開発工程での問題として，運用面から考えた場合は，積極的に導入されることが望ましい。

最近のエンドユーザ機能は，単に会話的，簡易的であるばかりでなく，手続き型や一括処理機能も有しており，専用のプログラムを開発する場合と変らないまでになってきている。

エンドユーザへの負担増や教育の充実，機密保護などについて考慮しておかなければならない点も多いのでやみくもに導入することには注意が必要であるが，是非検討すべきである。

(2) オンライン化の推進

エンドユーザ機能の実現も，オンライン処理が基本になっているのが普通である。

最近のコンピュータ・システムでは，小型のコンピュータ・システムやオフィスコンピュータのレベルでもオンライン処理機能は当り前になってきており，即時処理要求の有無は別として，一括処理方式であったとしても，何らかの箇所でオンライン処理との連結を考えるべきである。（例えば，

入力データの生成，結果の入手，マスタファイルの保守管理，処理の実行要求など）

それによって，得られるメリットは単純なバッチ処理であったとしても，利用者，運用保守部門双方にとって大きい。

資源管理や障害対策については，バッチ処理方式よりも念入りに行う必要があるので注意しなければならない。

得られるメリットの幾つかについてあげてみるとつぎのとおりである。

① ターンアラウンドタイムの短縮
② 運用要員への負荷の軽減
③ 利用者部門ではシステムが身近かになり，認識が高くなる
④ 入力ミスや処理要求のミスが直ちに現われるため少なくなっていく
⑤ 運用部門との関係が親密になり，理解も高まる

(3) プログラムの汎用性，操作の簡易化，運用部内での作業として，作業量が多く，時間がかかっているのにあまり着目されていない作業項目がある。

それは，利用者から要求される突発的，かつ，一時的な作業である。具体的には以下のような項目をあげることができる。

① 決められていない形式，もしくは分からないデータを運用システムへの入力データにならないか
② 外部から入手したデータを読み，かつ標準的なファイルに変換したい
③ 出力様式を一時的に変えられないか
④ 処理内容の一部を変えたい
⑤ 操作法や結果の内容がわからない　等々

これらの要求がシステムに対する根本的な問題であるなら，保守作業として保守部門や開発部門が取り組まなければならないが，一時的であり，かつ，緊急性を伴う場合が多いため，どうしても運用部門が相談にのり，対処してしまうことになる。

このような要求をよく分析すると，似た内容やパターンが多いことから何らかの対応策（ことわるなら簡易であるが）を考えることによって運用の効率化にもなるであろう。

例えば，データ入力に関する要求では，ファイルの形態や記録形式，フォーマット情報，データのチェック機能，編集機能，加工機能といったことが多く，これらの処理が汎用的，かつ，適用性の高い処理プログラムが

存在すれば，しかもその運用が簡単，かつ，会話的な操作で処理できるようになっていれば，運用部門が頭を悩ますことも少なくなるであろう。

処理方式についても，結果出力についても同様にあらかじめ決めた形式しか対処できない作り付けのプログラムではなく，パラメータによって運用的に柔軟に変更でき，そのパターンも汎用性が考慮された形式であれば，こういった要求の一部は迅速に簡単に対処できるであろう。

自づから開発しなくても，メーカやソフトウェアハウスにて開発され流通されているソフトウェアを活用することによっても同様の効果が得られるであろう。

汎用性や簡易性を追い求めた場合，汎用的な機能の分析の難かしさや，開発工数の増加が考えられるが，その必要性やメリットを十分に説明し，開発部門や利用部門の理解を得なければならない。

特に，過去の事例や，汎用性に対する要求は運用者の立場で整理し，かつ説得性のある資料を用意しなければならない。

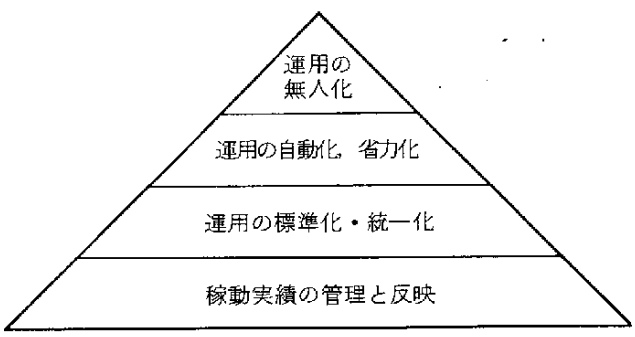
3.5.2 運用効率化への方策

ソフトウェア運用のうち，特にコンピュータ側からみた，運用の効率化，省力化対策について考えてみた。

具体的には，運用の標準化・統一化，運用の自動化，運用の無人化，そして，運用上の各種情報を記録・分析することの重要性について述べている。

これらの方策は，図表３－４に表わしたように段階を追って進めていくことが望ましい。

図表３－４　運用効率化へのアプローチ



(1) 運用実績の管理と反映

運用の効率化を図るための基礎データを集める必要がある。特に運用状況の継続的な監視と記録，各資源の利用度，利用者の特性など，広範囲に，詳細かつ信頼性の高いデータを収集したい。

これらのデータを整理し分析することによって，運用方式から，システム自体，ソフトウェア内部の問題まで，運用効率を妨げている要因を根こそぎ洗い出し，かつその改善策を講じたり，他部門へ提言することができる。

(2) 運用の標準化・統一化

運用効率化のためには，方策が決まっていたとしても，利用者や開発部門に対して説得力のある説明が必要である（通常，これらの部門とは，表面的な利害が相反する関係になりやすい）。

また，システムや運用ソフトウェアが持つ機能を十分に理解し，効果的な運用をするには，一般的な知識だけではまったく不充分である。広い分野に渡っての，高度でかつ勝れた知識と能力を有した運用担当者が求められている。

しかし，このような要員は，移動しやすく，また養成するにも時間・経験と費用がかかる。

そこで，運用要員の技能に依存することなく，質に左右されずに均一な運用サービスと効果を保証されるようにしなければならない。

そのためには，ドキュメントの統一化，運用作業の標準化を図ることである。

標準化・統一化のねらいは以下の点にあり，運用だけでなく開発から運用保守まで広く効果的である。

① 保守の安易化
② 運営管理上の効果
③ ソフトウェア開発期間の短縮
④ ソフトウェア信頼性の向上
⑤ ソフトウェアに対する理解と認識レベルの統一
⑥ 教育期間の短縮

(3) 自動化・省力化の推進

先に述べた，標準化・統一化については，あくまでも取り決め，ドキュ

メント上の記述であり，個人の受け止め方によって差が生じてくる。この問題を解決するため，方式論やツールの導入を図ることになる。

特に運用に対するツールでは，自動化機能が中心であり，これによって標準化され統一化された利用者サービスと運用が行われる。

① 利用者プログラム／ジョブ情報の管理ファイルからの取り出し
② ジョブスケジューリングの自動化
③ コンピュータ運転の自動化
④ 運用実績と予定の管理と通知
⑤ 処理結果の分類と配布
⑥ 媒体，格納棚，消耗品の管理
⑦ ソフトウェア資産の管理
⑧ 各種管理情報の自動収集と記録
⑨ ジョブ，プログラムのパターン化と自動生成
⑩ システムサービス状況の自動把握と通知

(4) 無人化の推進

無人運転実現のためには，先の標準化，統一化の定着，自動化での運用内容の把握と分析が十分であることと，利用者プログラムやシステム制御プログラムでの配慮，そして設備関係にも改造が必要となる。

無人化を指向するには，以下の点が基本的な条件となる。

① 操作員の介入が不要であること。またプログラム自体の品質が十分保証できる内容であり，障害発生に対しても十分な対策がなされている
② ハードウェアや基本ソフトウェアについても品質が保証できる
③ 緊急事態発生の検出機構と発生時の対応処置機能を有していること
④ 無人運転を行うパッケージが利用できること
⑤ 基本ソフトウェア，ハードウェア，施設面において対応する機能，装置などの導入・改良が可能であること

(5) 自動／無人運用ツール

自動／無人運用ツールを自社で開発するケースも多いが，最近では市販されているツールにも適用性が高く，また，ハードウェアやオペレーティングシステムとの連携もよく取られたものがあるため，多額の費用をかけて開発しなくても済む。図表３－５に代表的なツールについて紹介する。

図表３－５　自動／無人運転用パッケージ一覧表

分類	パッケージ名	提供／販売等の会社名
自動運転用運用管理用ツール	FIPS	日本電気㈱
	HOP SSⅡ	㈱日立製作所
	RECS	日本ユニバック㈱
	TASS	富士通㈱
	A－AUTO	㈱ソフトウェア・エージ・オブ・ファーイースト
	DSS	㈱協栄計算センタ
	CS2	㈱データエレクトロニクス
無人運転監視用ツール	AOS	日本電気㈱
	AOF	富士通㈱
	AOM	㈱日立製作所
	AUTOZAP	三菱電機㈱

3.5.3　外部資源の活用

要員不足，費用，ローテーション問題，そして労働条件などの問題を解消させるため，外部のソフトウェア会社からサービスを受けているケースが多い。アンケートからでも，コンピュータ部内では73.3％，運用部門に限っても６３.６％の企業が外部要員を活用している。

また，外部資源としては，要員だけでなく，ソフトウェア自体も対象として考慮したい。プログラムを作らなければ，保守も不要となるわけで，信頼性と汎用性の高いソフトウェアを積極的に導入することも運用の効率化に役立つ。

(1)　外部要員の活用

システムの運用において，問題となりやすい点の一つとして，要員不足がある。

要員不足を解決するために，現状の多くは，直接外部にその不足している部分を補充することを行っているが，メリットの裏に，多くの問題点を抱えていることも忘れてはならない。

（メリット）

①　要員不足の解消

②　運用経費の軽減

③ ジョブローテーション問題から離れる
④ 要員の固定化，専任化により，運用技術が向上する
⑤ 障害時，異常時の対応にも経験と技術に裏付けされた信頼性がある
⑥ 処理のピーク時や時間外など労働条件の悪化にも対応がとれる

（デメリット）

① 機密保護対策に特別注意を払わなければならない
② 運用技術が社内に蓄積されない
③ 担当者が止めたとき混乱が生じる
④ 管理体制が不明確になる
⑤ 運用担当者は，全社的視野や他部門までも考慮した行動はとらない

(2) 外部ソフトウェアの活用

外部で開発されたソフトウェアパッケージを活用することは，システム開発や保守作業において大きな効果が得られるが，運用に際しても同様に効果的な手段である。

しかし，この場合，むやみに流用するのでは利用者の理解が得られず，改造要求や機能追加の要求を出されて，結局新規開発することになってしまう。

ソフトウェアの流通を高めることの重要性が叫けばれてきていながら，いま一歩活発化しない理由の多くが，要求している仕様に合わない（若干であることが多い）とか，最適化するための改造要求やソースコードの公開が許されないといった点である。

この問題は重要であり，双方にとっても安易に妥協できる問題ではないが，今日のバックログ拡大，要員不足，保守工数の増大を考えた場合，ある程度の使いずらさ，手間の増加，出力様式の不満などが許容できる範囲であるか，ソフトウェアパッケージの活用によって得られる効果の程度はどうかを真剣に，かつ積極的に検討すべきである。

一方，提供する側（社内部門間でもソフトウェアの流用は良くあるため，すべてのソフトウェア開発では，提供する立場になりうる）においても，特定のシステム中心にまず考え，そこに汎用性を加味したやり方で開発されたソフトウェアを流通化するのではなく，設計当初から，流通性を十分に検討し，汎用性・柔軟性を高める機能を余分に加えたソフトウェア開発をするべきである。そのため，開発工数，期間の増加や，対象システムに

最適化されていないソフトウェアに見えたとしても，不満を持つ利用者の理解を求める努力と説得，そして決断をしなければならない。

3.5.4 開発段階での運用効率化対策

これまで述べてきたシステム運用の効率化対策は，運用部門で処理できる方策が主であったが，システム開発段階で考慮しておくことにより，運用効率化に大きく寄与する対策も多くある。

ここでは，それらについて特に効果的であり，これからのソフトウェア開発段階で是非とも検討してほしい項目について説明する。

(1) ドキュメンテーション

システム開発時点では，ソフトウェア関係のドキュメントだけでなく，運用関係のドキュメントについても用意されるようにしたい。

通常の運用マニュアルや操作マニュアルだけでなく，緊急時マニュアル，障害時マニュアル，復旧やバックアップ用マニュアルなどのドキュメントを体系付けて作成すべきである。

(2) データ中心型システムの構築

今までは，処理（プロセス）中心であり，プログラム中心であったシステムの開発方法を，改める動きが活発になってきている。

それは，データ中心型のシステム開発と称し，データベースを中核として，データの構造や関連性，データの利用／管理の面からシステムを構築していく方法であり，特に大規模なシステムでは，従来システムを壊わしてまで推進しなければならない状況にきている。

このデータ中心型システムの運用に際しては，運用部門に対して重要な役割りが課せられている（データ管理情報の充実，サービス性のチェック，情報構造の変化への対応，実績把握と評価，予測，各種要求への対応など）。

これでは，運用の効率化どころか，負荷拡大になってしまうが，この項で強調したい点は，直接的な運用維持作業の効率化ではなく，システム全体の運用が効果的に行われることを重要視しなければならない。このケースでも，データ中心型システムでは，システムの運用が大変重要になることから，運用部門や運用担当者のモチベーションや地位向上に役立ち，どちらかといえば軽視されてきた部分への見直しによって意識の高揚，スキ

ルの向上，それがミスの減少や効率化意欲の高まりとなって反映することも期待できる。

(3) 標準化

運用に際して，各システム間の運用システムに統一性がなかった場合，利用者や開発者が想像する以上に問題となる。

利用者や開発担当者は，一部のシステムに対してのみ担当することが普通であるため，他のシステムとの共通性や相違はほとんど気にならない。

システム開発に際しては，運用システムで，特に他システムとの統一性，関連性をふまえた開発が行われなければならない。

そして，運用部門も，積極的に意見や要望を出す必要がある。

(4) 汎用化・流用化の推進

“3.5.1 利用者部門の活用”でも述べたように，システム開発時に汎用性，流用性について十分な検討と，積極的な取り組みが行われることによって，運用段階での効率化にも結びついてくる。

(5) 簡易言語，エンドユーザ言語の活用

システムの運用に当って，対象プログラムを調べたり，内容を解折することは，保守部門はもちろん，運用部門でも良く要求される作業である（利用者からの問合せ，相談，トラブル時の処置依頼など）。

この場合，その都度保守担当や開発担当者へ依頼しているようではスムーズな運用はできても，利用者からの信頼は薄れてしまう。

このための努力は別にして，ソフトウェア自体が簡易言語やエンドユーザ言語で記述されていたとしたら，利用者自身で解決できるケースも多いと思われることと，運用担当者にとっても対応しやすいシステムになっていることであろう。

(6) 入力・出力方式の改善

運用システムの入力／出力方式については，利用者にとって最善の方式がとられていなければならないが，運用に際しても効率的な方法を取り入れるようにしたい。

例えば，

① カードレス

② ペーパレス

③ 大容量のファイルを用意し，磁気テープなどのハンドリング削減

④ オープン入出力方式による利用者直接操作

⑤ オートカッタ

⑥ メール機能

⑦ スプーリング機能による入出力方式の改善（オフラインによるデータ入力修正可，データの一元管理，装置専有時間短縮など）

(7) コードの統一

“(2)，データ中心型システムの開発”でも重要なテーマになるはずであるが，コードの統一は，小規模なシステム開発においても検討をおろそかにすることはできない。

コードの設計では，一面的な検討だけでなく，他システムや今後開発が予測されるシステムに対しても考慮に入れた設計が必要である。

一旦，コードがくい違ったままシステムの運用が開始された場合，データの利活用はもちろん，システムの結合の必要が生じた場合，コードの統一のための適合化作業や，ソフトウェアやデータの修正作業は想像以上に大きな工数と問題を表面化させる。

参 考 文 献

1) 「コンピュータ運用管理入門」日本電気㈱

2) 「分科会研究活動報告書」昭和５８年度版ラージシステム研究会

3) 「分科会研究活動報告書」昭和５９年度版ラージシステム研究会

4) 「ＥＤＰシステム監査」㈳日本能率協会

5) 「システム監査の方法」中央経済社

6) 「電子計算機システム安全対策基準（Ｓ59.8改訂）」 通商産業省

7) 「コンピュータ自書1984－85」㈶日本情報処理開発協会編

8) 「コンピュータの安全対策」日本電気㈱

9) 「情報システムの構築を目ざしたデータ管理運用の実際」事務管理第２５巻，第２号

10) 「システムの開発と運用」オーム社

4. ソフトウェア保守

4. ソフトウェア保守

アンケート調査結果によれば，ＳＥおよびプログラマは平均５４％の工数を保守にあてている。コンピュータ部門が抱えるバックログは年々増加の一途をたどっており，今後保守の重要性はますます高まるものと予想されるが，保守の体制や手続，保守を考慮した開発，などについてまだ不十分なところが多い。

この章では，はじめに保守の基本的概念にふれ，保守管理の基本事項の説明をしてから，応急／日常保守作業の管理を説明する。そのあと外注開発する場合の留意点，そして保守に関する技術とツールにふれる。

4.1 ソフトウェア保守の基本的な概念

4.1.1 保守のタイプ

ソフトウェアは開発されてから廃棄されるまでいろいろな変更を加えられる。その全ての変更を保守（メンテナンス）という。

保守をするにはそのタイプを識別することがまず大切である。保守する対象，保守の理由，および保守のタイミング（時間）によってつぎのように分類できる。

(1) 対象による分類
- データ／プログラム保守（data/program maintenance）
- ドキュメント保守（document file maintenance）
- システム保守（systems maintenance）

(2) 理由による分類
- 修理保守（corrective maintenance）
- 適応保守（adaptive maintenance）
- 完全化保守（perfective maintenance）

(3) 時間による分類
- 計画保守（scheduled maintenance）
- 予防保守（preventive maintenance）
- 応急保守（emergency maintenance）
- 遅れ保守（deferred maintenance）

つぎに簡単に説明する。

(1) 対象による分類

① データ／プログラム保守

データ（ファイルやデータベース）やプログラムを変更する場合である。

② ドキュメント保守

ソフトウェアに関連する仕様書，設計書やユーザ用マニュアルなどのドキュメントを変更する場合で，データやプログラムの変更に伴って行われる。

③ システム保守

操作方法，運用手続など，システム全体に関わる変更である。

(2) 理由による分類

① 修理保守

システムを運用するうえで検出されたさまざまなエラーを修正することをいう。

② 適応保守

コンピュータのハードウェアやオペレーティングシステムの急速な変化，あるいは応用ソフトウェアが最初に開発されたときからのシステム環境の変化に適応するための変更作業である。

③ 完全化保守

応用ソフトウェアが使われるにつれて，新しい要求や機能の変更，システム全体についての改善要求などがユーザから提出される。これを満たす変更作業をさす。

(3) 時間による分類

① 計画保守

予定された期日に計画的にソフトウェアの変更を行う場合である。

② 予防保守

将来保守が容易になるように，信頼性を高めるために，あるいは将来の改善を見込してソフトウェアを変更する場合である。

③ 応急保守

システム運用中に発生するエラーに対し，特に応急処置を必要とする場合に行われる変更である。

④ 遅れ保守

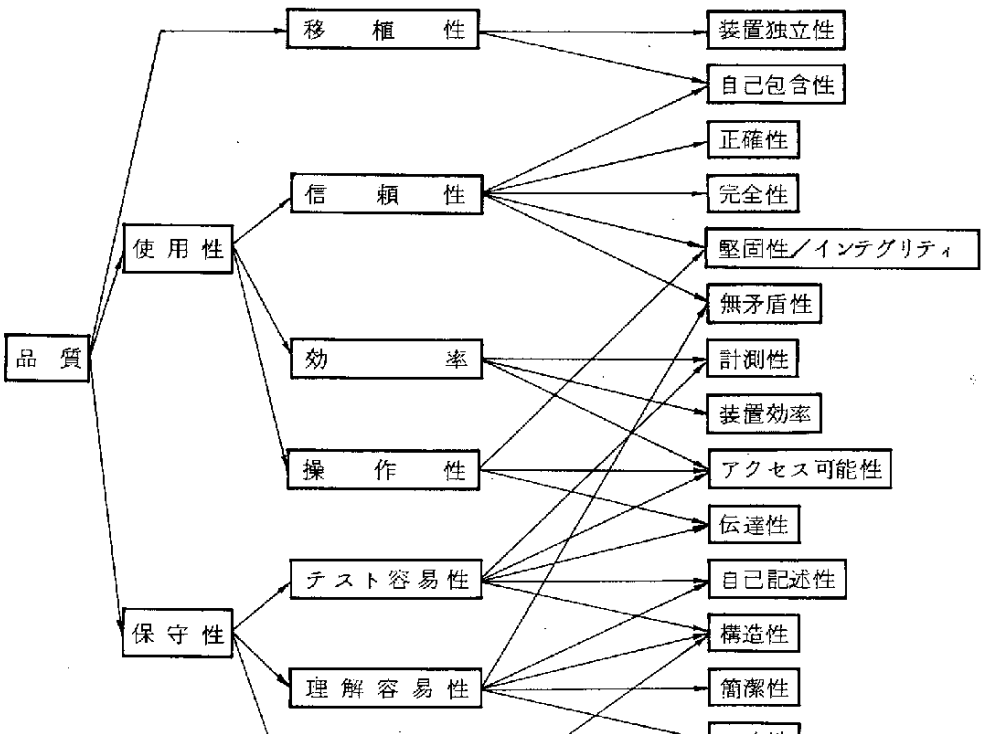
ソフトウェアの変更が必要であるけれども，急がない場合あるいはコンピュータ資源が直ちに十分に利用できない場合に，後で変更を行うことである。

なお，アンケート調査では理由による分類の保守の種類をきいてみたが，完全化保守が過半数を占め，その他修理保守と適応保守がそれぞれ2割であった。

4.1.2 ソフトウェアの保守性

ソフトウェアの保守性とは，保守が容易にできるように考慮されているかどうかをさす。Boehmによればソフトウェアの品質は図表4－1のような一種の階層構造をなしている。

図表4－1　ソフトウェアの品質特性（Boehm）

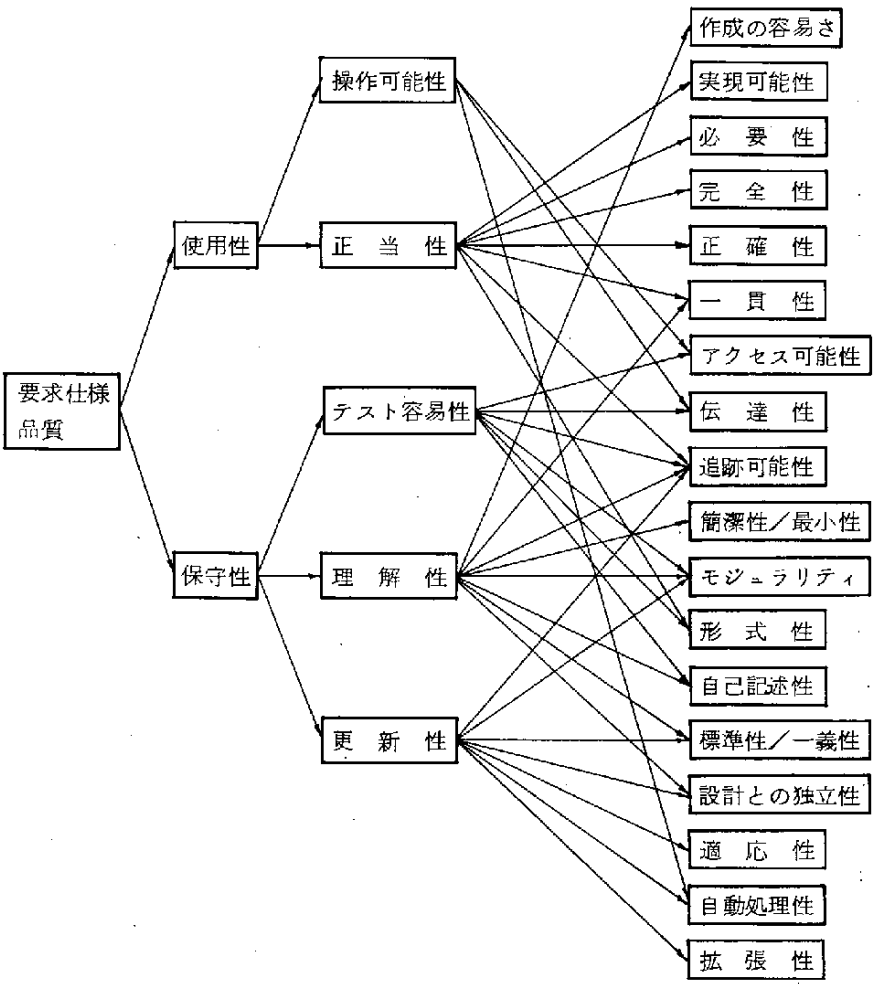


これによれば，保守容易性を持つソフトウェアは，理解のしやすさ，テストのしやすさ，変更のしやすさという品質性を持つ。

さらにこれらは，つぎのような特性に影響される。プログラムの内外に対し，記号，用語，表示法が統一され矛盾がないこと（無矛盾性），プログラムの動作状況をどの程度まで観察できるか（計測性），機能や装置を選択してどの程度自由に使用できるか（アクセス可能性），入出力の形式や内容がどの程度使いやすく統一されているか（伝達性），プログラム自身がどの程度，その目的，仮定条件，入力・出力，要素，改訂版の記号などを一目瞭然の形に記述しているか（自己記述性），また変更修正などが局所的にすむように，プログラムの構造化がなされているか（構造性），不要な情報は省き，有用なものだけがコンパクトに納められているか（簡潔性），プログラムが読みやすいか（明瞭性），拡張しやすいか（拡張性）などである。

ソフトウェアの保守を論じる場合にはこれだけでは十分でない。たとえばドキュメントからの観点も必要である。ここでは宮本が提唱している要求仕様書の品質尺度モデルを図表４－２に示す。

図表４－２　要求仕様書の品質尺度（宮本）



これらの特性のうち，重要なものをつぎに説明する。

(1)　実現可能性

要求仕様に対し，少くとも１つ以上の設計代替案がありうるとき，実現可能性があるという。解析的実現可能性（仕様を満足するアルゴリズムが存在する）と，実際的実現可能性（処理時間的に可能なアルゴリズム，データ・プロセッサ，ソフトウェア設計があるか）とがある。

(2)　必要性

仕様書に記述されている要求項目は，もともとの原始要求の目的に一致していなければならない。各仕様が原始目的に寄与しているとき，その仕

様は必要性にもとずいているという。

(3) 追跡可能性

仕様の各部分が，もともとの原始要求からの関係や，仕様に記述された要求項目への関係を追跡できるようになっていれば，要求の全体像や関連の把握，さまざまなチェックなどがしやすい。

(4) 形式性

仕様に形式性があることは，意図された事柄が正しく表現されていること（正当性）と密接に関係している。不正な記述は多くの場合，自然言語のような非形式的言語を使うことによってもたらされる。システム開発，保守のもととなる要求仕様は，正確な情報伝達のために形式性を持たねばならない。テスト容易性や自動処理性にとっても必要である。

(5) 設計との独立性

要求定義は特定の設計案を導くものであってはならない。問題を記述する際，最適な解決案を選択できる余地を残すために，解決案とは独立であるべきである。

(6) 適応性

要求仕様自体，適応性が高くなければならない。つまり，仕様自体設計と独立しており，モジュラリティが備わっていること，さらに，システム機能の他に性能，信頼性，保守性，融通性なども記述されていることが必要である。

(7) 自動処理性

仕様の記述，検証を助けるのに，どの程度機械処理できるか，ということである。

(8) 拡張性

要求仕様がどの程度容易に拡張できるか，という性質である。ソフトウェアのライフサイクルが長い場合，当初の要求仕様が拡張変更されることはよくあり，このような場合に拡張が容易にできる必要がある。

4.2 ソフトウェア保守管理の基本

4.2.1 保守の方針

ソフトウェアの保守を効率よく行うためには，コンピュータ部門としてソフトウェア保守に対する方針を明確にしておかねばならない。アンケート調

査では，過半数の部門で保守に関する方針があり，管理者はその実施に努力している実態がみられる。

保守作業上の方針は管理者レベルを中心として作られ，保守全体を作業担当者，担当部署，関係文書，手続き，技法，ツール，設備などの観点から規制し，方向づける。方針作りのための基準として，つぎのような項目があげられる。

(1) 保守の容易なソフトウェアを開発するような方向づけをすること
(2) 保守作業を支援するためにコンピュータ運用の手続きを設けること
(3) ソフトウェアの変更要求を承認するか却下するか判断する基準があること
(4) 複数の応用ソフトウェアに共通なエラーが発生しないように，情報の提供を行うものであること
(5) 徹底したテストを強制するものであること
(6) 保守の際に必要な文書の標準化を指示するものであること
(7) ユーザとの関係を重視したものであること
(8) 方針の実施にあたって上位管理者の強力な支援が得られるものであること
(9) 保守の作業においては，新システムの開発に必要な能力と同等かそれ以上の能力が必要とされることを明示していること
(10) 保守作業遂行の業績に対し，新システム開発と同等かそれ以上の評価（給与上の報酬など）を保証するものであること

従来，保守は開発よりやさしい仕事であると考えられてきた。あまり計画的でなくてもよく，専門的知識や技術もそれほど必要でなく，管理者の指示も重要ではないと思われてきた。したがって，保守は「新入社員の仕事」という通念があった。しかし，現在は新規の開発に比べて保守の占める割合は大きく，開発よりも面倒な保守の仕事はめずらしくない。実際，保守を行うにはシステムの全体にわたって深く突っ込んで調べなければならず，最初の開発者の考え方を十分に理解したうえでシステムの再構成をしなければならない。そのような意味で，保守は「専門家の仕事」と考えるべきである。

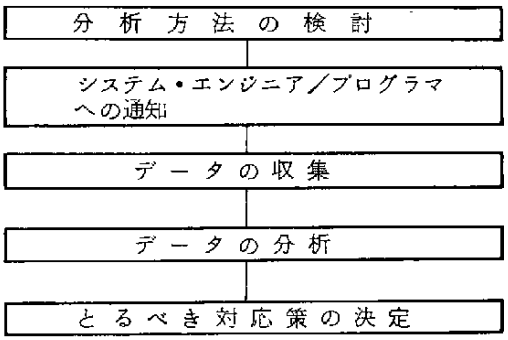
4.2.2 現状分析の方法

保守の作業を管理し，体系的な保守を行うためには，保守の現状を知らね

ばならない。つまり，コンピュータ部門における保守の現状を知り，そのうえで保守の方針を定め，後で詳しく説明する体系的な保守を目指すべきである。

現状分析は図表４－３のように５つのステップから成っている。

図表４－３　現状分析のステップ



それぞれのステップを以下に述べる。

(1)　分析方法の検討

一般的にソフトウェア保守にみられる問題と，そのうち個々のコンピュータ部門で特に困っている問題とを調べて，分析の対象を絞る。たとえば，ドキュメンテーションの問題，プログラミング言語の問題，コーディング基準の問題，要員の問題，管理の問題，技術の問題などがあげられる。

(2)　システムエンジニア／プログラマへの通知

信頼性のある情報を得るために，保守の担当者の理解と協力を得なければならない。理解してもらう点として，

①　ソフトウェア保守に多くの設備・資源を費していること

②　非体系的な保守作業は，システムエンジニア／プログラマに無理や無駄をもたらしていること

③　エンドユーザからよりよい保守の強い要望があること

④　保守作業に用いるツール・技術や手続きにまだまだ改善の余地があること

⑤　コンピュータ部門として，保守作業を改善したいと望んでいること

⑥ 保守作業の担当者から，エンドユーザのサービスレベルを維持しながら，保守をやりやすくするための提案を期待していること

⑦ 保守の現状を改善するために，現状の正確な情報が必要なこと

⑧ 実際の保守担当者であるシステムエンジニア／プログラマのみが信頼のおけるデータを提供しえること

⑨ 結果的には各個人の保守作業環境を改善するための定量的データと共に，各個人の協力が必要であること

⑩ データ収集はできるだけ短時間で簡単に実施したいこと

⑪ 集められたデータは保守環境の改善にのみ使用し，人事考課には用いないこと

などがある。

(3) データの収集

実際のデータの収集にあたっては，定型化された標準フォーマットがあればよい。これにはワープロなどのコンピュータの文書ファイルを利用するとよい。

標準フォーマットに記入すべき主な項目としてつぎのようなものがある。

① 応用ソフトウェア名

② 変更番号

③ 型；「修理保守」，「適応保守」，「完全化保守」のタイプ別

④ 変更名

⑤ 受付日付

⑥ 変更要求日付

⑦ 運用開始日付

⑧ 保守区分；「データ／プログラム保守」，「ドキュメント保守」，「システム保守」の区分と，「計画保守」，「予防保守」，「応急保守」「遅れ保守」の区分

⑨ 工数（予定と実績）

⑩ 変更要求元；ユーザの保守担当者に限らず「法的規制」などもありうる

⑪ 重要度（プライオリティ）

データ調査票はできるだけすみやかに（翌日など）集める。

(4) データの分析

収集されたデータは，ステップ1での分析対象にしたがって分析する。一般的に考える分析として，

① 応用システムごとの保守状況
② カテゴリごとの保守状況
③ 変更作業の工数による保守状況
④ 型ごとの保守状況
⑤ 変更の要求元による保守状況
⑥ 重要度による保守状況
⑦ 変更の重要度と変更作業の工数との関係
⑧ 新規要求と変更作業工数との関係
⑨ 拡張変更と変更作業工数との関係
⑩ 変更数の分布
⑪ 変更作業工数の分布
⑫ 予定工数と実績との比較

(5) とるべき対応策の決定

分析結果をもとにとるべき対応策を決定する。基本的につぎの3つがある。

① 対応策をとらない

現状のままでよいと判断されるが，当面は修正の必要がないと判断される場合。

② 詳細な調査を実施する
③ 変更作業を始める

これまでの調査分析で問題点が明確に把握され，変更に着手することが決定された場合。

4.3 応急保守

コンピュータの運用中にはさまざまな障害が発生する。大別するとハードウェアが原因のもの，オペレーションのミスが原因のもの（エンドユーザ，オペレータ），そしてソフトウェア（OS，応用ソフトウェア）が原因のものがある。

いずれにしてもそのときは応急の処置を施し，すみやかに運用を再開しなけ

ればならない。

この節では応用ソフトウェアの障害に対する応急保守作業について述べる。

4.3.1 応急保守の特徴と管理の方針

(1) 応急保守の特徴

応急保守はその緊急性のために，日常の保守に比べ，つぎのような特徴を持つ。

① 管理者やユーザの正式の承認を得ずに保守を始めることがある

② 形式的手続を後回しとして作業を先行させることがある

③ 障害の程度によってはすべてのコンピュータ運用スケジュールに優先して作業を進めることがある

このような緊急性を有する応急保守作業については，あらかじめ応急保守の対象となる緊急性の度合を設定しておかねばならない。これは各企業の運用方法に関係して定められることだが，一般には「２４時間以内」に何らかの処置（復旧，回避など）をする必要があるものを表すことが多い。

(2) 応急保守のための管理方針

管理方針には

- 障害の文書化
- 設備・資源の優先的使用
- 修正の許可
- 方針にもとずく手続きの徹底
- 応急的処置から正式処置への移行

といった項目があげられる。

以下簡単に説明する。

① 障害の文書化

緊急事態が生じた場合は，復旧が第一で文書化は後回しにされがちである。そのため，つぎのような項目について検討すべきである。

- 保守担当者が障害を復旧する前に，必要な文書化の範囲
- 保守管理者のレビューの復旧作業前の必要性
- 文書化の目的の明確化

② 設備・資源の優先的使用

緊急性を有するために優先的使用によって復旧を図る。

・応急的保守に使用可能な設備・資源
・その使用基準
・不正使用を防ぐ手段
・使用のための許可手続き

③ 修正の許可
・許可なしで修正作業を開始する範囲
・許可を与える者
・許可判断する人がいない場合の処置
・文書以外（電話連絡など）の手続き

④ 方針にもとずく手続きの徹底
・手続きの強制の限度
・手続きが守られない場合の処置
・例外事項について

⑤ 応急処置から正式処置への移行

応急保守はあくまでも臨時の処置であり，早めに正式処置へ移す必要がある。このときつぎのような項目を検討する。
・応急処置の保持手続き
・正式処置に移行するまでの仮運用期間の限定
・正式処置の検証

4.3.2 応急保守での担当者の役割

前項で定めた方針に従い，障害発生の緊急事態に対処できるよう，コンピュータの運用にかかわる担当者についてその役割をあらかじめ決めておく必要がある。

(1) コンピュータ部門の管理者の役割
・部門の応急的保守の方針策定
・方針の実施
・応急的保守のための設備・資源の優先的使用に関する管理

(2) オペレータの役割

実際にコンピュータ室でオペレーションに携れるので，運用には注意が必要である。通常コンピュータ室にはつぎの点について記された障害処理のマニュアルが整備されている。

・障害発生時の状況の記録
・障害の解決に必要な情報（ダンプリストなど）の保持

(3) コンピュータ運用管理者の役割
・応急保守のための関係者を召集する
・保守のために必要な情報が集められているか確認する
・設備・資源の優先的使用の許可と管理

(4) 保守担当者の役割
・障害の原因の追求
・応急的処置作業
・応急的処置から正式処理への移行

(5) エンドユーザの役割

障害によって直接影響をうけるのはエンドユーザである。そのために，
・障害や問題点の発見に努め
障害が復旧したあとは，
・応急処置の結果の承認
を行う。

4.3.3 応急保守の管理手続き

管理の方針に従って応急保守を遂行するための管理手続きを設定しなければならない。これは保守を管理する文書の書式と手順（マニュアル）を設定することになる。

応急保守の管理手続きとしては，
・関係者への連絡
・ＯＳ特権命令の使用
・データ保持
・障害の記録
・承認手続き
・作業コントロール
・応急的処置から正式処置への移行
などを含む。

(1) 関係者への連絡

あらかじめ障害の発生にそなえて，関係者への連絡・召集体制を整えて

おく必要がある。これは応用システム毎に作成され，関係者へ連絡する緊急度合の条件，連絡先（システム保守担当，エンドユーザ，メーカ，ソフトウェアハウス）のリストと連絡体制などがわかるようになっていなければならない。

(2) ＯＳ特権命令の使用

ＯＳには特別な場合に対処できるように，機械語レベルの変更機能，パスワード制御の変更，システム起動・停止・再開，などについて特権命令を持っている。しかし使い方を一歩誤ると重大な事態となるため，その使用には細心の注意が必要である。ここでは，使用のための許可条件，使用可能者，使用機能の範囲，使用許可手続，使用記録，などに関して基準を作成しておく必要がある。

(3) データ保持

コンピュータ室のオペレータは，障害発生にそなえてあらかじめとるべき手段について十分に教育・訓練されているはずである。後のプログラムのデバッグのための基礎資料として，障害発生時のさまざまなデータを保持し収集しておかねばならない。そのためにデータ保持のチェックリストを作成しておくとよいだろう。

(4) 障害の記録

障害が発生した場合，定められた書式に従って障害の状況を記録しておく必要がある。記録すべき項目としては，状況の記録として，

- システム名
- 発生年月日
- 障害発生内容
- 障害の影響度

などを記録し，復旧のための処置として，

- 応急処置に要する時間と資源の推定
- 応急処置のリスク
- 今後の運用への提案

などを含む。

(5) 承認手続き

障害を記録した調査票をもとに，実際に修正するかどうかの判断が下される。もちろんさらに詳細な分析を必要とする場合もあろう。

(6) 作業コントロール

応急的保守の管理上大切なのは，

・承認済み変更作業の監視

・未承認変更の禁止

である。前者については，作業に連番をふって識別し，作業を監視すればよい。後者についてはささいな変更と担当者がみなしている場合などに承認されてない変更を行うことがある。これを禁止するために管理方針の徹底を図る必要がある。

(7) 応急的処置から正式処置への移行

応急保守はあくまでも一時的なものである。そのままにしておくともとのシステムは次第にゆがんだ形に変形され，ますます保守しにくいものとなってしまう。従って応急処置のあと必らず正式処置の要求を起こすようにしておけばよい。

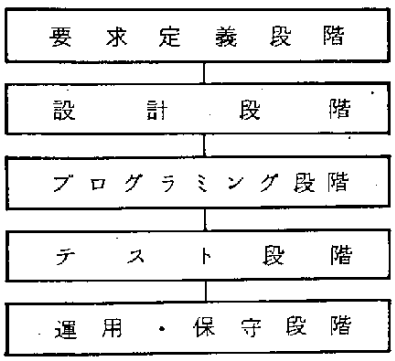
4.4 日常保守の管理

4.4.1 ソフトウェア保守のライフサイクル

ソフトウェア開発におけるライフサイクルという考え方はよく知られている。

「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（I）－開発計画」（(財）日本情報処理開発協会，昭和５９年３月）では図表４－４のようになっている。

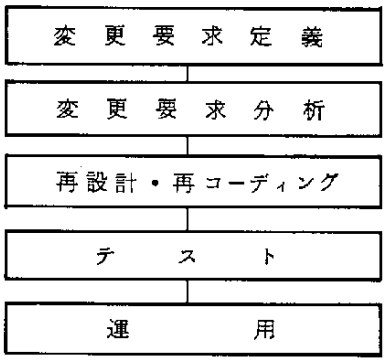
図表４－４　ソフトウェア開発ライフサイクル



このライフサイクルの基本的な考え方は，ソフトウェアの特徴である不可視性をいかにして可視性のあるものにして品質や進捗状況を管理するかにある。つまりソフトウェアを開発する段階をいくつかの工程に分け，各工程の作業のアウトプットとしてのドキュメントを作成して開発作業の確実な進捗を管理するのに利用している。

ソフトウェアの保守でもライフサイクルの考え方をとり入れて，図表4－5のように5つの段階に分けることができる。

図表4－5　ソフトウェア保守ライフサイクル



アンケート調査によると，保守のライフサイクルという概念はほとんど使われていないが，このような工程に分けて管理することが大切である。

つぎに，各段階を簡単に説明する。

(1)　変更要求定義

ソフトウェアの保守要求を型ごとに記録する段階である。また実際に変更するかどうかや，変更の工数見積り，作業する担当者などをきめる。

(2)　変更要求分析

変更要求を詳細に分析し，コスト効果などを考慮して，変更作業を実際に進めるかどうか決定する。

(3)　再設計・再コーディング

変更のために再設計・再コーディングを行う。

(4)　テスト

変更が正しく実施されたこと，システムは正確に機能することなどをテストする。また変更に伴って運用手続きにも変更がある場合にはユーザに

対する再トレーニングが行われる。

(5) 運　用

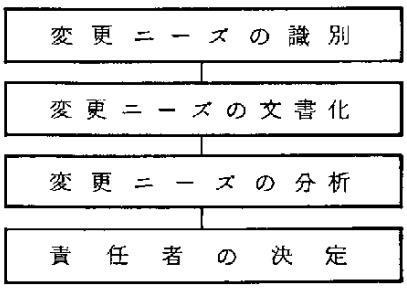
変更されたソフトウェアを正式の運用に移し，新たな問題が生じないように監視する。

以下，各段階でとくに詳しく述べる。

4.4.2 変更要求定義

この段階の作業の流れを図示すると，図表4－6のように4つのステップから成る。

図表4－6　変更要求定義の作業の流れ



(1) 変更ニーズの識別

変更ニーズには，エラーばかりでなく，新規の要求，機能の拡張の要求などがある。これらのニーズを正確に識別する手順を整えておかねばならない。

変更ニーズにはつぎのようなものがある。

① エラー

- オペレーション・エラー
- インターフェイス・エラー
- プログラム・エラー
- データ・エラー

② 新規要求

- 入出力変更
- 処理機能変更

③ 拡張要求

・機能拡張
・操作性改善
・品質改善

④ 法的ニーズ
・税法規の変更
・医療関係法規の変更

(2) 変更ニーズの文書化

変更ニーズが識別されたならば，そのニーズが正確に文書化されて記録されねばならない。変更ニーズはエンドユーザからばかりでなく，システムの担当者からも起こりうる。エンドユーザが記述すべき主要な項目としては，

・現システムの問題点
・変更ニーズ
・変更完了期限
・プライオリティ

などがあり，システム担当者用としては，

・現システムの問題点
・変更ニーズ
・影響範囲

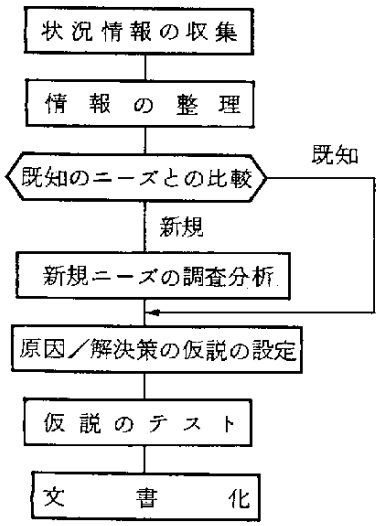
などがある。

(3) 変更ニーズの分析

ここではエラーの原因の分析，あるいは新規要求などに応えるための修正箇所の分析などを行うため，システム全体を熟知している者が行うべきである。

分析作業は，図表4－7の7段階で実施するとよい。

図表４－７　変更ニーズの分析



① 状況情報の収集

ニーズに関する情報をできるだけ集める。

② 情報の整理

収集されたさまざまな情報を整理して，有用な構造を見い出す。

③ 既知のニーズとの比較

同じような変更ニーズは過去に発生していることも多い。既知のニーズを調べることも有効である。

④ 新規ニーズの調査分析

過去に似たような変更ニーズがなかった場合，新しい変更ニーズなので慎重に調査分析しなければならない。

⑤ 原因／解決策の仮説の設定

考えられるいくつかの原因や解決策に関し，仮説をたてる。

⑥ 仮説のテスト

仮説をテストして，正しいことを検証する。

⑦ 文書化

調査分析の最終結果として，原因や新規要求などを実現するための修正箇所を文書化しておく。

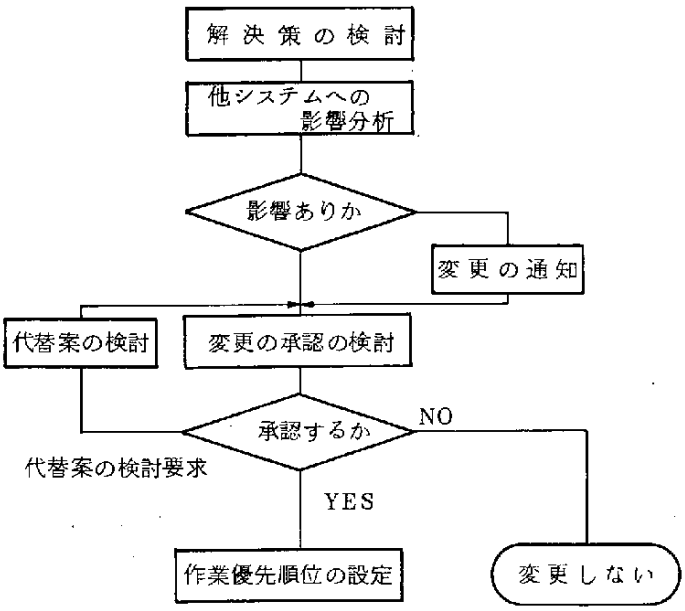
(4) 責任者の決定

変更ニーズを正確に効率よく実現する責任者を決定する。

4.4.3 変更要求分析段階

この段階の作業は図表4－8のように7つのステップから成る。

図表4－8 変更要求分析の作業の流れ



(1) 解決策の決定

このステップでは，前の段階で示された一応の解決策をレビューして決定し，応用システム内の影響範囲を文書化する。詳しくはつぎの4部分に分けられる。

- 解決策と影響範囲の識別
- データの変更の文書化
- プログラムの変更の文書化
- 他の変更の文書化

① 解決策と影響範囲の識別

解決策が決定されたならば，応用システムのどの部分に影響するか識別する必要がある。そのためには仕様書，設計書，マニュアルなどのドキュメントをレビューするのみならず，ユーザやオペレータにインタ

ビューすることも必要である。

② データの変更の文書化

処理プロセスのみの変更ならば影響は小さいが，データ（ファイルやデータベース）の変更の場合影響範囲は広い。一貫性，理解性，信頼性に注意してデータ変更について文書化されるべきである。

③ プログラムの変更の文書化

データ変更の場合は多くのプログラムの変更をひきおこす。厳密な保守管理のためには個々のプログラムの変更の記述を文書化しておかねばならない。

④ その他の変更の文書化

データやプログラム以外にも，支援システムの変更や運用手続き（エンドユーザ，オペレータ）の変更などがあり，これらの変更についても文書しておくことが必要である。

(2) 他システムへの影響分析

変更にともなって，影響する他の応用システムとそのインパクトを調べなければならない。直接インターフェイスを持つシステムばかりでなく，一見無関係にみえるシステムを注意しなければならない。

影響範囲は文書化して，関連するシステムの担当者に通知する準備をしておく。

(3) 変更の通知

他のシステムへ影響する場合は，影響する変更箇所を関連システムの担当者に通知する手順を決めておく必要がある。

(4) 費用／効果の分析

法律の改正に伴う変更のように，費用に関係なく必ず変更する場合もあるが，一般には変更の費用と効果を分析して実際に変更するかどうか決めることになる。

直接的な費用の削減につながる場合は効果の算定をしやすいが，間接的な効果は算定がむずかしい。

(5) 変更の承認検討

実際に変更作業をするかどうかの判断を，ユーザの責任で行う。

(6) 代替案の検討

最初の解決策が承認されないこともある。そのときユーザは代替案の提

案を要求する必要がある。

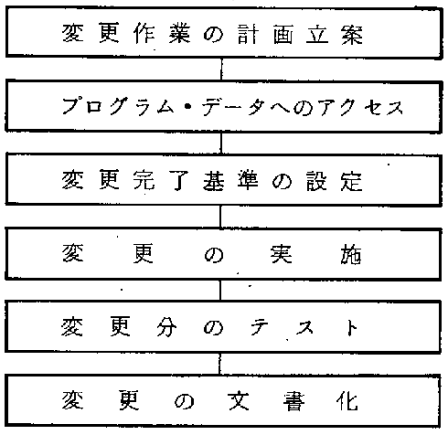
(7) 作業優先順位の設定

承認された変更については，ふつう，作業の優先順位を設定する。先着順や，個々の作業を逐一検討する方法，グループ分けによる優先順位の決定などの設定法が考えられる。

4.4.4 再設計・再コーディング段階

この段階の作業はつぎの図のように6つのステップからなる。

図表4－9　再設計・再コーディングの作業の流れ



この段階で留意すべきことは，新規開発のときとちがって修正の対象となるシステムが運用されていることだろう。現行の運用が業務に与える影響を把握し，現行システムで変更作業中の業務をどのように進めるか，検討しておかねばならない。

(1) 変更作業の計画立案

小さな変更作業の場合は詳しい計画はいらないだろうが，長期にわたる変更作業の場合は詳しい計画を立てる必要がある。

(2) プログラム・データへのアクセス

プログラムやデータへは保守担当者といえども保守作業の期間中のみアクセスが許可されるべきである。

(3) 変更完了基準の設定

変更作業が完了したかどうか判断するための基準をあらかじめ定めておく。基準の内容にはつぎのようなものがある。

① 機能の達成

ユーザ要望の機能が達成されているか。

② 波及効果の回避

変更が，変更の対象となっていない部分に悪い影響を与えていないか。

③ 負荷

特に重い負荷の下で正しく稼動するか。

④ 性能

費用／効果に見合う性能が実現されているか。

(4) 変更の実施

このステップには，詳細なシステム設計，プログラム設計，それにコーディングが含まれる。このステップのポイントはつぎのとおりである。

① 変更する変数はできるだけ少なくする

② 共通モジュールを変更するときは，それを呼んでいるモジュールを全部調べる

③ 1個のモジュールのみで使用される変数を変更する場合は，そのモジュールで参照される他の変数が変更の影響をうけるかどうか調べる

④ 複数のモジュールで使用される変数を変更する場合は，その変数を参照するモジュールを全部調べる

⑤ 一度に複数の変更を実施する場合は，変更の小さいモジュールから順に変更し，変更の波及効果をみきわめる

(5) 変更分のテスト

変更対象にもとづいて行う個々のテストと，各モジュールをシステムに再統合する際に行う回帰テストを組み合わせて行うのがよい。

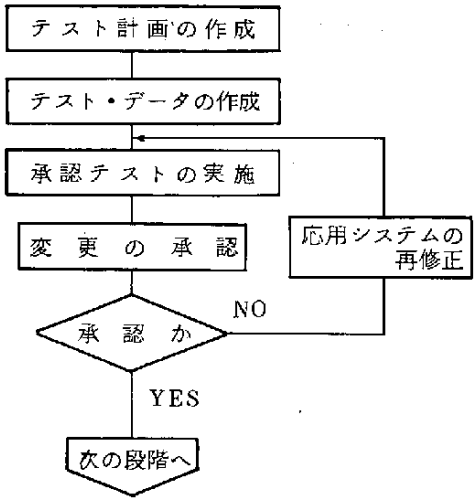
(6) 変更の文書化

この段階で行われた変更は，きちんと文書化しておく必要がある。

4.4.5 テスト段階

この段階の作業は図表4－10のように5つのステップから成っている。

図表４－１０　テストの作業の流れ



ここではユーザが中心となる承認テストが実施される。

(1)　テスト計画の作成

新規開発のテストとはちがって，保守の場合は短期間に変更点に集中してテストを行う。

テスト計画にはテストの目的，テスト方法，テストによる期待される結果を含む。さらに変更により，変更されない部分が意図しなかった動きをしないかどうか確認する回帰テストも行われる。さらに複数のシステムに関係する変更ならば，システム間のテストも計画しなければならない。

(2)　テストデータの作成

開発時に作成されたテストデータが利用できることも多い。新規の部分のテストには新しいテストデータが必要になる。また新規開発のときとちがって現在の運用環境を利用できる場合もある。

(3)　承認テストの実施

ユーザが中心となって実施され，保守チームはユーザを助ける。保守担当者が用意するチェックリストに従って，実際の運用環境に近い条件の下でテストを実施する。

(4)　変更の承認

基本的には開発時と同じ要領である。変更要求に合ったシステムになっていれば，承認される。承認の責任はシステム担当者でなく，主にユーザ

にあることを忘れてはならない。

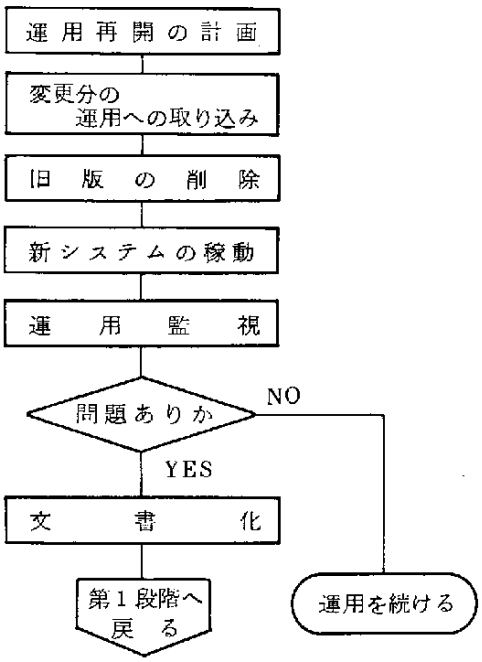
(5) 応用システムの再修正

テストの結果，承認できない部分があった場合は，応用システムの修正をやり直し，テストをやり直す。

4.4.6 運用段階

この段階の作業は図表４－１１のように６つのステップから成っている。

図表４－１１　運用の作業の流れ



(1) 運用再開の計画

運用を再開するに当っては，一時的にシステムをダウンさせて新システムに移行し，運用を開始することになる。このとき，ファイルやデータに影響する場合には回復の手続きが必要となる。

(2) 変更分の運用への取り込み

承認テストによりユーザの承認を得たならば，正式の運用に移される。このとき変更の履歴を応用システムごとに記録しておくことが望ましい。また変更の実施については口頭でなく文書で通知すべきである。

(3) 旧版の削除

新しい版が入れられたときは，古い版は削除されるべきだが，新版にいくらかでも不安があれば，安全のために旧版もしばらく保存しておいた方がよいだろう。

(4) 新システムの稼動

新システムは，ユーザやオペレータの操作手順の変更があればその変更も行われた上で稼動を開始する。

(5) 運用監視

新規開発の場合と同様，稼動直後に問題が発生することが多い。システムの処理結果のチェックがしばらくは必要である。

(6) 文書化

チェックの結果問題がある場合は，文書化してこれまでの保守過程の最初に戻る。

4.5 外注する場合の保守に関する考慮点

コンピュータ部門では，システム開発段階からさまざまな局面で外注の活用が図られている。工数や要員の不足，あるいはスキル・経験の不足から外注を利用することが多い。外注の形態には業務委託（一括外注）および外部要員がある。

とくに，ソフトウェア開発を外注した場合に問題となるのは，ソースプログラムの保守性と保守に関する契約である。

4.5.1 外注先に示すコーディング基準

外注先にソフトウェアの開発を委託した場合，プログラミングを外注先の自由に任せると，運用開始後に保守に入ってからプログラムを見ても理解できないことが多い。

従って各企業が独自にコーディング基準を作って，外注先に遵守させることが必要である。アンケート調査でも，外注先に対して保守性を重視したプログラミングを注文しているところが過半数ある。

ここでは，そのようなコーディング基準の一例（ＣＯＢＯＬの場合）を示す。

(1) 変数名は，その物理的あるいは，機能的属性をうまく表現したものとする。

(2) 手続き名は，処理内容を適切に表現したものとする。

(3) プログラムの各モジュールの最初には，つぎの記述を行う，見出しブロックを含めること。

① プログラム名，機能

② 作成日，作成者

③ 目的

④ 制限や制約

⑤ 障害発生時の回復処理手続き

⑥ 修正変更の履歴

⑦ 入力と出力

⑧ 処理方法の説明

⑨ 仮定，前提条件

⑩ 当該モジュールを呼んでいるモジュール，および呼ばれているモジュール

(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をつけること。

(5) 判断点と継続する分岐は適切に説明する。

(6) 異常終了の条件はすべて説明する。

(7) 算術表現式は意味がわかるようにコーディングする。

(8) 1行には1つの代入文や実行文が書かれていること。

(9) コメント文の形式は全モジュールで統一する。

(10) ある手続きに制御が移るもとの分岐がわかるようにコメントする。

(11) 1つの算術式の中に繰り返して，同じ表現を使わないこと。

(12) 繰り返し実行しない命令は，ループの外に書くこと。

(13) 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を書いておく。

(14) プログラムを修正した場合には，いつ，だれが，どのような修正をしたか明確に解るようにしておく。

(15) コーディング形式は全モジュールで一貫していること。

(16) 入力データ構造を定義するときに，将来の増加にそなえて多めにとっておくこと。

(17) ラベル付き共通ブロックの組は，空行や注釈で明確に区切られ，理解し

やすくなっていること。

⑱ 全プログラムで共通な宣言文や定義文は，全プログラムで一貫した形で書かれていること。

⑲ モジュールは，１つの機能から成っていること。

⑳ プログラムは，常に上から下に制御が流れるようになっていること。

㉑ PROCEDURE DIVISION は原則としてプリンタ用紙１枚以内とすること。

4.5.2 保守に関する契約上の要件

保守に関する契約上の要件としては，開発ソフトウェアの納入後あるいは検収後の保証に関する事項と，納入ソフトウェアに関して常時とる保守の体制に関する事項の２つがある。

(1) 納入ソフトウェアのかし担保責任

契約書には，かし担保責任，保証期間，保証範囲について盛り込まねばならない。アンケート調査によれば，これらの点については，約半数が明確にしている。

① かし担保責任

かし（ソフトウェアのエラー，バグ）が発見された場合，開発者（外注）が無償で修正することを明確にするものである。

② 保証期間

契約書には，責任をもって迅速に，かつ，無償で修正する期間を明確にすべきである。全工程の一括受注やそれに近いものは１年，部分工程の場合やかしの有無の判断が速やかなものは６ヶ月とすることが多い。

③ 保証範囲

エラーの内容にはさまざまなものがあって，委託側がエラーと考えても開発者は仕様の変更（有償）を受けとったりして，その判断が難しいものもある。発注者と受注者の間でトラブルを防ぐために，保証の範囲を明確にしておいた方がよい。

(2) 保守契約

納入したソフトウェアの修正に対し優先的に対応したり，あるいは修正の有無にかかわらず常時保守サービスの体制をとるものであって，有償で行われる。契約は上記の開発とは別途に保守契約を結ぶ場合と，必要があ

った都度に契約する場合がある。

なお，①の保証期間中でも，かしでなかった場合（保証の範囲を越える場合）は，有償の保守となる。

4.6 ソフトウェア保守の技術とツール

4.6.1 ソフトウェア保守技術

ソフトウェア保守に関する技術は，新規開発の時に利用する技術とほとんど同じであり，ツールの形で提供しているものが多い。

この節では保守技術の分類を，構成制御に関する保守技術，運用監視／評価に関する保守技術，再作成に関する保守技術，文書化に関する保守技術，コード生成と解析に関する保守技術，検証と確認に関する保守技術，そしてテストとインテグレーションに関する保守技術に分け，それぞれの区分にどのようなものがあるかを列挙する。

(1) 構成制御に関する保守技術
  ① 支援ライブラリ
  ② 自動化再構成
  ③ 状況レポーティング

(2) 運用監視／評価に関する保守技術
  ① 性能分析
  ② 障害エラー情報
  ③ 性能監視
  ④ 自動回復

(3) 再作成に関する保守技術
  ① 要求決定
  ② 再設計分析
  ③ 設計言語

(4) 文書化に関する保守技術
  ① 文書化支援
  ② 自動文書化

(5) コード生成と解析に関する保守技術
  ① 構造化プログラミング（ＳＰ）
  ② フォーマッティング

③ プリプロセシング
④ 再構造化
⑤ 標準化
⑥ コード比較
⑦ オプティマイゼーション
⑧ 自動変更
⑨ コード・レビューイング
⑩ デバッギング

(6) 検証と確認に関する保守技術
① 静的分析
② パス構造分析
③ 変則分析
④ 変数分析
⑤ インターフェイス分析
⑥ 到達性分析
⑦ 記号実行
⑧ 正当性証明
⑨ 表明チェック
⑩ 動的分析
⑪ 実行分析
⑫ パス・フロー分析
⑬ タイミング分析
⑭ 使用カウンティング
⑮ インタラクティブ実行

(7) テストとインテグレーションに関する保守技術
① テスト記述言語
② テスト・データ生成
③ テスト・ケース選択
④ 達成度分析
⑤ ドライバ／スタブ
⑥ テスト・ファイル管理
⑦ 出力分析

⑧ テスト状況レポーティング

⑨ テスト・ベッド

⑩ シミュレータ

⑪ 回帰テスト

4.6.2 保守用ツール

保守用ツールは，既に述べたように，ほとんどが新規開発用のツールを保守にも利用しているものである。その中でも特に保守に有用と思われるものの例を，図表４－１２に示す（(社) ソフトウェア産業振興協会・ソフトウェア流通促進センターのプログラム調査簿による)。

図表４－１２　ソフトウェア保守関連ツール

ツール名	会社名	対象言語／システム
TEST DATA GENERATOR	SRA	UNIVAC 1100
FORTUNE V1	富士通	FORTRAN／FACOM M
SDSS Ⅰ~Ⅳ	富士通	FACOM M
FORTRANフローチャータ	構造計画研究所	FORTRAN／FACOM，ACOS
MTS	富士通	FACOM M
DATAGEN	マイネス	FACOM M
F-SCAN	構造計画研究所	FORTRAN／FACOM 他
LIME／MLS	日立	HITAC M
MSP GEN V10	富士通	FACOM M
LAXYM	三菱総研	FORTRAN／IBM360他
CSP	JSD	COBOL／ACOS
DYANA	SRA	IBM 他
MAID	西武情報センター	COBOL／FACOM
PL／I INDT	東京ガス	PL／I ／ IBM360 他
SMART	IBM	IBM370
PAMAS	日本タイムシェア	ACOS
FIPS	NTIS	ACOS
GEM V1, V2	富士通	FACOM M
COBOL／SPA-4	NTIS	COBOL／ACOS
FORTRAN／SDA-4	NTIS	FORTRAN／ACOS
クイックチャート	構造計画研究所	COBOL／FACOM230
HIDOC	日立	COBOL／HITAC M
COBALT	三菱総研	COBOL／IBM370
FLOWAR	三菱総研	FORTRAN／IBM370
ILIB	NEC	ACOS
CORAL	日立	HITAC M
SYSTEM-EAST	JSD	FORTRAN／FACOM 他
PROPA	日立	HITAC M
PTSS	三菱総研	IBM 他
DREAM	NBC	COBOL／HITAC M
C-QUAT	インテック	FACOM M 他
MSF	電力中研	COBOL
SDMS	NEC	ACOS
PAVES	NTT	DIPS
MSS	三井情報開発	UNIVAC
SMEF	JSD	FORTRAN他／スーパーミニコン

参 考 文 献

1）宮本勲「ソフトウェアの保守」　bit 連載記事

2）菅野孝男「ソフトウェア開発のマネージメント」

3）宇都宮公訓「ソフトウェアの開発効率と評価」，総務庁

特に1）の文献は大変有意義なものであったことを付記し，感謝の意を表する。
（ハワイ大学の宮本先生にお断りしたうえで参考にさせていただいた）

5. ソフトウェア評価

5. ソフトウェア評価

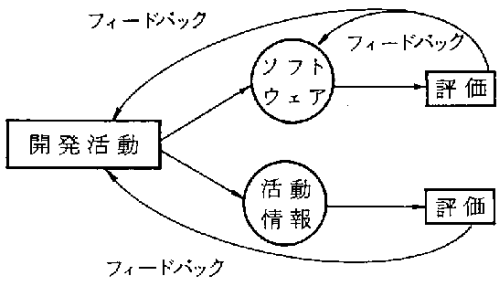
5.1 評価と改善の枠組み

(1) 開発活動およびソフトウェアの評価

評価作業は，開発活動の評価と完成したソフトウェアそのものの評価に分けることができる。前者は，その結果をつぎの開発活動にフィードバックさせ，より優れた開発のやり方を実現するために行うものである。後者は，その結果に基づきソフトウェアそのものを改良することと，その結果をもたらした開発活動にその原因を深り，より優れた開発方法を策定することの２つの目的を持っている。その概念を図表５－１に示す。

いずれにしろ，評価を行う時は，その結果を何に対しどのようにフィードバックするかを明確に意識しておかなければならない。

図表５－１　開発活動およびソフトウェアの評価



(2) 評価の時点による評価目的の違い

評価作業は，評価を行う時点によって，開発過程で行われるもの，開発完了時点で行われるものおよび運用開始後に行われるものに分けられる。３者には大きな違いがあるように考えられがちであるが，その目的，方法ともに本質的な違いはない。

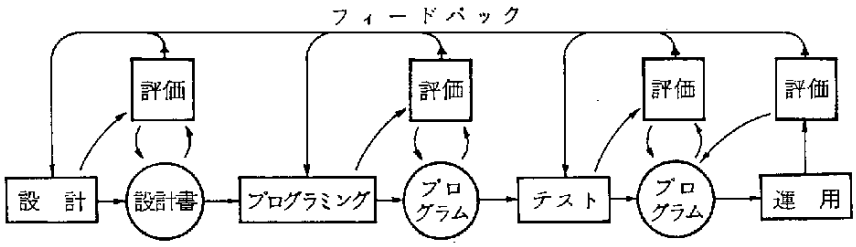
開発過程における評価作業は，各工程でのアウトプットを評価し，次工程に不良品を渡さないこと，および，作業のやり方そのものを評価し，作業のやり方の改善を図ることを目的とする。

開発完了時点で行う評価作業は，製品（ソフトウェア）そのものを評価し，利用者に迷惑を与えるものを市場に出さないことであり，また，開発活動全体を評価し，より優れた開発活動を策定することである。

運用開始後の評価は，運用中のソフトウェアの改善を行うとともに，改善事項の原因となった開発活動にその根本原因を探り開発活動の改善を策定することである。

これらの概念を図表５－２に示す。なお，本報告書の位置付けから考え，この章では，開発完了時点および運用開始後の評価作業について触れる。

図表５－２　評価の時点とフィードバック



(3)　評価項目

評価を行う主な目的は，ソフトウェア品質の向上，開発コストの削減，開発期間の短縮である。

ソフトウェア品質の評価項目としては，信頼性，処理効率，使いやすさ，保守のしやすさなどがある。開発コストの評価項目は，投入工数，使用計算機時間が主なものである。開発期間の評価項目は，各工程の完了日の計画からのずれである。

しかし，これらの評価項目の評価だけでは，ソフトウェアの改善を行うことはできるとしても，工程改善（開発活動の改善）に結び付けることはできない。工程改善を行うためには，これらの評価項目の代用特性になると考えられるデータを詳細に集収しておき，その評価も必要である。代用特性として，プログラムサイズ，工程別発見バグ数，原因別バグ数，レビューでの指摘事項数，ドキュメント枚数，テスト項目数，コメント率，ＧＯＴＯ文率，テストカバレージ率，工程別作業工数などが考えられる。

5.2　生産性の評価

(1)　生産性の尺度

ソフトウェア開発の計画立案時の重要な項目の一つとして投入予定工数がある。投入予定工数を策定するためには，生産性の尺度が必要であるが，ソ

フトウェア開発においては，確固たる生産性の尺度がないのが実状である。

ソフトウェアの規模の尺度として，ファンクションポイント（入出力の数とその特性からそのソフトウェアの持つ機能を定量化した値）が用いられている事例もあるが，ソースプログラムの行数を用いるのが一般的である。

しかし，生産性の基準値については極めて曖昧な状態にある。今回の調査では，新規ソフトウェア開発においても，生産性基準値があると回答した企業は１０％弱であり，プログラムの改造（基準値ありは５％），プログラムの再利用（基準値ありは３％）においては，ほとんど基準値が定められていない。

このような実状を考えると，ソフトウェア開発完了時点でしっかりと生産性の評価を行い，生産性の基準値を策定しておく必要がある。

(2) 生産性に影響を与える要因

確固たる生産性の基準値が存在しない理由は，ソフトウェアの場合は生産性に影響を与える要因が非常に多いため，１つの企業では常に異なった条件の下に開発作業を行うことになり，基準値の設定が困難になるからだと考えられる。

しかし，条件を同一にするか，非常に多くのデータを集めれば基準値の設定も不可能ではないはずである。それではその条件を何にするかが問題になるが，その条件の設定事例について以下に紹介する。以下，その条件のことを生産性要因と呼ぶことにする。

生産性要因の事例として，Boehm（TRW社）の提唱しているＣＯＣＯＭＯ見積りモデルで採用している生産性要因とその影響度合を図表５－３に示す。また，同表中に，ＣＯＣＯＭＯ見積りモデルを自社用にカスタマイズした宮崎幸生（富士通㈱）のモデルでの各要因の影響度合を示した。さらに，今回の調査では，この生産性要因のうちどれが生産性に影響を与えると思うかを回答してもらったが，その結果も同表中に示した。なお，１人４項目の選択回答方式としたので全体が４００％となっている。

図表５－３　生産性に影響を与える要因と影響度合

生産性に影響を与える要因	影響度合		アンケート
	COCOMO	宮崎幸生	回答率
要求されるソフトウェアの信頼性	３３％	１０％	２２％
データベースのサイズ	１１	２５	２
プロダクトの複雑さ	４８	２８	５１
システムに課せられた実行時間の制約	３３	７３	１０
主記憶の制約	２８	－	４
DBMSやOSの変更頻度（ＯＳが安定しているか）	２２	－	２
プログラム開発時のコンピュータ・ターンアラウンドタイム	１４	１８	１２
設計者の能力	３８	－	８６
開発作業環境	－	－	４０
同様なアプリケーションの経験の度合	２４	２４	４４
プログラマの能力	３６	４０	５４
プロジェクト・メンバーのＤＢＭＳ，ＯＳの経験	１６	１９	１７
使用されるプログラム言語の経験	１０	１０	１０
生産技術（ストラクチャードプログラミング技法等）	２１	１９	２８
使用ツール（ソフトウェアＣＡＤ等）	２１	１０	２１
開発期間	１７	－	－

（注）影響度合：無影響を１とした場合，±に与える影響範囲の平均を示した。

図表５－３を見ると，プロダクトの複雑さ，設計者の能力，プログラマの能力が，ＣＯＣＯＭＯ，アンケート調査回答ともに上位となっている。宮崎幸生のモデルでは設計者の能力が生産性要因から外されているが，他の２つとシステムに課せられた実行時間の制約が上位となっている。実行時間の制約はＣＯＣＯＭＯでも４番目になっているが，それは，システムによってはその制約が非常に大きい場合があるからである。アンケート回答でその値が低いことは，一般的にはその制約を感じることがないことを意味している。

これら以外の生産性要因で各データともに高い値を示しているのは同様なアプリケーションの経験度合である。これらのデータから，生産性に影

響を大きく与えているのはやはり人的な要因であることが裏付けられている。

(3) 生産性の見積式

生産性を評価するには，標準工数を定めるための正確な見積式が必要である。その式による見積値からの差異によって個別のシステムの生産性を評価することができる。評価に当っては，単に生産性の高低を論じるのではなく，標準からの差異の原因を分析してつぎの開発活動に役立てる必要がある。

代表的な見積式の例として，以下にＣＯＣＯＭＯの見積りモデルを紹介する。

ＣＯＣＯＭＯでは，基本ＣＯＣＯＭＯ，中間ＣＯＣＯＭＯ，詳細ＣＯＣＯＭＯの３種類の見積りモデルを定めている。

基本ＣＯＣＯＭＯは，ソフトウェアをつぎの３種類のモードに分類して簡易見積式を定めている。

- 基幹モード：普通のＯＳ，コンパイラ，アプリケーション
  工数＝2.4（開発Ｋステップ）$^{1.05}$
- 一般モード：新規ＯＳ，新規ＤＢＭＳ，簡単な指揮統制システム
  工数＝3.0（開発Ｋステップ）$^{1.12}$
- 組込モード：複雑な大規模ＯＳ，複雑な指揮統制システム
  工数＝3.6（開発Ｋステップ）$^{1.20}$

なお，開発Ｋステップには，ソースプログラム，ＪＣＬ，フォーマット文，データ宣言文などユーザに引渡されるすべての行数を含めているが，注釈行は除いている。

基本ＣＯＣＯＭＯは，見積値が実績値の±２０％におさまる確率が２５％であり，十分な見積式とは言えない。その精度を高めたのが中間ＣＯＣＯＭＯである。中間ＣＯＣＯＭＯの見積式はつぎのとおりである。

工数＝仮工数×係数１×……×係数１５

ここで，仮工数とは基本ＣＯＣＯＭＯで求めた工数である。また，係数とは，図表５－３の各生産性要因のレベルによって定めた値である。

中間ＣＯＣＯＭＯで求めた見積値が実績値の±２０％におさまる確率は６８％である。

詳細ＣＯＣＯＭＯは，中間ＣＯＣＯＭＯに工程別配分やプロダクトレベ

ル（モジュール，サブシステム，システム）を考慮したものであるが，中間ＣＯＣＯＭＯと精度において差が少ないのでここでは説明を省略する。

宮崎幸生（富士通㈱）は，自社のシステム開発事例３１件を分析してＣＯＣＯＭＯ見積りモデルのカスタマイズを行った。

宮崎の分析では，ＣＯＣＯＭＯと同じ１５個の生産性要因を使用した場合の見積式では，実績値が見積値の±２０％以内におさまる確率は４８.４％であった。ＣＯＣＯＭＯの生産性要因から，主記憶の制約，ＤＢＭＳやＯＳの変更頻度，設計者の能力の３つを除いた場合の見積式では，実績値が見積値の±２０％以内におさまる割合は６７.７％となり，ほぼＣＯＣＯＭＯに等しい精度が得られている。その時の実績値と見積値の差の３１システムの平均は１５.３％となっている。

5.3 信頼性の評価

(1) 信頼性評価の必要性

近年におけるコンピュータ・システムの高性能化，ネットワーク化に伴い，その利用分野が急速に拡大，多様化しているとともに，システムの大規模化，高度化がもたらされている。このため，そこで用いられるソフトウェアも大規模かつ複雑なものとなっており，ソフトウェアの信頼性を確保することが技術的にますます困難となる一方，コンピュータ・システムにおいて障害が発生した場合の社会的影響も大きなものとなっている。このため，ソフトウェアの信頼性向上が強く求められている。

しかしながら，ソフトウェアの開発は，従来からプログラマなどの労力に大きく依存しており，ソフトウェア開発技術が必ずしも十分には工学化されていない状況にある。このため，工学的アプローチによりソフトウェアの信頼性を評価するための技法，ツールの開発，実用化が極めて遅れており，これが基本的にソフトウェアの信頼性確保を困難なものとしている。しかしながら，このような状況下においてもさまざまな信頼性評価の試みは行われている。本節ではそのような事例の幾つかを紹介する。

(2) 信頼性を直接的に評価できるのは，運用開始後に発見されたバグ数である。その尺度には単位ステップ中に発見されたバグ数を使用するのが一般的である。

運用開始後の信頼性の評価は，開発活動全体を評価してその結果を工程改

善にフィードバックするのに有効であり，重要なことではあるが，そのソフトウェアの利用者にとっては手遅れである。

それ以前にやっておかなければならないことは，出荷直前に信頼性を評価し，欠陥商品を市場に出さないことである。従って，本節では出荷直前の信頼性評価方法を中心に報告する。

(3) バグ成長曲線による信頼性評価

出荷直前のソフトウェアの信頼性評価は，テスト過程での発見バグ数の収れん度を分析することによりある程度可能である。

テスト過程における発見バグ数は，ゴンペルツ曲線（図表５－４），または，ロジステック曲線に従って成長することが知られている。各々の式はつぎのとおりである。

- ゴンペルツ曲線

$$y = k \times a^{bt} \qquad 0 < a\ ,\ b < 1$$

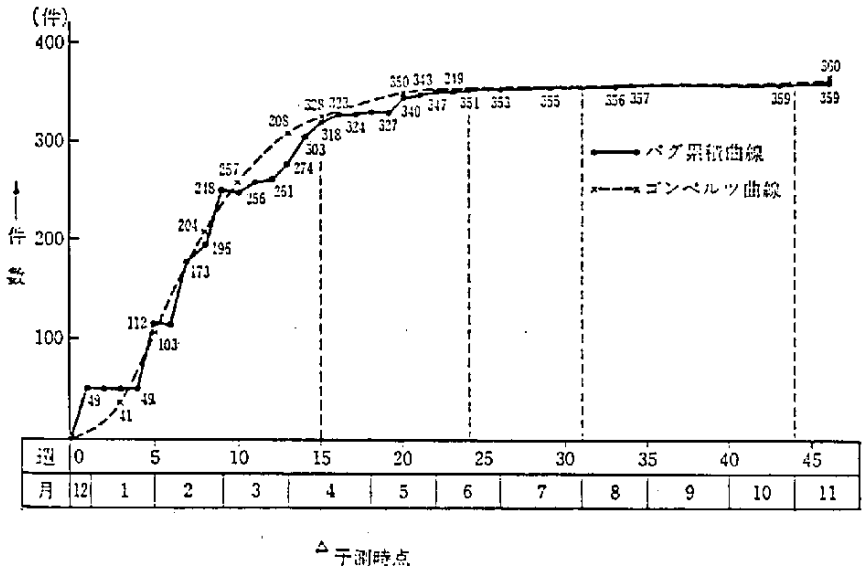
- ロジステック曲線

$$y = k / (1 + m \times e \times p\ (-at))$$

ここで，k は $t \to \infty$ の収束値，t は時間，y はバグ数

テスト過程で発見されたバグ数を上記の式に当てはめることにより，本来どのような曲線でバグが成長すべきであるかを推測することができる。テスト後期における発見バグ数が成長曲線の下にある時は，テストの不十分さが予測され，テスト項目の見直しを行う必要がある。また，発見バグ数が成長曲線を上回る時は，プログラムに何らかの異常があることが予測され，設計書あるいはソースコードの再レビューが必要である。

図表５－４　ゴンペルツ曲線に従った発見バグ数の収れん



（出所）　菅野文友「ソフトウェアエンジニアリング」

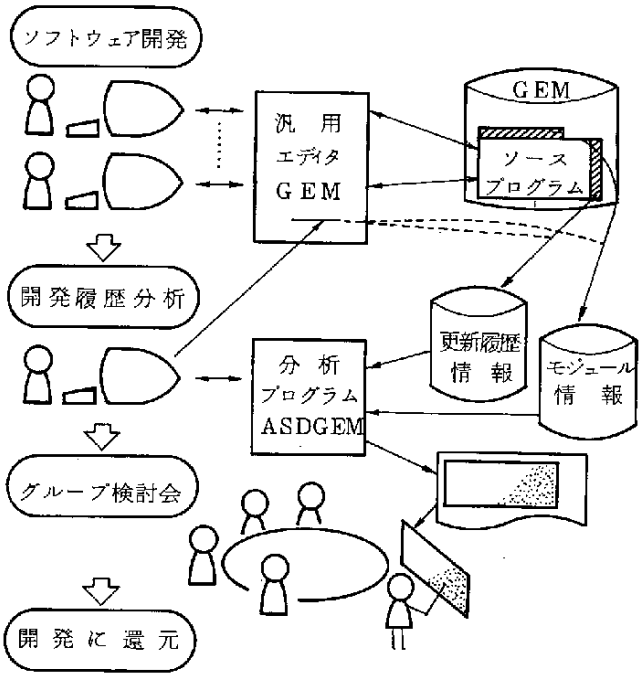
(4)　ソフトウェア開発履歴分析

中川徹（富士通㈱国際研）のソフトウェア開発履歴分析によるソフトウェアの信頼性評価方法について紹介する。

開発過程におけるバグ数の収集は，非常に根気のいる作業であるとともに，作業者の申告によってバグ数の把握を行うため信頼性の乏しいデータとなりがちである。バグ数以外の完成ソフトウェアの信頼性を評価する手段を考えてみる必要がある。

本事例では，プログラムの更新履歴を自動的に収集・分析することによりソフトウェアの信頼性評価を行っている。その手順を図表５－５に示す。

図表５－５　ソフトウェア開発履歴分析の運用



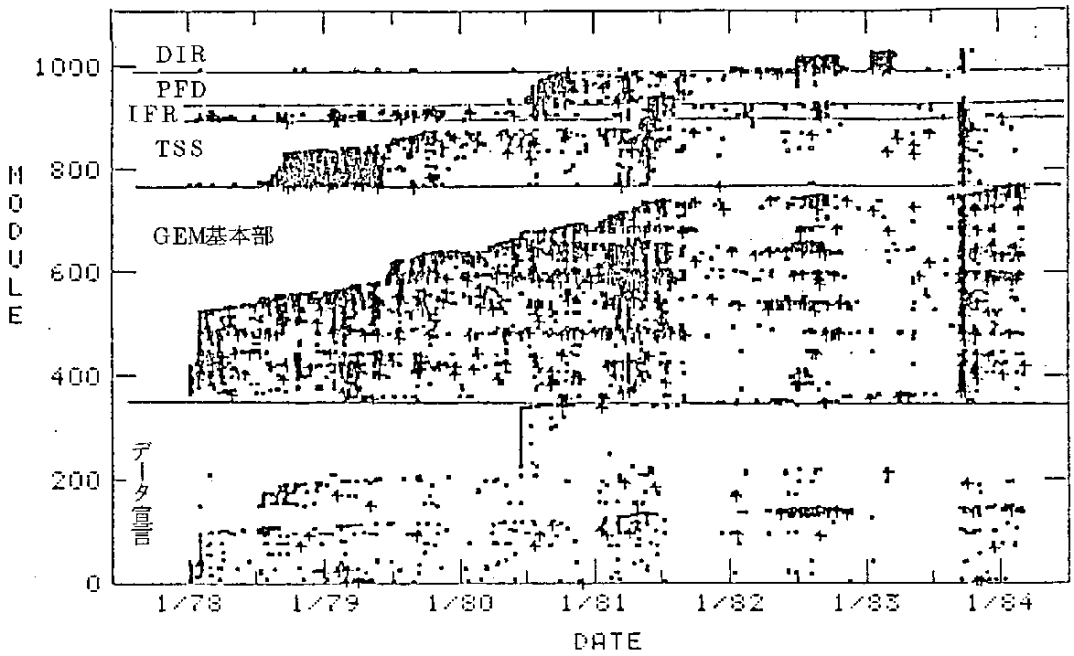
プログラムの更新履歴を自動的に把握するためのライブラリ管理ツールは数多くあるが，その情報を信頼性評価のためにいかに活用するかは重要な課題である。本事例は，分析結果をすべてグラフ表現して視覚的にその結果を訴えかけているところに特徴がある。

分析の基礎となるデータは，モジュールの機能分類，作成者などのモジュール属性情報と，どのモジュールを，いつ更新し，その時に何行投入して，何行削除したかを記録したモジュール更新履歴情報である。これらの情報から２３種類の分析グラフを作成している。その中からつぎの２例を紹介する。

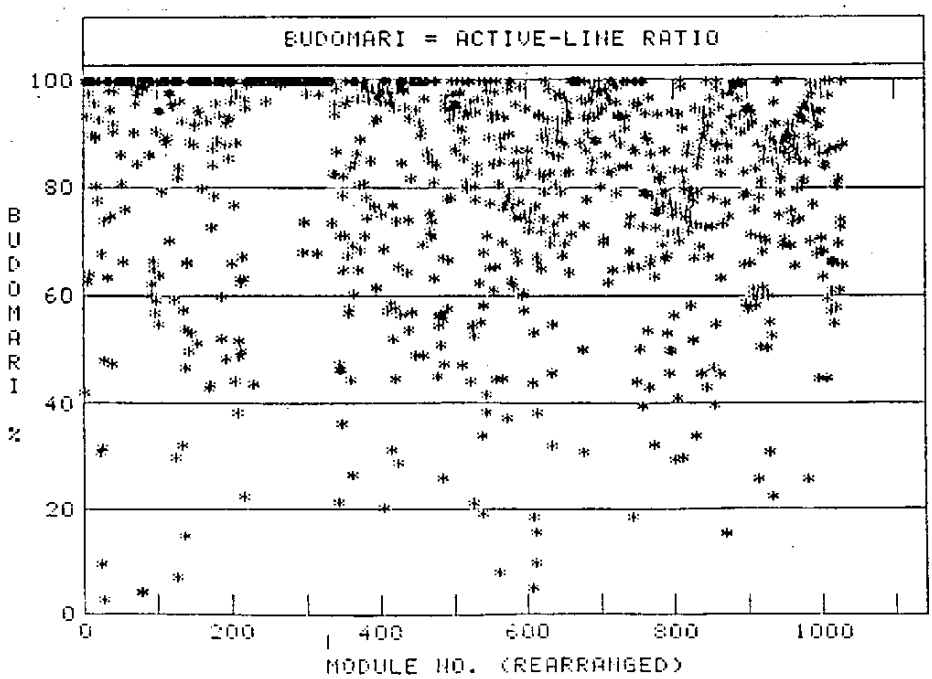
① 横軸に日付をとり，縦軸にサブシステム別にグループ化したモジュール番号をとり，いつどのモジュールに対し更新が行われたかをプロットしたグラフ（図表５－６）

② 横軸にモジュール番号をとり，縦軸にモジュールの歩留り（当初投入行数の何％が未修正で残っているか）をとったグラフ（図表５－７）

図表５－６　更新履歴の分析



図表５－７　歩留りの分析



図表５－６からは，いつ頃どのサブシステムに多くの更新が行われたか，また安定状態にあるサブシステムはどれであるかなどが良く把握できる。図表５－７からは，どのモジュール群に多くの更新が行われたかを一目で把握でき，このグラフも信頼性評価の一助とすることができる。

(5) 信頼性の影響要因

本当にソフトウェアの信頼性を評価できるデータは運用開始後に発見されたバグ数である。しかし，ソフトウェアの信頼性評価が意味を持つのは出荷直前である。従って，出荷までに得られるデータから信頼性を評価しなければならない。出荷までに得られるデータのうちに信頼性に影響を与えるデータがあれば，それらのデータを評価することにより出荷ソフトウェアの信頼性を間接的に評価することができる。

そのようなソフトウェアの信頼性の影響要因を定量的に分析した一つの事例を紹介する。

高橋宗雄（日本電信電話㈱）は，２２個のモジュールについて，発見されたバグ数と信頼性影響要因の相関を定量的に分析した結果を報告している。その結果を図表５－８に示す。

図表５－８　信頼性影響要因と影響度

信頼性影響要因		影響度(%)
開発規模	流用規模(KS)	7.7
	改造率(%)	29.9
設計・製造経験年数の平均（年）		33.4
ソースコード中のディバッグカードの割合(%)		9.6
開発時の信頼性	単体テスト机上検出バグ数(件/KS)	7.2
	結合テストマシン検出バグ数(件/KS)	12.2

(6) 代用特性の評価

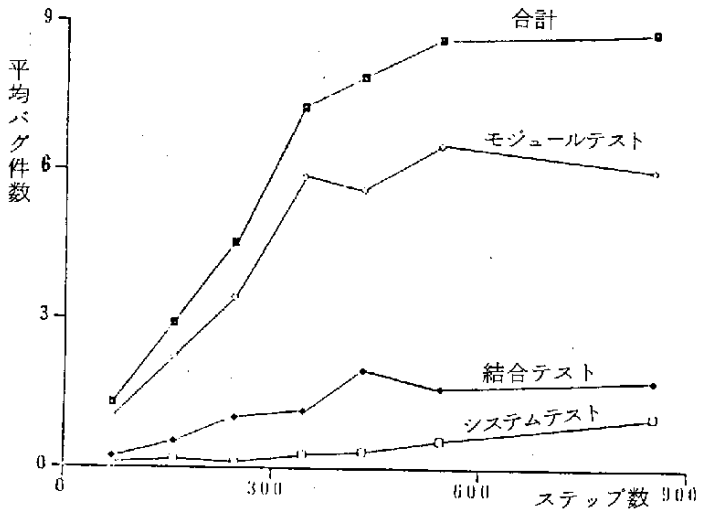
効果的な評価作業は，単に製品評価だけに終るのではなく，その結果を工程改善に結び付けることのできる評価作業である。そのような評価作業のうち特に重要であると思われることは，信頼性の代用特性を捜し出すことである。もし十分確かな代用特性を見つけることができれば，開発活動の過程で

それを管理することにより信頼性の高いソフトウェアを作り出すことができる。そのような事例として，宮崎幸生（富士通㈱）の「U管理図の考え方による品質データの分析」を紹介する。

本事例では，モジュールステップ数とバグ発見率，ドキュメント枚数とバグ発見率の相関を分析して，モジュールステップ数やドキュメント枚数が信頼性の代用特性となり得るかを評価してその管理限界値を求めている。

図表５－９はモジュールステップ数によって各テスト工程で発見されるバグ数がどのように変化するかをグラフ化したものである。

図表５－９　ステップ数とバグ件数



このグラフからつぎのことが分かる。

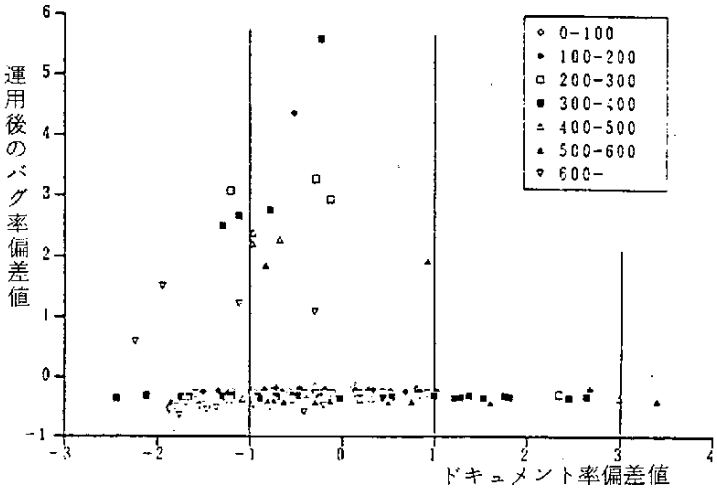
- モジュールステップ数が３００を越えるとモジュールテストで検出されるバグ数が余り変わらなくなる
- 結合テストでは４５０ステップ前後で同じような傾向になる
- システムテストでは単調に増加する

以上のことは，モジュールが大きくなればなる程バグの検出時期が遅れることを示している。これは，運用開始後でも，大きなモジュールには単位ステップ当りにより多くのバグが含まれていることを意味している。従って，モジュールの大きさは３００～４００ステップ以下にすることが望ましい。

図表５－１０は運用後の発見バグ率（単位ステップ当りのバグ数）がドキ

ュメント率（単位ステップ当りの設計書枚数）によってどのように変化したかをグラフ化したものである。このグラフから，運用後に発見されたバグはドキュメント率が平均以下のモジュールに偏っていることが分かる。

図表５－１０　ドキュメント率と運用後のバグ率



(7)　グラフによる代用特性の総合的評価

以上に紹介したような事例により代用特性の有効性は明確である。しかし，１つや２つの代用特性でソフトウェアの信頼性をうんぬんすることはできない。数多くの代用特性を用いて総合的に評価する必要がある。そのような評価を行うためのグラフ化の事例としてＦＰＴ（ファコム・ハイタンク㈱の作業標準）を紹介する。

本事例では，システム，サブシステムおよびモジュール単位に図表5－11に示すようなグラフを作成して，多くの代用特性によりソフトウェアの信頼性を総合的に評価している。

図表５－１１　グラフによる代用特性のチェック

モジュール品質チェック票（プログラム）

サブシステム名：例題　／　プログラム名：REIDAI　／　モジュール名：

・単純チェック（@=①）　・実績チェック（@=②）　／　検印

工程	品質特性		基礎データ		評価用データ		評価グラフ（$\frac{x-\mu}{\sigma}$）	母平均（μ）	母標準偏差（σ）
SD	サイズ	見積りステップ	① 1,500 ステップ		評価用換算式	換算値(x)	(負)−3σ　−2σ　−σ　μ　+σ　+2σ　+3σ(過)		
PG	サイズ	実積ステップ	② 1,894 ステップ		②/①(対見積り比)	1.26		(1)	(0.3,0.2)
MD	ドキュメント量	モジュール設計書	③ 90 枚		③/@×1000	48 枚/Kステップ		(50)	(10, 8)
MD	ドキュメント量	モジュールテスト仕様書	④ 13 枚		④/@×1000	6.9 枚/Kステップ		(20)	(4, 3)
MD	ドキュメント量	モジュール結合テスト仕様書	⑤ 4 枚		⑤/@×1000	2.1 枚/Kステップ		(20)	(−, 3)
PS	プログラム構成	モジュール数	⑥ 3		@/⑥(平均ステップ)	631 ステップ		(300)	(100, 75)
MD	プログラム構成	サブモジュール数	⑦ 51		@/⑦(平均ステップ)	37 ステップ		(50)	(12, 10)
PS	プログラム構成	結合延べモジュール数	⑧ 4		⑥/⑧(モジュール削減率)	0.75		(0.6)	(−, 0.18)
PS	プログラム構成	結合延べステップ数	⑨ 2,200 ステップ		@/⑨(ステップ圧縮率)	0.86		(0.7)	(−, 0.13)
PG	ステップ構成	命令部	⑩ 921 ステップ		⑩/⑩+⑪+⑫	37 %		(45)	(11, 9)
PG	ステップ構成	定義部	⑪ 723 ステップ		⑪/⑩+⑪+⑫	30 %		(30)	(8, 6)
PG	ステップ構成	コメント	⑫ 730 ステップ		⑫/⑩+⑪+⑫	31 %		(25)	(5, 4)
PG	ステップ構成	COPY句ジェネレイト部	⑬ 144 ステップ		⑬/⑩+⑪+⑫	6 %		(15)	(5, 4)
MD	項目テスト数	モジュールテスト	⑭ 51 件		⑭/@×1000	26.9 件/Kステップ		(50)	(15, 12)
MD	項目テスト数	モジュール結合テスト	⑮ 30 件		⑮/@×1000	15.8 件/Kステップ		(33)	(−, 9)
PG	バグ数	モジュールテスト	⑯ 18 件		⑯/@×1000	9.5 件/Kステップ		(10)	(4, 3)
IT	バグ数	結合テスト	⑰ 5 件		⑰/@×1000	2.6 件/Kステップ		(2.3)	(0.5,0.4)
ST	バグ数	システムテスト	⑱ 2 件		⑱/@×1000	1.1 件/Kステップ		(0.3)	(0.1,0.08)
OT	バグ数	運用テスト	⑲ 0 件		⑲/@×1000	0 件/Kステップ		(0.05)	(0.02,0.01)
OT	バグ数	合計	⑳ 25 件		⑳/@×1000	13.2 件/Kステップ		(2.65)	(2.5,2.0)
							0.1%　2.2%　13.5%　34.2%　34.2%　13.5%　2.2%　0.1%		(モジ, プロ)

このグラフではどの代用特性が管理限界を外れているかが一目で分かるようになっている。幾つかの代用特性の評価を同一グラフ上で比較することができるので，特定の代用特性の問題の原因を他の代用特性の傾向によって説明できる場合も多い。また，ある代用特性に問題があったとしても，他の代用特性がそれを補っていれば，総合的にはソフトウェアの信頼性に問題がないと評価できる。

また，グラフ中の点線は開発過程で評価したものであり，実線は開発完了後に評価したものである。このように異なった時点で行った評価を同一グラフ上に多重化して表示することによりさらに多くの情報を得ることができる。

5.4 処理効率の評価

本節で取扱う処理効率の評価は，ソフトウェアの処理能力面およびその原因となる項目に関する評価とする。端末での入出力時間などは広義には処理効率に含められるとも考えられるが，ここでは対象外とする。コンピュータ資源の有効利用に関する評価は，直接ソフトウェアの処理効率に影響を与えるので，本節の対象範囲とする。

5.4.1 評価項目

ソフトウェアの処理効率の評価は，単に評価のみに止まるのではなく，必ず改善作業を伴うことになる。そのために，評価作業は，概ねつぎの手順に従って，より根本的な原因へと問題を絞り込みながら進められる。

(1) 利用者から見た処理効率の評価

処理効率の面で第一優先としなければならない評価であり，レスポンスタイム，単位時間の処理件数などが評価対象項目となる。

(2) ソフトウェアの資源利用状況の評価

利用者から見た処理効率に問題が発見された場合にはその改善を図る必要がある。その問題の原因を探るためには，まず，そのソフトウェアの資源利用状況を評価して見るのが良い。資源利用状況評価は，上記の問題と係わりなく，ハードウェア資源のコスト削減などを目的としても必要が発生する。

評価対象項目は，ＣＰＵ使用時間，ファイルアクセス時間などである。

(3) ソフトウェア内部構造の評価

資源利用状況の評価により，問題の原因の絞り込みが行われる。さらにその根本原因を究明して問題の解決を図るためには，プログラム自体の内部構造あるいはデータ構造の評価が必要である。

(4) コンピュータ資源の負荷評価

大規模システムでは，ソフトウェア自体の問題だけでなく，コンピュータ資源の負荷状態が処理効率に大きな影響を与える。従って，コンピュータ資源の計画もソフトウェア開発者の責任下にある場合は，ソフトウェア内部構造の評価と共に，コンピュータ資源の負荷評価を行う必要がある。

評価対象項目は，ＣＰＵ負荷，入出力装置負荷，メモリ使用率などである。

5.4.2 評価作業の手順

評価作業は，前節で述べた各評価項目についてつぎの手順で進められる。評価の結果何らかの対策をとればこの手順が繰り返される。

(1) 評価項目の設定

何のために何を評価するのかを明確にする。目的が不鮮明だと無駄な作業になりがちであり，つぎの環境設定作業も不的確なものとなり，測定のやり直しなどが多発する。

(2) 測定環境の設定

実運用環境を想定して測定環境を設定する。良い環境，平均的な環境，悪い環境の３段階程度の環境が必要である。操作不可能な外乱が入らないように注意する必要がある。そのような外乱の下で測定したデータは再現性がないため評価不可能である。

(3) 測定

マン・マシンインタフェースに係わる項目以外の測定にはツールが必要であり，既存ツールの調査・開発が事前準備として必要となる。バラツキのある評価項目についてはＮ件のデータを取る必要がある。

(4) データ分析・問題の究明・対策の実施

必要によりグラフ，層別，散布図などを使用して問題の把握を行う。基準値などがあれば評価も容易であるが，そのような評価ができない場合には，取得データ間で相対評価を行う。

が，やはり，余り大きな記憶領域を使用するとプログラムの外部記憶装置への退避が発生して処理効率の低下を招く。改善方法としては，テーブルサイズの縮小，データの可変長化，セグメンテーションなどがある。

メモリの使用量は会計情報またはリンケージエディタの出力から把握できる。

(3) ファイル入出力回数

ファイル入出力回数は処理時間に大きな影響を与える。改善方法としては，アクセス方式の変更，データベース構造の変更などがある。

ファイル入出力回数は会計情報から大雑把に知ることができるが，ファイル単位のアクセス回数を知るためにはメーカ提供の特殊なツールが必要である。

(4) 帳表出力枚数

バッチ処理の場合は最も処理時間に大きな影響を与える。帳表の必要性の見直し，印刷情報の集約化によって改善を図ることになるが，必ずしも改善できるとは限らない。

(5) 回線伝送時間

回線伝送時間はオンライン処理のレスポンスタイムに占める比率が大きい。会話数の削減，伝送量の削減によって改善できるが，回線速度を上げる方がてっとり早い。

5.4.5 ソフトウェア構造の分析

(1) ソースプログラムの行単位の実行回数

ソースプログラムの各行の実行回数およびその比率を測定することにより，プログラム中の繰り返しの多い部分を簡単に把握できる。その部分の最適化を図ったり言語を変更することにより改善することができる。

コンパイラ自体の持つ機能で測定することができるが，専用のツールも存在する。

(2) トレース

プログラムの命令の実行順番を調べることにより静的な実行回数からだけでは得られないプログラムの実行特性を知ることができる。

(3) データ構造の分析

ファイルの編成方法，データベースの論理構造の見直しにより処理時間

が大幅に単縮されることが多い。

5.4.6 コンピュータ資源の負荷分析

(1) ＣＰＵ負荷

ＣＰＵ負荷が大きくなるとレスポンスタイムの急激な低下を招く。問題がある場合には，ＣＰＵの上位機種へのレベルアップが必要である。

(2) チャネル負荷

問題がある場合には，チャネルの増設が必要である。

(3) ＤＡＳＤ負荷

特定のＤＡＳＤにアクセスが偏っている場合は，ファイル割り当ての変更が必要である。

(4) ページング回数

メモリ負荷の代用特性として使用できる。ページング回数が多い場合は，実行可能ジョブ数の制限，各ジョブの優先順位の見直しなどで最適化を図ることができる。

(5) 回線負荷

回線負荷はオンライン処理のレスポンスタイムに大きな影響を与える。問題があれば，より高速の回線に変更する必要がある。

5.5 使い易さの評価

5.5.1 使い易さの評価項目

(1) 使い易さとは

使い易さを広義に捉えれば，その評価項目は品質評価項目全般に渡ると考えられる。それは概ねつぎの項目に分けられる。

- 信頼性：そのソフトウェアに与えられた指命を誤りなく遂行する度合
- 処理時間：データを投入してから処理が終るまでの時間
- 機能の充足度：利用者が必要としている機能をどの程度充足しているか
- 携帯性・運用性：ソフトウェアを使う上で，本来のソフトウェアの目的以外でどの程度手間がかかるか
- 操作性：利用者がいかに短かい時間で操作方法を習得できるか，利用者が抵抗なくその操作方法を受け入れられるか，無駄なく効率的に目

的を達成できるかなど

本節では，使い易さを狭義に把え，操作性を中心にまとめた。また，操作性を使い方の学び易さ，親しみ易さ，効率の良さ，誤操作への強さに分けて，それぞれの評価項目について説明した。

(2) 使い方の学び易さ

使い方の学び易さとは，操作方法をいかに短時間で習得でき，覚えたことは忘れにくく，操作時には混乱なく正確に操作できるかなどの操作性の側面である。これらの側面を評価するためのチェック項目としてつぎのものがある。

- マニュアルが不要である
- 利用開始時に操作方法の説明がある
- 習得時間が短い
- 自習機能が組み込まれている
- ＨＥＬＰ機能がある
- 似たような機能が複数存在しない
- 機能の異なるコマンド名が似ていない
- コマンド名は機能を連想できる
- 操作全体を通じて思想が統一されている
- 初心者向けから熟練者向けまで，習熟段階に応じた機能が準備されている

(3) 親しみ易さ

利用者が違和感，不快感，抵抗感を感じずに操作できるかなどの操作性の側面である。この側面を評価するためのチェック項目としてつぎのものがある。

- インタフェースは一般に使用されている常識的な表現である
- メニュー選択方式のインタフェースである
- 図式表現のインタフェースである
- 自然文に近い表現でのインタフェースである
- エラーメッセージが親切である
- 必要な場合はメッセージの詳細が見られる
- 処理の完了がポイント，ポイントで明確に表示される
- 表示形式が統一されている

・人間の思考に沿った操作手順である
・応答時間のバラツキが少ない
・利用者の注意を促すのに音を使用している

(4) 効率の良さ

短時間でいかに多くの目的を達成できるかと言う操作性の側面である。この側面を評価するためのチェック項目としてつぎのものがある。

・応答時間が短い
・人の習性（指の動きなど）が考慮されている
・簡単なメッセージである
・会話数が少ない
・キータッチが少ない
・コマンドなどの省略形が用意されている
・省略値が自動的に設定される
・省略値を利用者が設定できる
・利用者のレベルに応じた操作方法が準備されている
・機能の切り替えが柔軟である

(5) 誤操作への強さ

利用者の誤操作に対してどの程度の考慮を払っているかと言う操作性の側面である。この側面を評価するためのチェック項目としてつぎのものがある。

・単純な操作ミスが起きにくい
・操作ミスなどによって誤動作をしない
・操作ミスによって操作手順が迷路に入らない
・利用者のミスを早く検出する
・軽微な入力ミスの自動修正機能がある
・操作ミスをした場合，前の時点に戻ることができる
・操作ミスの原因を詳細に指摘できる
・データの保護機能がある
・データのバックアップ機能がある

(6) 利用者による評価基準の違い

使い易さはそのソフトウェアの利用対象者によって大きく異なる。例えば，不特定多数の多くの人を対象としたソフトウェアと特定の人によって

ルーチンワーク的に使用されるソフトウェアの使い易さは評価基準は異なる。また，初心者と熟練者によっても評価基準は異なる。

初心者にとっては，使い方の学び易さ，親しみ易さ，および，誤操作の強さが重要な評価ポイントとなり，熟練者にとっては，効率の良さ，誤操作への強さが重要な評価ポイントとなる。また，熟練者を対象としたソフトウェアでも最初は初心者を相手とすることになるので，熟練度に応じたインタフェースを持つべきである。

このような要素から，使い易さの評価項目を一律に決めることは困難であり，ソフトウェアに応じて何を評価項目とするかを選択しなければならない。

5.5.2 使い易さの評価方法

(1) 官能検査

使い易さを評価するためのてっとり早い方法はまず使って見ることである。これを官能検査と呼ぶ。官能検査の効果的なやり方として，木村泉（東京工業大学）はつぎの4つのポイントをあげている。

① その場で書き留めること

大切なのは出会いである。始め何てひどいソフトウェアだと思ったものが，2，3度使っているうちに当たり前に見え出すことはよくある。人の適応力はすばらしい。だからこそ適応し出さないうちに思ったことをその場で書き留めておかなければならない。

② 予備知識を前もって仕込んでおくこと

検査時にそのソフトウェア固有の問題でないバックグランドの知識（例えば計算機の知識）に始めて出会うと，それらの出会いによってソフトウェアそのものとの出会いが薄められかねない。それを避けるためには，バックグランドの知識を持った人を被検者にすることが望ましい。

③ マニュアルはなるべく読まないこと

あまり始めからマニュアルをしっかり読んでしまうと，そこで出会いが終ってしまい，肝腎な時になって印象の鮮明さが失われかねない。マニュアルを読んでいないためにマニュアルに書いてある禁止事項に抵触してファイルを破壊したとしても，それもソフトウェアの品質のうちで

ある。

④　そのソフトウェアを使わずにはいられない仕掛けをする

例えば，ワードプロセッサーの場合は，そのソフトウェア自身を使って記録を取るというふうに，そのソフトウェアを実用的な目的で使用してみることである。

(2)　チェックリストによる評価

使い易さの定量的評価は，チェックリストを使用することによってある程度可能である。そのようなチェックリストの一例として，斉藤修（富士通㈱）の実施結果報告を実施手順に従って報告する。

①　使い易さの要因を洗い出し，大分類，中分類，小分類に分類して，評価項目一覧表（図表５－１２）を作成する

図表５－１２　使い易さの評価項目

大分類	中分類	小分類	評価項目
親しみ易さ	・助けになる ・人間工学 ・	・構文の柔軟性 ・多様な指示方法 ・	・曖昧な入力に対し明確化な要求がある ・
効率の良さ	・高い生産性 ・	・作業時間の短さ ・	・実行時間が短い ・
使い方の学び易さ	・容易な習得 ・	・習得時間が短い ・	・
誤操作への強さ	・システムの保全 ・	・リカバリ ・	・

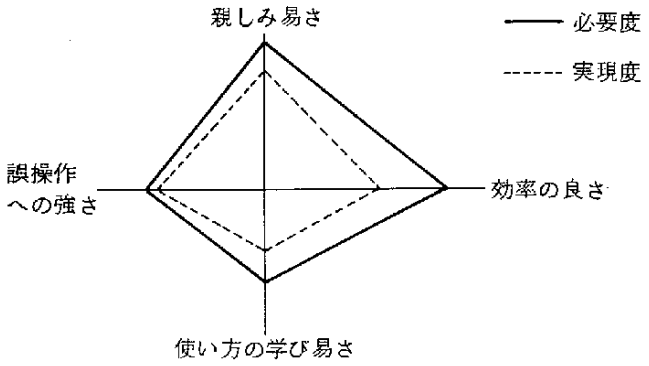
②　各分類段階別に，全体が１００％となるように評価項目の重み付けを行う。その方法は，アンケート調査で多くの人に重み付けを行ってもらいその平均を取る

③　重み付けを考慮した採点表を作成し，そのソフトウェアの使用者に配布して，各評価項目の必要度と実現度を採点してもらう

④　採点結果を集計して，各分類段階別にそれぞれの評価項目の点数を積み上げ計算する。

⑤　採点結果をレーダーチャートでグラフ化（図表５－１３）して，操作性の各側面の評価のバランス，現状と必要性のギャップを評価する。

図表５－１３　使い易さの評価



5.6　保守性の評価

5.6.1　保守性評価の目的

開発されたソフトウェアには必ずバグが潜在しており（一般的には１０Ｋステップ当り数件），運用中にそれが顕在化した場合にはその修理が必要である。また，ＯＳやハードウェアなどの環境が変化した場合には，それに適応させるためのソフトウェアの変更が必要である。さらに，運用を続けていると多くの場合利用者からの新しい要望が発生し，それに対応するためにソフトウェアの機能追加が必要となる。

このようなソフトウェア保守作業工数のソフトウェア開発総工数に占める割合は，一般に５０％を超していると言われており，今回の調査においてもＳＥ・プログラマの５３.８％の工数が保守作業に向けられていると回答されている。従って，ソフトウェア開発の生産性を高めるためには，いかにして保守を行い易いソフトウェアを開発するかが重要な課題となる。

ソフトウェアの開発者と保守者が同一人の場合は保守性の問題はそれほど顕在化しないが，両者が異なる場合には大きな問題となる。長期的に考えれば両者が異なるのは一般的であり，また，近年のようにソフトウェアが巨大

化してくると増々その傾向は強くなる。

従って，開発完了時点でソフトウェアの保守性を評価し，問題があれば部分的な作り直しを行う必要がある。また，保守の観点からドキュメント類に不備があればその修正・追加作成も必要である。

しかし，開発完了時点に保守面からの大きな問題が発見され，それに対応するためにはソフトウェアの全面作り直しが必要となり経費上実施できない場合も考えられる。従って，開発過程で常に保守性を考えて設計・製造をする必要がある。保守性の評価作業にとって，次回のソフトウェア開発作業では保守性を高めるためにどのようなことを考慮して設計・製造すればよいかを策定することも重要な目的の一つである。

5.6.2 保守性評価のポイント

保守性とはいかに少ない手間でかつ短時間で修正，変更，機能追加が行えるか示す尺度であるが，直接そのような特性を表わす尺度は見当らない。従って，保守性を評価するにはその代用特性を評価する必要がある。

一般的につぎのような点が考慮されたソフトウェアが保守し易いと言われており，ソフトウェアの保守性はこのような観点から評価することができる。

(1) 構造化

開発者はユーザ要件の実現のし易さからソフトウェアの構造を設計しがちである。しかし，ソフトウェアのバグは機能上の問題として露見し，また，ユーザ要件の変更は機能上の問題から発生する。

従って，保守者はまずテーマとなった機能を実現している部分を特定する作業から取りかからなければならない。もしその部分が広範囲に渡るならば，保守者はそのソフトウェアを広範囲に渡って調査しなければならず，ソフトウェアはより概略的なユーザ要件からより詳細なユーザ要件へとトップダウンに構造化しておくことが望ましい。

(2) モジュール化

保守を行い易くするには，機能を実現している部分を局所化すべきであることは，ソフトウェアの非常に小さな構成単位（モジュール）についても言うことができる。

あるモジュールの変更が他のモジュールに影響をおよぼすことは度々あり，これが保守作業を面倒にしている。モジュールが自己完結的に一つの

機能を実現していればこのような問題は無くなる。そのようなモジュールであるかどうかを客観的に評価することのできる尺度として，モジュールの出口数，外部とやり取りする情報の数などがある。

(3) データ名称の一元化

機能は常にデータ項目と密接な関係を持っている。従って，同一のデータ項目を取り扱っている部分は何らかの関連を持っている。ソフトウェアの部分によって同一データの名称の付け方が異なると保守が非常に困難となる。反対に，名称が統一されていれば，データ項目を媒介として変更機能の影響範囲を洗いあげることができ，保守の手助けとなる。

(4) ソースプログラムの読み易さ

変更の影響範囲が特定されれば，つぎにプログラムの具体的な変更作業に移ることになるが，そこで問題となるのがソースプログラムの読み易さである。一般的に，つぎのようなことが読み易いプログラムの条件とされている。

- 機能単位に階層的に構造化を行っていること
- GO TO命令が少ないこと
- データ名称はデータの意味を判別できる名称になっていること
- 保守者が理解しにくいと考えられる部分には，保守者へのメッセージとして，コメントが付けられていること

(5) 保守用ドキュメントの充実

開発を目的として作成されたドキュメントは，保守作業にとって必ずしも充分とは言えない。それらのドキュメントの中には，プログラムが完成した後には不要と考えられるものもある。また，それらのドキュメントは，開発過程における度重なる仕様変更で，プログラムとの同一性が保証できない場合も多い。従って，ドキュメントについてつぎのような観点から評価する必要がある。

- 開発過程で作成されたドキュメントのうち，プログラムの保守のために必要であり，将来とも保守しなければならないのはどれであるか
- ドキュメントのプログラムとの同一性が保証されているか
- 開発過程では必要なかったが保守の手助けとなる情報が，保守者へのメッセージとしてドキュメント化されているか
- 流用したプログラムについて，保守可能なレベルのドキュメントがあるか

(6) 再テストの考慮

保守時に発生する大きな問題の一つにテストがある。ある機能の修正のために行ったプログラムの変更が他の機能に影響をおよぼすことは多い。そのため，わずかなプログラムの変更であっても，その変更の信頼性を保証するためには，開発時と等しい程のテストを必要とする。従って，保守時のテストを考え，開発時のテスト環境をそのまま保存しておく必要がある。保存しておくべきテスト環境としてつぎのものがある。

- テスト項目表
- テストデータ，テスト結果
- テストツール
- プログラム中のテスト命令

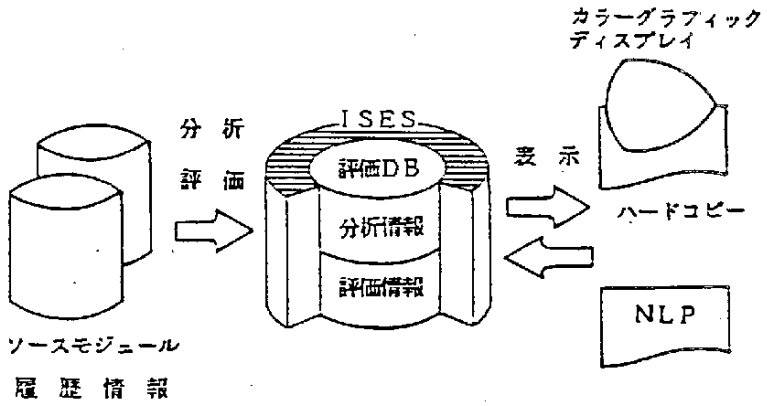
5.6.3 保守性評価のためのツール

保守性の評価を目的としたツールは余り多く見られないが，モジュール化の評価ツールについては幾つかの報告がある。それらのツールの中からＩＳＥＳ（富士通㈱）とＥＳＱＵＴ（㈱東芝）の２つを紹介する。

(1) 総合的ソフトウェア評価システム「ＩＳＥＳ」

ＩＳＥＳは図表５－１４に示すように，評価データベースを中心に各種の分析・評価ツールを統合化したシステムである。評価データベースは分析情報と評価情報から成る。分析情報は各モジュールを分析して得られる複雑度に関する基礎データである。評価情報は利用者の指示する評価基準に従って分析情報を評価した結果である。利用者が評価項目と表示形態を

図表５－１４　ＩＳＥＳの構成



指示すると，結果がカラーグラフィックディスプレイまたは日本語ラインプリンターに出力される。例えば，そのサブシステムであるＭＥＴＳはつぎのような機能を持っている。

① つぎの３種類の抽出条件を組み合わせて評価対象モジュール群を選択できる

モジュールの属性（命令規則，プログラム種別など），プログラムの階層レベル，世代。

② つぎのような評価指標で評価できる

プログラムとデータ間のアクセス関係，プログラム相互の呼び出し関係，ソフトウェア階層構造，呼び出し・アクセス関係のクラスタリング，呼び出し関係の正常性チェック。

例えば，プログラムとデータ間のアクセス関係を指標とした場合は，横軸にプログラムモジュール番号，縦軸にデータモジュール番号をとり，その間のアクセス関係の存在を点で表わした図が作成される。その図で，プログラムモジュールおよびデータモジュールの並びを調整してもプロットした点が分散している場合は保守性に問題があるといえる。

(2) ソフトウェア保守性評価ツール「ＥＳＱＵＴ」

ＥＳＱＵＴはモジュールの諸特性を定量的に分析して保守のし易さを評価するツールである。本ツールは，保守性の良し悪しを判断するだけでなく，開発者に改善方策をメッセージとして伝える機能を持っている。

本ツールでは，保守のし易さをつぎの４つのモデルで表現できると考え，各モデルで有効と考えられる計測尺度を設定している。

① モジュール機能サイズモデル

モジュールの持つ機能数を定量化することにより，機能の過度な集中を発見する。計測尺度として，出口数，変数ブロック数，関数群の数を使用する。

② プログラム構造モデル

モジュール間の関連，グローバルデータの使用状況を定量化することにより，機能の過度な集中や混合を発見する。計測尺度として，外部変数の使用量，平均出口数，モジュール呼び出し数を使用する。

③ 読み易さモデル

見た時の読み易さ，コメント文による説明の量，位置の適切さがプロ

グラムの理解を助けると考え，読み易さとして定量化する。計測尺度として，コメント率，コメント位置，変数有効範囲，ステップ数，内部変数密度，外部変数密度を使用する。

④ プログラム複雑さモデル

モジュールの持つ複雑さが保守時の修正作業量に影響を与えると考えられるので，複雑さを定量化して評価する。計測尺度として，実行文数，Vocabulary Size, Program Length, Program Volume, Cyclomatic Number, 情報量を使用する。

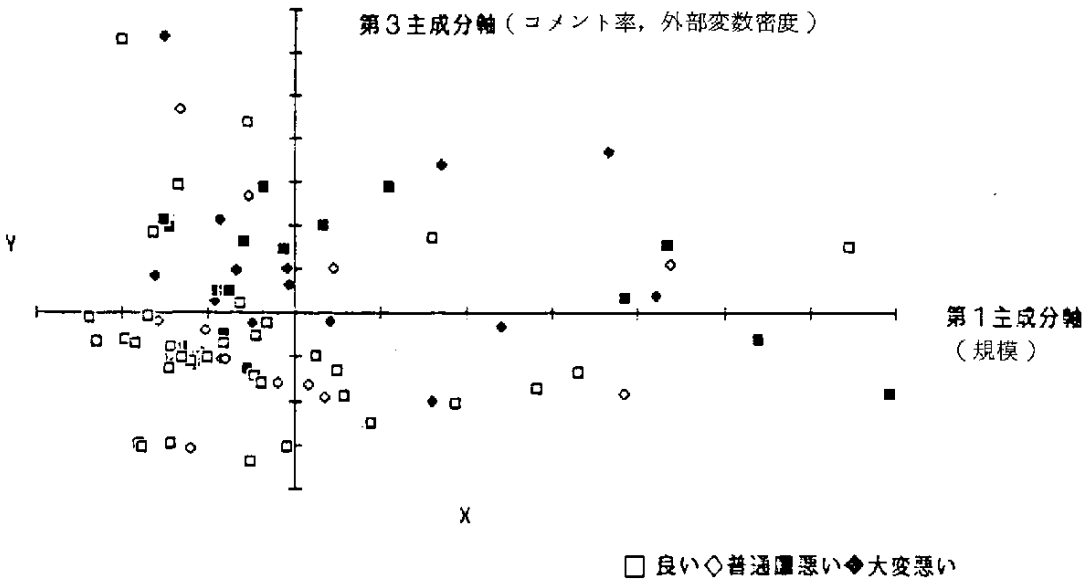
これらの尺度が保守のし易さに対しどのように影響を持たらすかが，具体的データに基づいてつぎのように報告されている。

報告では，主成分分析により上記尺度中の相関の高い尺度同志を整理し，つぎの合成尺度を作り出し，その合成尺度と実際の保守のし易さの関係について分析している。

- 第1主成分：ステップ数，複雑さの尺度から構成された規模の合成尺度
- 第2主成分：出口数，変数ブロック数などから構成された合成尺度
- 第3主成分：コメント率，コメント位置，外部変数密度から構成された合成尺度

保守時に出現したモジュールの不具合件数をデータとし，モジュールの保守のし易さを4グレード（良い，普通，悪い，大変悪い）に分類して第1－3主成分平面上にプロットしたものが図表5－15である。この図表よりつぎのことが言える。

図表５－１５　ＥＳＱＵＴ保守性分析事例



①　規模を表わす合成尺度が大きい所では保守性が悪い可能性が高い

②　コメント率，外部変数密度が小さい所では保守性が良い傾向が高い

5.7　開発活動自体の評価

以上整理してきたことは完成されたソフトウェアの評価が中心であったが，開発活動を振り返り，開発のやり方を評価しておくことも必要である。その結果はつぎの開発に役立てることができるし，他組織の開発活動の参考にすることもできる。そのような試みを行っている事例として㈱富士通の「品質管理パラダイム」について紹介する。

(1)　品質管理パラダイムとは

品質管理パラダイムとは，日頃，実作業の中で工夫し，実践している「品質管理」の活動事例を一定の形式で整理して形あるものとして蓄積し，再利用・追体験可能なようにしたものである。

品質管理パラダイムは，ＱＣストーリーに似たＱＣ手順に従ってドキュメント化されるので，それを作成することによって，自分達の行ってきた開発活動を反省することができる。

品質管理パラダイムは，管理のサークル（ＰＤＣＡサークル）の各フェイズに対応してつぎの３種類のパラダイムに分けられる。

① プロジェクトパラダイム

一つのソフトウェア開発プロジェクトの管理方式の全体像を記述する。すなわち，計画から開発・運用にいたる全工程に渡って，作業内容・生産物・生産技術・ツール・管理方法・組織形態などの諸側面を記述する。

② チェックパラダイム

各工程に対して固有の作業方法が必要であるのと同様に，作業監視（チェック）のやり方も工程に即したものがあるはずである。そのような作業監視の事例を記述するのが「チェックパラダイム」である。

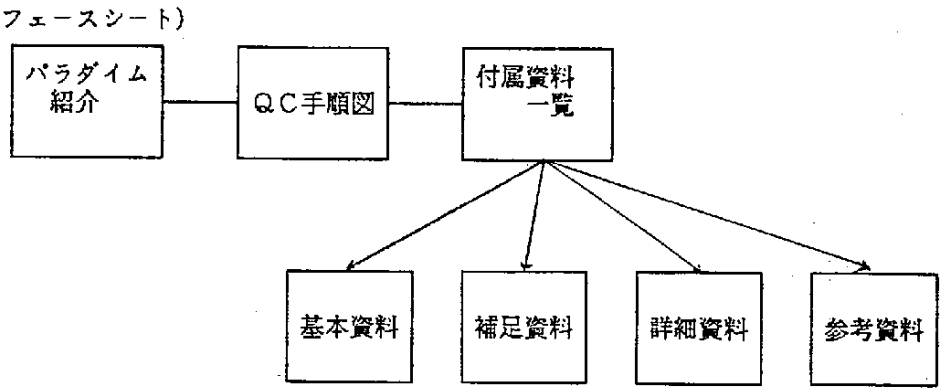
③ アクションパラダイム

作業監視の結果判明した問題に対して，根本的な問題分析と解決を行った事例を記述するのが「アクションパラダイム」である。

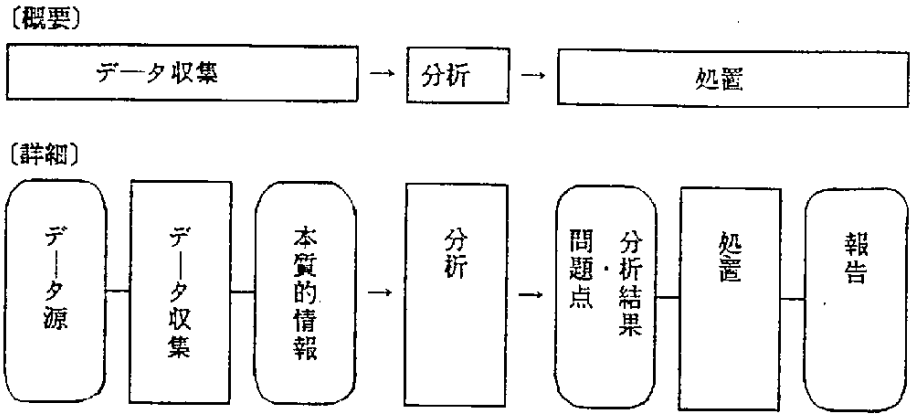
⑵ 品質管理パラダイムの形式

品質管理パラダイムは，図表５－１６で示す文書体系をとっている。ポイントはＱＣ手順図にあり，図表５－１７および図表５－１８に示すストーリーに従ってフォーマッティングされた用紙に記入される。

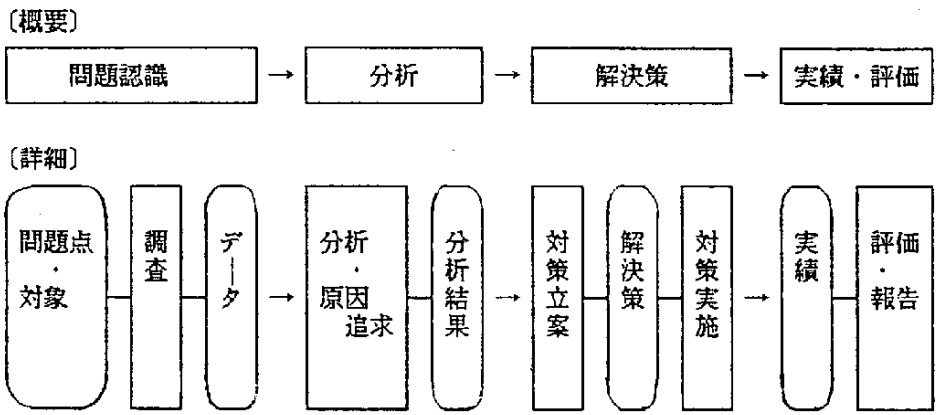
図表５－１６　品質管理パラダイムの文書体系



図表５－１７　「チェックパラダイム」のＱＣ手順



図表５－１８　「アクションパラダイム」のＱＣ手順



⑶　ＱＣ手順図の作成方法

品質管理パラダイムの作成は，基本的には第三者が行うのが望ましい。それは，当事者が作成すると当事者が当り前と思っていることが漏れてしまいがちであるが，第三者が作成すれば当事者が気づかなかった事例の長所・短所に気づくことが多いからである。効果的な品質管理パラダイムの作成方法にはつぎの３つがある。

①　第三者が事例の当事者からヒアリングを行いながらＱＣ手順図を作成する

②　複数の聞き手で当事者にヒアリングしながら模造紙上にポストイットカードを使用してＱＣ手順図を作成する

③ 当事者が原案を作成し，複数の人でヒアリングを行う

参 考 文 献

1） 中川 徹「ソフトウェア開発履歴分析（ＡＳＤＧＥＭ)」(財）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

2） 宮崎幸生「Ｕ管理図の考え方による品質データの分析結果」以下同上

3） 山田淳他「ソフトウェア保守性品質評価ツール」（財）日本科学技術連盟，第５回ソフトウェア生産における品質管理シンポジウム，１９８５.９.１８

4） 木村 泉「ソフトウェアの使いやすさの評価」以下同上

5） 中川 徹「ソフトウェア品質管理パラダイム」以下同上

6） 菅野文友「ソフトウェアエンジニアリング」

7） 青山幹雄他「総合的ソフトウェア評価システム（ＩＳＥＳ）」（社）情報処理学会第２９回全国大会講演論文集，１９８４.９.１１

8） 高橋宗雄他「ソフトウェア信頼性影響要因」以下同上

9） 斉藤 修「エンドユーザ向け製品の"使い易さ"の定量的評価」ＥＮＧＩＮＥＥＲＳ，１９８３.１０.３１

10） Ｍ．Ｌ．シュナイダ「ユーザ・インタフェースの人間工学的設計とは何か」日経コンピュータ，１９８３.１０.３１

11） Ｂ．Ｗ．ベーム「SOFTWARE ENGINEERING ECONOMICS」Prentice-Hall，Inc，１９８１

12） 「プログラム権法の立法」「ソフトウェア品質評価の制度化」日経コンピュータ，１９８４.２.６

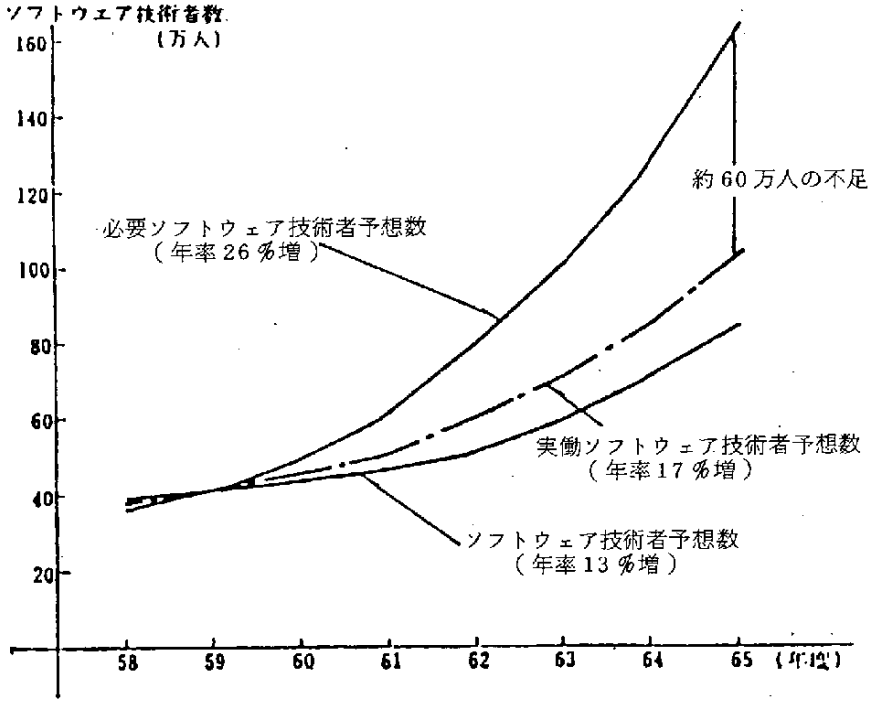
6. ソフトウェアの今後の課題と展望

6. ソフトウェアの今後の課題と展望

ソフトウェア分野においては，需給ギャップに対する対策が急務となっている。BoehmやMartinの報告によると，労働者数と生産性はそれぞれ年間４％しか伸びないのに対して，ソフトウェア需要は年間１２％も増加すると言う。

また，我が国においても，ソフトウェア需要の年間伸び率２６％，生産性の年間伸び率４％，ソフトウェア技術者数の年間伸び率１３％であり，昭和６５年度においては約６０万人のソフトウェア技術者が不足すると予測されている。（図表６－１）

図表６－１　我が国におけるソフトウェア需給ギャップの予測



このような状況のなかで，現在の最緊急課題はソフトウェアの生産性の向上であると言える。ここで言う生産性の向上には単に開発効率の向上という意味だけでなく，出来上がった製品の品質向上，保守の効率向上の意味も含めている。

この章では，上記の観点からソフトウェアの今後の課題と展望について検討したい。

6.1 今後の課題

現在のソフトウェアを取り巻いている環境を，ソフトウェア要員，技術・ツールおよびソフトウェア品質・評価の3つの側面に分けて，以下その課題について述べる。

6.1.1 ソフトウェア要員に関する課題

(1) 要員面

冒頭で述べたように，ソフトウェア需要に対するソフトウェア要員の不足は，不可避的なものとして深刻に捉えなくてはならない。いかにしてソフトウェア要員を確保していくかが緊急の課題である。

ソフトウェア要員の年齢限界説が，よく言われる。確かにプログラマなどには若年層が多い傾向にあり仕事の内容も体力がものをいう部分がある。しかし，システムアナリストなど高度な技術力や判断力を必要とする職員に耐え得る人材としては，むしろ年配者の方が向いている場合もある。このような既存技術者の適材適所の配置を図りながら，必要に応じて外注稼働を活用していくことが迫られてくるだろう。

最近，各分野への女性の進出が取り沙汰されているが，ソフトウェアの分野にも女性プログラマが徐々に増加している。一方，現在のソフトウェア関連企業は大都市近郊に集中しており，一部の小規模ソフトウェア作成業者を除いて都市部以外の労働力は余り活用されていない。活用できる分野は限られるかも知れないが，今後増加が予想される，在宅勤務やサテライトオフィス勤務といった新しい勤務形態の実施を含め，これらニュー・パワーの活用を図っていくのが得策と考えられる。

(2) 組織面

ソフトウェアの設計から運用・保守にいたる一連の工程においては多数のプロジェクト・チームが存在するが，各チーム内外のコミュニケーションが滞りなく行われるような仕組みが必要である。

また一般に，ソフトウェア開発におけるプロジェクトチームの必要人数は，ライフサイクルによって異なる。このため，要員の異動が容易に行えるよう組織的な配慮がなされていることが必要である。とくに，従来とかく軽視されてきた運用・保守サイドを含め，質と量のバランスがとれた要員配置を行うことが大切である。

今後組織面から見て注意しなければならないのは，高齢化問題である。

ソフトウェア生産支援ツールが整い，標準化も浸透して来れば，職人芸で売って来たプログラマは，依然としてバックログに苦しんでいる現状のなかですぐさまそうなるとも思えないが，相対的な存在価値が下がる可能性がある。プログラマだけでなく，他のソフトウェア要員についても，5年後，10年後の自分自身の存在価値については，一抹の不安を抱かざるを得ないのではないだろうか。

高齢化問題では，高齢化した要員に対し，会社が年齢に見合った役職・ポストを用意できないことが最大の問題であり，専門職制度など有効な対策を模索する必要があろう。

(3) 教育・訓練面

ソフトウェア開発については技術革新が激しく，これに供わる技術者は絶えず新しい技術に対応した知識を要求される。他方，実際に直面している作業に追われる余り，直接関連する知識以外は吸収しにくく，また体系的な教育がなされない傾向がある。

このような状況の中で，技術者が新鮮な知識をタイミング良く修得できるようにするためには，作業計画の中に訓練計画を盛り込み，定期的なスキルアップを図る必要がある。また，技術者をオペレータ，プログラマ，プロジェクト・リーダそして維持管理要員の順に一定期間ずつ経験させるという，ジョブ・ローテーションにより知識と経験の巾を広げることも効果的である。

また，要員個々の技能管理を徹底し，適材適所の要員配置を行うことはもちろん，各自の技能に応じたキャリアパスを設定し長期的な展望に基づく要員の養成に当たるべきである。

6.1.2 技術・ツールに関する課題

技術・ツールについては，設計，製造，保守の各工程および生産設備，生産支援ツールの6つの分野に分類して，以下に述べる。

(1) 設計技術

設計における問題点としては，ユーザ要求の定義・分析が十分でないことによる稼働の増大，設計の再利用手法が確立されていないことによる非効率性といった点が指摘されている。

要求定義・分析のためのツールとしては，ミシガン大学で開発されたISDOS，TRW で開発されたSREMが有名であるが，何れも未完成度は低く実用には遠いのが現状である。

また，設計の再利用についても，ソフトウェアの設計内容を蓄積し検索する技術が未熟であるために，コードを再利用するよりもはるかに難しく現状ではこれといった有効な対策はない。

(2) 製造技術

製造面の問題点として，既製品の流用がなかなかうまくいかないこと，製造を支援するツールが不十分なことが指摘されている。

既製品の流用については，プログラムの部品化が有効であると言われているが，対象を極く一部にしぼった上での部品化（バッチの事務処理プログラムなど）を除いては実現されている例がない。また，言語自体についても従来からCOBOLなどが広く使用されているが，言語仕様が必ずしも一致していないため，流用できる範囲が狭くなっているのが実情である。

対策としては，記述性に優れ，部品化も指向したAdaなどの新しい高級言語による効率的なソフトウェア開発が期待される。

また，特定業務のアプリケーションプログラムを対象にして，従来の手続き型言語の枠を超える非手続き型言語の開発も進められており，その成果が期待されるところである。

(3) 保守技術

保守コストがソフトウェア・ライフサイクル・コストの中で大きな割合をしめていることが報告されており，保守コストの低減，保守作業の効率化などが大きな課題となっている。

現行の保守について，トラブル発生時に解析情報が不十分で原因不明になる場合が多い，プログラム修正時のディグレードチェックが行いにくい，遠隔地でのトラブルの場合に迅速・省力化の点で改善が必要であるなどの問題が指摘されている。

対策としては，故障情報の自動収集や解析ツールの改善，ディグレードチェックツールの開発，遠隔保守方式の導入，保守情報データベースの構築などが進められている。根本的には，設計時点から保守を意識した開発を行うとともに，高級言語の実用化などにより保守工程においても手の掛からないプログラムが作られるようにならなければ解決しないと思われる。

＜保守情報データベースの例＞

①ドキュメント作成検索データベース

②遠隔保守情報データベース

③バグ情報データベース

④保守支援ツールデータベース

⑤更新履歴情報データベース

⑥システム情報管理データベース

(4) 生産管理面

プロジェクトの計画策定や進捗，稼動，品質の管理については，従来個々のプロジェクト・リーダの経験や勘に頼ることが多かった。しかしソフトウェアの規模がこれだけ増加し，またその内容も複雑化してくると，人手ではとうてい精度の高い見積りは行えなくなってきた。

そこでソフトウェアの分野でもこれら生産管理に関して，工学の基礎であるメトリックス技術を確立し，各管理項目を客観的に定量化する必要性が指摘されている。当面は製造規模，品質情報，稼動などの基礎データを蓄積し，それらを分析するシステムを開発していくとともに，開発しようとするソフトウェアが必要とするコストが把握出来るようなシステムの出現が期待される。究極的には工程管理，コスト管理，品質管理，製品管理，要員管理，組織運営管理などが一元的に行えるような管理システムの構築が望まれる。

(5) 生産設備面

生産設備としては，以前は実（ターゲット）マシンでのソフトウェア開発が普通であったが，現在ではターゲット・マシンとは別の開発用マシンにより効率的に開発を行うことが主流になってきた。

今のところ開発要員の自由度を高めるリモートデバッグ技術，デバッグ・マシンを有効に利用するためのVM（virtual machine）技術などが導入されているが，さらにソフトウェア開発の効率向上を支援するためには以下のようなことが必要と考えられる。

①WS（ワークステーション）からの会話形態による設計，製造，テスト支援

②各種支援ツールの充実

(6) 生産支援ツール

ソフトウェアの生産性向上を推進する上で，重要な役割を果たすのが各種の生産支援ツールである。

現在でも既に色々なツールが開発されているが，単発的に開発されるケースが多く，そのため使用効率が悪くなっている。一連の生産過程において総合的に使用できるよう徐々に統合化を進めて行かなければならない。

6.1.3 ソフトウェア品質・評価に関する課題

ソフトウェア生産性向上に取り組む課程で忘れてはならないのが，品質管理である。

(1) 品質の造り込みと評価

とかく品質の良否は製造工程だけの問題として捉えられがちであり，他の工程では余り強く意識されないことが多いのが現状である。また，製造工程においても納期に間に合わせることに終始し，ともすれば品質に対する配慮が失われてしまうこともある。

ソフトウェアのようにオーダメイドのものについてはむしろ計画・設計段階の方が品質にとって重要な意味を持っている。各工程に携わる要員1人1人が品質に関する意識を持ち続けるよう，啓蒙していくことが必要である。

ソフトウェアの品質が各工程において造り込まれるものである点はハードウェアや他の工業製品と同じである。したがって，各工程の中で品質の造り込み方を作業標準として確立しておき，次工程へのバグの混入を極力防止する必要がある。

具体的には，開発開始当初に品質管理目標値を設定し，各工程の中途・終了段階で品質についてのレビューを行ない目標値を満足したかをチェックし，次工程への着手可否の判断を徹底しなくてはならない。

また，ソフトウェアは完成した製品を目で見ることができない。そのため，その品質がどうなっているのか把握しにくい難点がある。この点，出来上がった製品の検査をどのように行うのか，またそのときの評価尺度をどのように決めどう評価するのかが，非常に難しい。

以下に，試験工程で用いられる評価尺度を示す。

① 試験密度

プログラム規模当たりの試験項目数。

② バグ検出密度

プログラム規模当たりのバグ検出数。

③ バグ収束率

予測バグ件数のうち実際に発生したバグ件数の比率。

(2) ディグレードの防止

運用・保守段階に入って発生した故障の修理や機能の拡張などによってプログラムの修正を行った際，該当箇所以外の部分へのディグレードを引き起こすことがある。この影響範囲の見極めには，高い技術力を必要としており，ほとんど人手に頼っているのが実情であり，信頼性に問題がある。

このディグレードが起こらないようチェックするためのシステムを構築する必要がある。ただし，このとき単にディグレードチェックのみを目的としたものではなく，開発段階におけるテスト支援システムや保守段階におけるプログラム変更管理支援システムと一体のシステムづくりを行うべきである。

6.2 今後の展望

ソフトウェアに関して，6.1に述べた課題を解決する技術，方策などについて，その展望を以下に述べる。

(1) 標準化

ソフトウェアの生産性を向上させる上でその前提条件となるのがこの標準化である。もう少し詳しく言えば，標準化はソフトウェアの複雑化，開発規模の増大，要員の質的・量的不足といった最近の状況の中で，品質の高いソフトウェアを効率的に大量生産するための方策である。

標準化のねらいは，誰が作っても同様のものができるようにすることによって，組み合わせが可能な製（部）品を効率生産することにあると言える。

そのためには，従来ソフトウェアの開発において巾をきかせていた個人技による作業ではなく，共同作業としてのソフトウェア生産を重視することになる。具体的には，ドキュメント記述方法や用語，プロジェクト・チーム間のコミニュケーションの方法，各工程における作業方法などの標準化を図り生産性の向上に資することが大切である。

また，完成した製（部）品についても，それがのちに流用・改造により，有効に利用できるように，という観点から，今後は使用言語やインタフェースの標準化がより積極的に推進されよう。

(2) 再利用技術

＜方法論やツールの改良による生産性の向上には限界がある。より大きな改善をするためには，いつも無から作っていてはだめであり，プログラムが知識であることを最大限に活用し既存のものを利用すること，すなわち再利用が決め手となる。第5回ソフトウェア工学国際会議　J．マンソン＞このような考え方の下に，ソフトウェアの再利用による生産性向上の気運が高まって来た。

今後開発するソフトウェアについては，部品化を進め再利用する方向にある。最終的な目標は，ソフトウェアの仕様を再利用してそこからプログラムコードを自動的に生成することであるが，極く限られた一部の応用分野でこういったツールの実用化が試みられている程で，全般的にはまだまだ仕様の再利用が進んでいるとは言えず，現状では未だ研究段階の技術である。

一方，プログラムコードレベルでの部品化・再利用の試みも盛んであり，バッチの事務処理など処理が規格化可能な分野では，例えば“入力チェック”，“照合更新”，“ソート”，“報告書作成”などの処理をある程度汎用的に標準パターンとして作っておくことで，実用的なレベルに達している。また，より広い分野での再利用を図るために，Adaをはじめとする新しい言語やツール（Adaのパッケージ機能，Smalltalkのクラスの概念）によるプログラムコードレベルの部品化・再利用の試みも始められている。

今後，再利用技術を確固たるものとするには，新しい言語によるアプローチ，人工知能（ＡＩ）技術の実用化を推進する必要があろう。

(3) ソフトウェア・パッケージの活用

ソフトウェアとハードウェアの価格分離（アンバンドリング）により，ソフトウェアが独立した商品としての価値を得た。さらに，ソフトウェア開発費用が嵩むこと，開発に掛けるソフトウェア要員の確保難などから，自前で新規開発するよりも汎用化された既製品を購入するほうが経済的なケースがでてきた。こうして，目的別の既製品であるソフトウェア・パッケージが1つの商品として流通するようになってきた。米国では既にこのソフトウェア・パッケージの市場が急激な拡大を遂げている。

ソフトウェア・パッケージの種類としては，制御プログラムや言語処理プログラム，データベース管理プログラムなどから業種別アプリケーション・プログラムに至るまで，多岐に渡っている。

ソフトウェア・パッケージの活用については，（２）で述べた部品化同様，ソフトウェアの有効利用の観点から積極的に推進されるべきものである。そのためには，ソフトウェア・パッケージの標準化・汎用化や，その市場流通を促進するソフトウェア流通システムの構築などが必要であろう。

また日本では昭和５４年に汎用ソフトウェア開発準備金制度が設けられ，流通を目的としたソフトウェア・パッケージの開発が奨励されているが，こういった政策的な援助も当面必要であろう。

(4) 移植技術

コンピュータの普及に伴い，ソフトウェアに対する需要が拡大し，既存のソフトウェア資産をドキュメント，または，プログラム・レベルで再利用しようとする動きが活発になってきている。

以前は機種更改の際に，旧システムで蓄積したデータやプログラムを新しいシステムで使用できるようにするために移植が行われたが，現在では他の機種のプログラムを互換性のない別の機種にのせかえて走行させる目的での移植も行われるようになってきた。このようなソフトウェアの生産性をたかめる再利用技術については，特に異機種間でのプログラム再利用を可能にする移植技術が有効性・適用領域の面から注目されている。

しかし，使用言語が違ったり，同じ言語でも言語仕様が異なったりしているため，完全な自動変換を行うコンバータの開発には経済性および汎用性の面で限界がある。

このため，主としてコンパイラにおいて汎用中間言語方式の採用，機種依存機能の局所化，また，ハードウェア機種仕様を記述しそこからオブジェクトコード分類表や処理ルーチンを自動生成する方式が試みられ再利用促進が図られている。

今後は設計，製造段階において，作業標準の整備や仕様言語，インタフェースの統一を図っていくことになろう。

(5) ラピッドプロトタイピング

ソフトウェアの開発には，従来試作品を作るという過程がなかった。しかし，開発の初期の段階ではユーザの要求が明確でなく，出来上がった後にな

って変更を余儀なくされることも多い。

これを解決する方法として，ラピッド・プロトタイピングがある。実用化の例としては，カリフォルニア大学アーバイン校がＲＰＬ（Rapid Prototyping Language）と呼ぶプロトタイピング用の言語（文法はＡｄａに類似）を作り，試行中である。また，㈱富士通研究所では電子交換機用ソフトウェアのプロトタイピングツールを試作している（言語はＡＩＦＬＳ）。要求分析・定義技術の飛躍的発展が期待できない現状では，ラピッドプロトタイピング技術の方が適用の可能性が高いと言えよう。

(6) 新高級言語

① Ａｄａ

Ａｄａは考え方としては特に「革命的」と言えるようなものはなく，従来の手続き型言語に近いものであるが，

㋑ パッケージ化機能（ジェネリックパッケージ）

㋺ 並列処理，リアルタイム処理機能（ランデブ）

㋩ ＡＰＳＥと呼ばれる充実した支援環境もセットで開発

㋥ コンパイラの検定制度による標準化

㋭ 仮想ＯＳインタフェースによるポータビリティ強化

などの実現を図っており，現状で実用化されている最新技術の集大成という意味で有用であろう。

② ＬＩＳＰ，ＰＲＯＬＯＧ

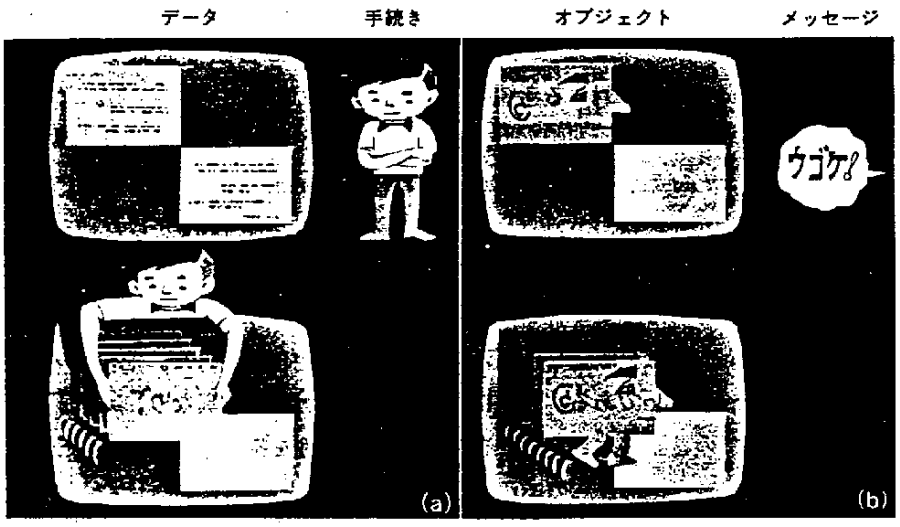
従来の言語がプログラムを手続きの集まりとして表現するのに対して，ＬＩＳＰに代表される関数型言語やＰＲＯＬＯＧに代表される述語論理型言語では，プログラムを処理内容を記した関数とそれに与える引数であるリストの集合，あるいは処理の仕様を記述した論理式の集合として表現する。これらは共に記号処理に適しており，かつ必要なデータがすべてそろった時点で実行がなされるというデータフロー型になじむ言語であるため，今後の人工知能や知識処理に適した言語として注目されており，その結果が期待される。

③ オブジェクト指向言語

これも従来の手続き型とは全く異なる考え方をとっており，データとそのデータに対する操作とをひとまとめにしたオブジェクトという概念でプログラムを表現するものであり（図表６－２），代表的なものにXEROX

社のSmalltalkがある。処理速度がまだ遅いため当面はプロトタイピングが主な用途となろうが，今後汎用的な言語となってくることも予想され，クラスの概念などによりプログラムの部品化技術向上に貢献するものと期待される。

図表６－２　手続き指向とオブジェクト指向の違い



手続き指向とオブジェクト指向の違い。ディスプレイに表示したウインドウの位置を動かす場合を例にとった。手続き指向(a)では，ウインドウ（データ）とその移動（手続き）が分離されている。オブジェクト指向(b)は，この両者を統一したもの（オブジェクト）としてウインドウをとらえる。

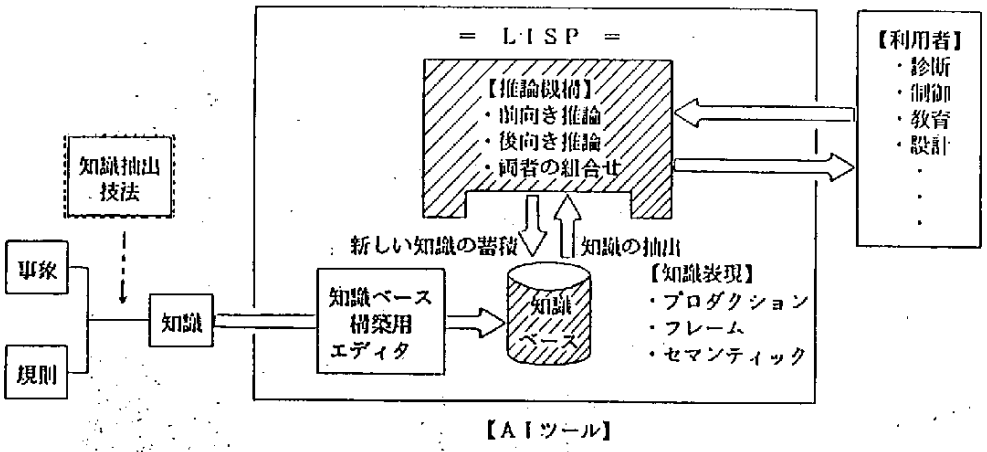
(7)　人工知能（ＡＩ）

＜人工知能（Artificial Intelligence）とは，知的な計算機システムの設計に関係した計算機科学の一分野である。知的な計算機システムとは，我々が知的であると感じる人間の行動（言語理解，学習，推論，問題解決など）を代行するシステムである。Edward. A. Feigenbaum ＞という定義にもあるように，ＡＩの技術とは知識を蓄える知識ベースと知識の処理を行う知識処理とを組み合わせ（図表６－３），人間の知的活動を支援しようとするものである。

ＬＩＳＰやＰＲＯＬＯＧを用い，ハードウェアとしては非ノイマン型のデータフローマシンを使用することによって，実用的なシステムを作ろうという研究が活発に行われている（㈶新世代コンピュータ技術開発機構，マサチューセッツ工科大学など）。

ＡＩの応用分野として，エキスパートシステム，ソフトウェアＣＡＤなどソフトウェアの生産性を飛躍的に向上することのできる領域が考えられており，この技術に期待するところは大きいが，人間の「知識」の抽出方法，新

図表６－３　人工知能応用システム



しい知識の発見方法といった難問がひかえており，今後より一層研究・実用化に力を入れる必要がある。

(8)　ソフトウェアＣＡＤ／ＣＡＭ

装置設計などの分野でＣＡＤ／ＣＡＭ技術を確立しようという試みが活発になってきている。その状況は以下のとおりであるが，最終的には人工知能（ＡＩ）の成果に待つところが大きい。

①　ソフトウェアＣＡＤの目標

㋑ 図形入力によるソフトウェア設計・製造（日本での主流）

- グラフィック表示機能をプログラム用ワークステーション，および，システム設計者用ワークステーションに標準装備し，人間のパターン認識能力の活用を図る
- プログラム設計書やシステム設計書の記述方法を従来の方式から大幅に改良したグラフィックチャートにする
- 単にドキュメンテーションの簡易化，正確化を行うだけでなく，設計図から直接ソフトウェアを作り出すＣＡＭへの方向を目指す

(b)自然言語によるソフトウェア設計・製造（欧米での主流）

② 技術レベルおよび現状

レベル1（現状）

プログラム設計に関するＣＡＤ

レベル2

システム設計に関するＣＡＤ

レベル3

人工知能，知識工学の応用によるソフトウェアオートメーション

③ 具体的なシステムとして実現される順序

ステップ1

従来の設計作業を紙の上で行うのではなくグラフィック端末上で実行可とし，設計情報を収集して後の工程に役立てる。

ステップ2

非手続き的な仕様入力によりプログラムを作成する。

ステップ3

プログラミングの知識，システム化対象領域の業務知識を知識ベース化し，必要最小限のシステム要件の入力によりプログラムを作成する。この際，蓄積された知識に基づいてアルゴリズムの発見も行う。このステップの実現にはソフトウェア製品における部品の確立が必要である。このステップはとりも直さずＡＩ実用化の段階である。

（システム化対象領域を極く狭い範囲に絞れば，案外早期に実現されることも考えられる）

(9) ＰＷＢ（Programmer's Work Bench）

従来，設計や製造で用いるツール類は，個々別々に開発されており，これら断片的なツールの導入には限界があった。というのは，ツール類が仕事の流れに即してサポートされたものではなく，ツール毎にその使用法を修得しなくてはならなかったからである。

そこでツールを統合化し，それをＴＳＳ端末やパソコンで利用できるようにしたのがＷＢ（Work Bench）であり，さらにプログラマ向けにしたのがＰＷＢである。

ＰＷＢの構成要素としては，ＯＳ，ファイルシステム，開発支援ツール群，ユーザインタフェースがある。なかでもユーザに直接見える，ユーザインタフェース部については特に操作性がよいよう，念入りな設計を必要とする。

(10) ソフトウェア・テスト

ソフトウェアの品質を高めるには十分なテストが必要であるが，人手ではそのためのコストもまた大きなものとなる。

現状では，テスト対象のプログラムを実行させることなくそのプログラム内の制御やデータの流れを解析する静的解析，プログラムを実際に動作させその結果をチェックする動的解析の2段階で効率良くテストを行う方法が主流である。また，テストデータの抽出，テスト十分性のチェックなどのために，分岐箇所に着目したテストデータ生成ツール，パス網羅率測定ツールなどが開発されている。

さらに，どの程度テストをしたらどの位の品質が期待できるかという予測技術についてもソフトウェアメトリックス技術の一つとして開発されつつある。

ソフトウェアテストの問題を根本的に解決する方法として，プログラムの正しさを論理的に証明しようとする試み（Program verification）の研究も行われてきてはいるが，現時点では行き詰っており実用化の見通しは暗い。

今後はテスト支援ツールの高度化（テスト中にバグを検出した場合，自動的にその部分のソースを表示し，エディットモードになるなど）を進めることが必要である。

(11) 開発組織

ソフトウェアを開発するプロジェクト・チームを運営していくにあたって，留意して置くべきこととして以下のようなものがあげられる。

① プロジェクト・チームの要員数が多いときは，チームをさらに細区分し，その単位毎にチーフを割りつけ責任を持たせる

② プロジェクト・リーダにはできるだけプログラムを担当させないようにし，管理作業に専念させる

③ 他のプロジェクト・チームとの意志疎通が，十分に図れるような体制を組む

④ 各要員の担務の分界点を明確にするとともに，不在時の代行者，対処方法を決めておく

⑤ 品質管理を行うグループを，開発チームから独立させる

⑥ 会議などにより，定期的に進捗状況を報告させる仕組みにしておく

⑦ 必要に応じて，他のメンバーが開発に専念できるよう，プログラミング

以外の事務作業や成果物の整備などを行うライブラリアンを配置する

⑿ 品質管理（ＱＣ）活動

ソフトウェアの品質は，開発から保守までの各工程において一貫して造り込まれるものであり，ソフトウェア要員全体の意識の高さに比例する，と言っても過言ではない。したがって，日々の作業において，いかに品質について意識を持たせるかが重要である。要は，担当者１人１人が意識を高めないと品質は向上しないし，生産性も向上しない，ということである。このための有効手段として，品質管理（ＱＣ）活動がある。

品質管理活動は本来工業製品の製造に関して，実施されて来たものであるが，標準化によるソフトウェア生産を行うようになるにつれて，ソフトウェア分野においても上記の見地からますます適用が進むものと思われる。

以下に，その目的を示す。

① 個人技術力の向上，動機付け

② チーム技術力の向上，円滑なコミニュケーションによるチームワーク

③ 品質意識，コスト意識，問題意識，改善意識の高揚による職場の活性化

④ 張り合いのある明るい職場づくり

⒀ 要員管理

ソフトウェア生産プロジェクト・チームの構成員数はソフトウェア・ライフサイクルによって，大きく変化する。

現状では，必要とする時期に，必要とする要員を質・量ともに集めることは，かなり難しい。そのとき，幸いにして確保し得た人材をいかに有効に活用できるかで，成否がきまる。このため，各技術者の経験，スキルなどについて木目の細かい管理を行い，すきのない要員配置を実施することが必要である。

⒁ 生産支援ツール

ソフトウェア生産を支援するツールとして，以下に示すような色々なものが考えられている。既に開発され効果を上げているものもあるが，今後はこれらを統合化し使いやすいものにしていく方向に進むと思われる。

① 設計支援ツール

要求仕様化ツール

② 製造支援ツール

プログラムジェネレータ

プログラム部品データベース

構造化エディタ

③ テスト支援ツール

コードオーディタ

パス解析ツール

環境シミュレータ

ディグレードチェック支援ツール

- テストデータ生成ツール
- テスト環境生成管理ツール
- テスト実行検証ツール
- 出力コンパレータ

④ 保守支援ツール

バグ原因解析ツール

- トレーサ
- デバッガ
- バグ情報データベース
- 故障情報自動収集ツール

プログラム修理ツール

- エディタ
- パッチ施工ツール
- 影響範囲抽出ツール
- 更新履歴データベース
- データフロー解析ツール

性能評価支援ツール

- 多端末シミュレータ

遠隔保守情報データベース

④ 管理支援ツール

- 品質管理ツール
- 進捗管理ツール
- 部品管理ツール
- 要員管理ツール
- コスト管理ツール

- ツールデータベース
- システム情報管理ツール

⑤ ドキュメンテーション支援ツール

- ドキュメントエディタ
- オートフローチャータ

6.3 まとめ

以上述べてきたように，現状ではこれといった特効薬は見当たらず，地道にできるところから積み上げることが必要である。

しかし，将来的に技術的ブレークスルーが期待できるものとして，以下のものが考えられ，これらの研究・実用化を一層推進する必要があろう。

① ＡＩ技術を駆使したソフトウェアＣＡＤ／ＣＡＭ技術

多少のあいまいさを含む命令であっても，推論機能によってプログラムの作成ができるようになろう。自然言語にかなり近いレベルの高級言語が発達し，知識ベースシステムと相まって生産性を飛躍的に向上することができよう。

② 言語の高級化

標準化に適したＡｄａ，ＡＩ用言語のＬＩＳＰ，ＰＲＯＬＯＧ，実用化が見込まれるオブジェクト指向言語などによりソフトウェア生産の効率化が図られるであろう。

また，技術面のみでなく，要員育成面，組織管理面でも従来の成功例や効果のあがった手法を継承し，さらに改善していくことにより，ソフトウェア開発の工業化に１歩ずつ近づいていくであろう。

参　考

米国における動向

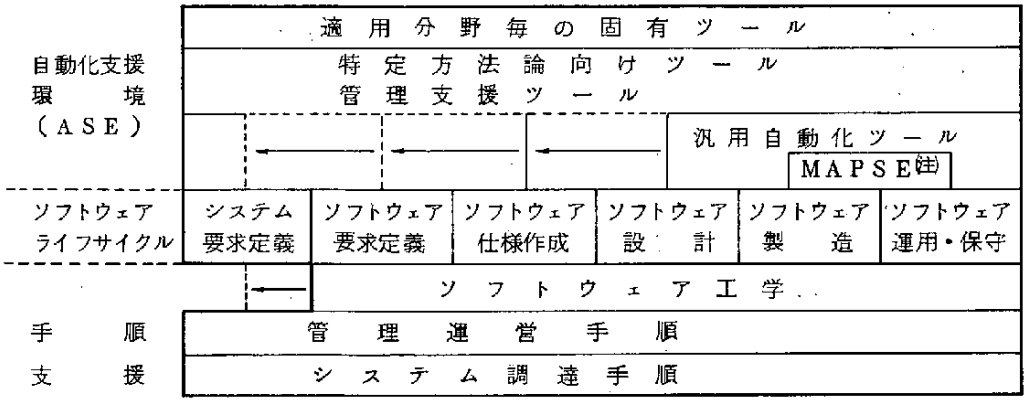
米国ではソフトウェア産業を第二の自動車産業にしないよう，ソフトウェア分野の技術開発に力を注いでいる。ＤｏＤ（米国国防総省）では，新たにSTARS計画というソフトウェア開発支援環境構築のためのプロジェクトを開始した。その概要を図表６－４～６に示す。

ＳＴＡＲＳ計画で対象としているのはソフトウェアの蓄積・再利用，Ａｄａの普及・高度化，知識ベースシステムの構築，組織論や人間の思考過程等人間工学の研究等実に幅広いものである。日本においても，技術的な面での研究・実用化のみでなく，人間工学の面においてももっと力を入れるべきであろう。

図表６－４　STARS計画におけるソフトウェア環境の概念

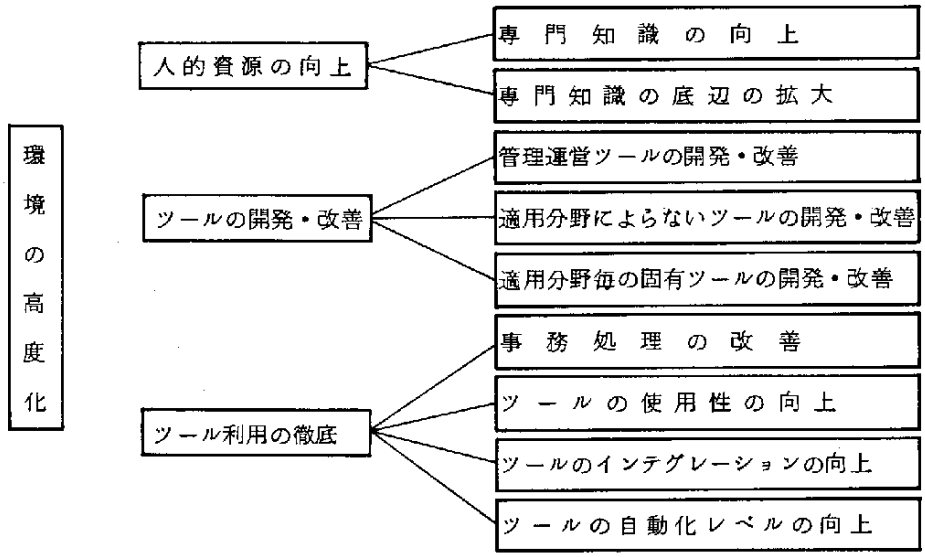
米国ＤＯＤにおけるＳＴＡＲＳ計画

＊Software Technology for Adaptable, Reliable Systems

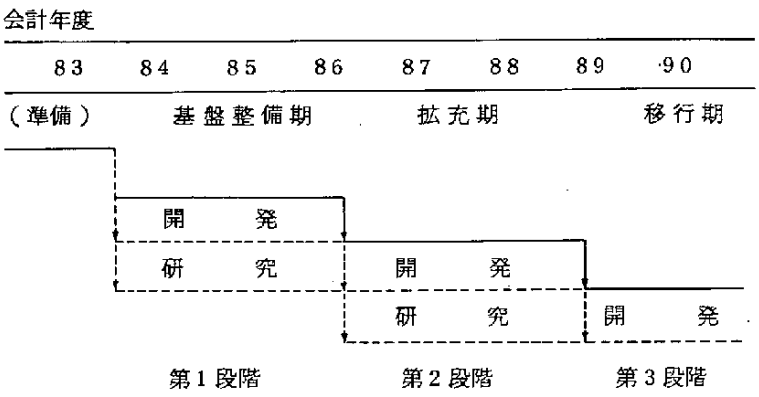


(注) MAPSE：Minimal Ada Programming Support Environment

図表６－５　ＳＴＡＲＳ計画の目標



図表６－６　開発計画



向こう７年間に２億ドル以上の投資を計画している。

7. ま と め

7. まとめ

——3年間の調査研究を終るに当って——

近年におけるソフトウェア開発は，以下の点が特徴としてあげられる。

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に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著しい進歩によって，各種ソフトウェアの改良が必要になっている

⑤ システムの規模の増大や高度化によって，開発技術の高度性が要求されるため，開発要員の不足が顕在化している

このように，システムが大型化，複雑・高度化し，社会生活の中で，一つのインフラストラクチャーとして大きな位置を占めるようになり，そのおよぼす影響が計り知れないものになるにつれて，また，各分野にわたるソフトウェア開発ニーズの高まりによるソフトウェア需給ギャップの増大に伴い，ソフトウェア開発の高度化・効率化，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きたわけである。

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，各組織の実態にそくして積極的に行われているところであるが，当協会（ＪＩＰＤＥＣ）においても１つのプロジェクトを設けて，官民の協力のもとに，昭和58年度より３か年計画で調査研究を行ってきた。すなわち，昭和58年度においては，ソフトウェアの開発計画段階，昭和59年度においては，ソフトウェアの開発段階，昭和60年度は，その運用・保守の段階を，それぞれ中心として行ってきたところである。

3年間にわたる本調査研究を終了するに当り，一つのまとめとして，各年度において，民間企業のコンピュータ・ユーザを対象に行ったアンケート調査結果を中心に，ソフトウェアのライフサイクルにおける各段階，すなわち，開発要求・計画段階，開発段階，運用・保守段階の実態を一括して紹介しておきたい。

なお，このアンケート調査の実施に当り，業務ご多用中にもかかわらず，また，

この種の調査が多くの機関で実施されているという煩わしさにもかかわらず，積極的にご協力をいただいた民間企業（コンピュータ・ユーザ）各位に対し，厚くお礼を申し上げる次第である。

7.1 ソフトウェア開発計画段階の方策

7.1.1 要求仕様の明確化

ソフトウェア開発の計画段階において重要なのは，要求仕様の明確化である。ソフトウェアの開発は，要求仕様に基づいて行われるものであるから，必要かつ十分な要求仕様を定義することは，ソフトウェア開発を成功させる第一歩であるといえる。この点に関して，当協会（ＪＩＰＤＥＣ）が実施したアンケート調査「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（以下「アンケート調査」という。）」によると，要求仕様に関してはつぎのような問題があるとしている。

①システム化に関する検討が不十分

②ユーザの意向が反映されない

③将来動向の見極めが不十分

④技法などの活用がなされていない，など。

したがって，要求仕様を明確化するための方策としては，つぎの点を検討していく必要がある。

(1) 費用対効果を中心としたシステム化に関する十分な検討

ある業務をシステム化しようとする場合，最も重要なことは，システム化による効果の予測，つまり費用対効果の分析である。図表７－１は，効果測定項目体系を示したものであるが，このように種々の観点からシステムの効果について十分な検討を行うことが必要である。

図表７－１　効果測定項目体系

区分 / 影響局面		測定項目＊	効果（電算化前に比べて，以下の事例があれば効果である。）	マイナス効果（電算化前に比べて，以下の事例があればマイナス効果である。）
管理局面	業務処理	迅速性	迅速化	遅延化
		正確性	正確化	不正確化
		簡易性	簡素化・標準化	煩雑化
		弾力性	弾力化	硬直化
	データの利用	適時性	向上	低下
		容易性	容易化・高度化	不便（読みにくいなど）
		利用範囲	拡大	縮少
	データの管理	容易性	管理の容易・適切	管理の困難・不適切
		安全性	安全性（秘密保護など）向上	安全性低下
	人事・組織	モラール	勤務環境改善	勤務環境悪化
			モラール向上	モラール低下
		連携	整備・連携の円滑化	連携の組語など
コスト局面	—	—	電算化前の事務処理方式を続けた場合の経費と，現在の方式での経費比較	
			経費減少	経費増大

＊この項目の範囲は，「コンピュータ技術へ直接的な影響を及ぼした効果」とする。

出典：「電子計算機利用の効率化に関するガイドライン(1)」行政管理庁

(2)　ユーザ部門との十分なコミュニケーション

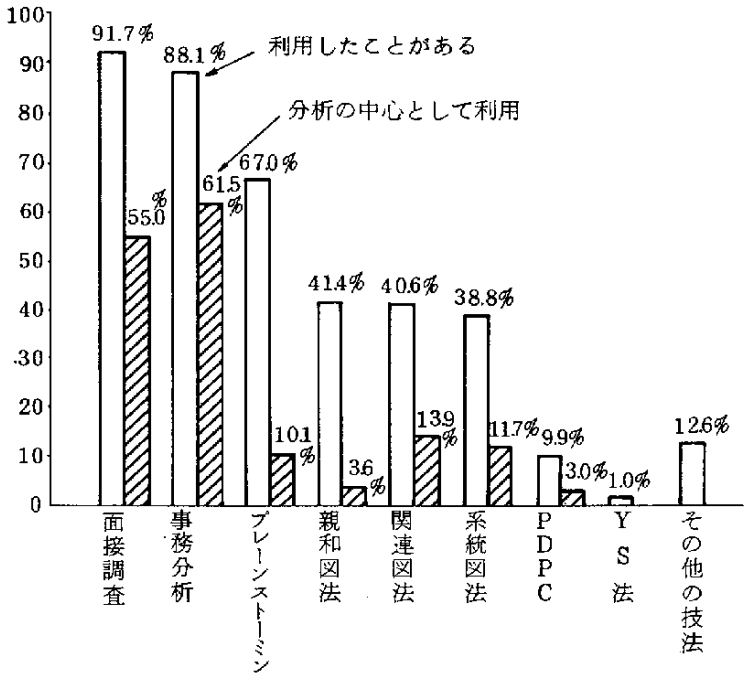
ソフトウェアの開発は，ユーザの意向を十分に反映させた形で進めなければならないが，アンケート調査（前記）によると，コンピュータ部門が自己の都合のよいように仕様を決めてしまうことが多いとしている。要求仕様は，原則的にはユーザ部門が独自に作成するべきであるが，コンピュータ部門がそれに関与する場合は，ユーザとの十分なコミュニケーションを図り，ユーザの意向を正しく反映するようにする必要がある。また，この場合，将来の技術動向や企業戦略といったことも十分検討し，ライフサイクルの長いシステム作りを目指すべきであろう。

(3) 技法などの積極的活用

要求仕様をより信頼性の高いものにするためには，各種技法やツールを積極的に活用していくことも方策の一つであろう。

アンケート調査によると，各技法の利用状況は，図表７－２のとおりである。

図表７－２　要求定義技法の利用度

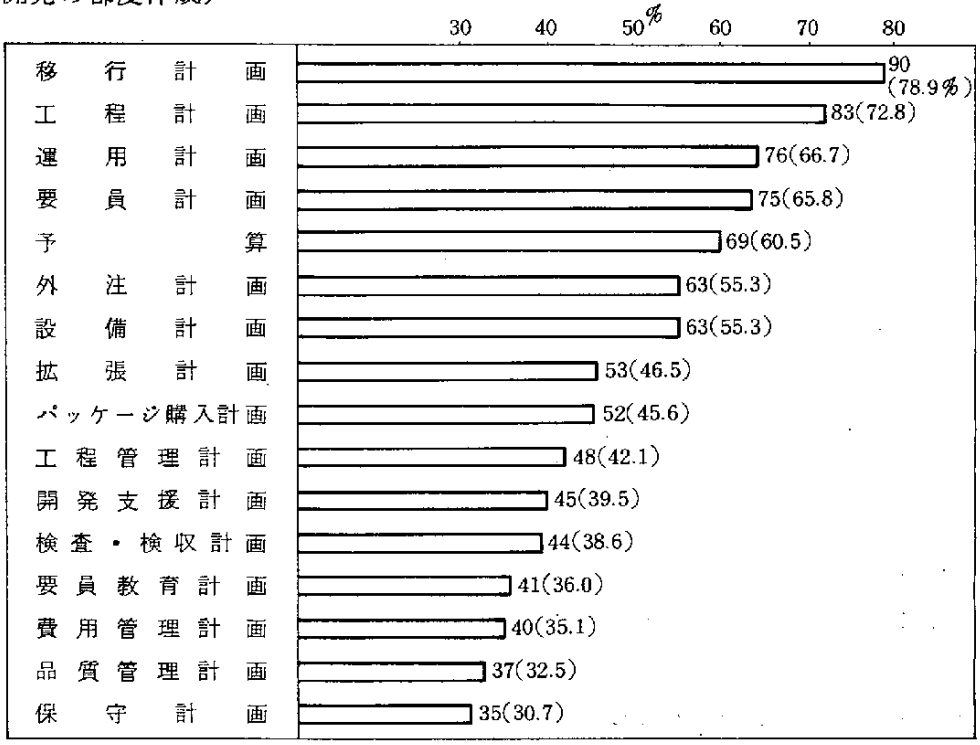


7.1.2　開発計画の確実な立案

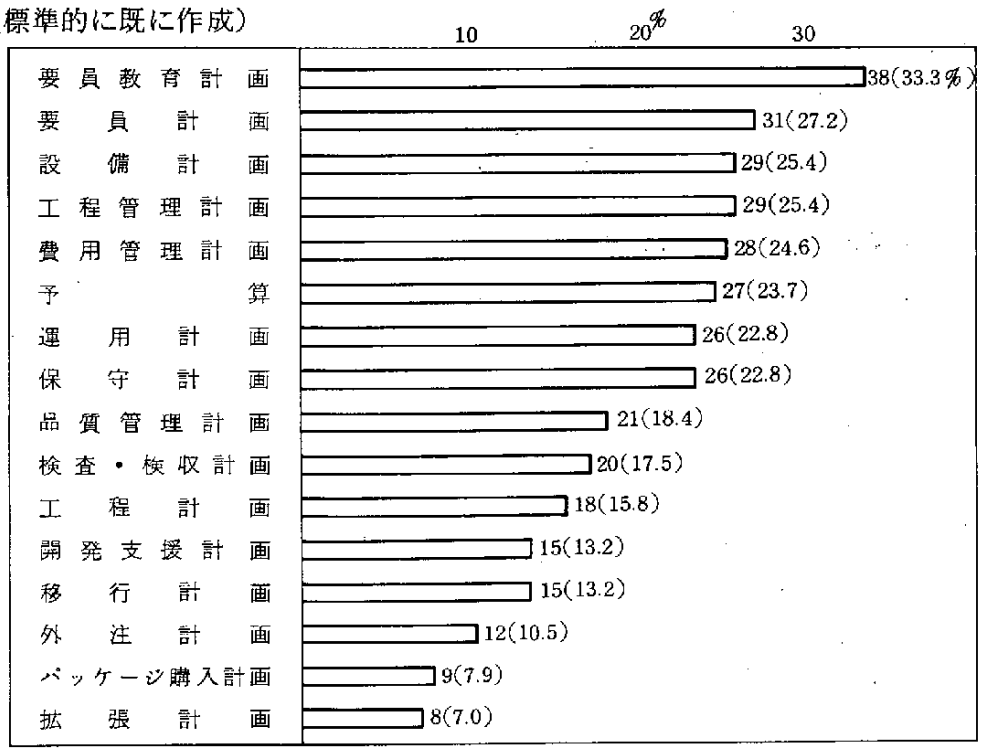
ソフトウェア開発計画の段階において立案される計画には，①ソフトウェア開発における見積り，予算，スケジュールなどに関する計画，②開発作業の実施における管理および支援などに関する計画，③移行，運用，保守などに関する計画などがあるが，アンケート調査によると，その作成状況は図表７－３のとおりであるが，例えば，工程計画，予算，要員計画，移行計画といった非常に重要な計画であっても，その作成状況は７０％程度となっている。少なくともこれらの計画を確実に作ることが必要である。

図表７－３　開発計画の作成状況

（開発の都度作成）



（標準的に既に作成）



また，計画立案の際に利用できる技法（WBS，TRM，PERTなど）についても積極的に活用し，さらに，開発する規模や必要となる費用の見積り方法についても各企業においてデータを収集し，可能な限り科学的に行うことを考えるべきであろう。

7.1.3 品質に対する意識の向上

最近では，新規開発よりも既に開発したプログラムの保守作業にかかる割合が大きくなってきており，今後ますますその傾向は高まると考えられるが，この保守を容易にするためには，プログラムが保守性を中心として高い品質を有していることが必要である。つまり，正確性や信頼性といったソフトウェアが本来的に必要である品質特性の他に，他人がみても理解しやすいプログラムであることが重要となるのである。このような保守性を含めたソフトウェアの品質を考えると，品質を高めることは生産性を下げることになるという見方もあるが，高い品質を持つプログラムを作ることが結果的に高い生産性を得ることになるとの観点から意識の向上を図っていく必要がある。

7.2 ソフトウェアの開発段階の方策

ソフトウェアの生産性を向上させるためには，種々の方策があるが，何れの方策といえども，その方法を採用することによって必ずしも同様の効果を期待できるとは限らない。

そこで，これらの方策を効果の面に視点を置いて分類してみるとつぎのようにグルーピングされる。

(1) 生産性向上のための基礎的な方策

生産性を向上させるための前提条件的なもので，これが不十分であれば(2)，(3)の方策が期待したほどの効果を発揮でき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方策

これらの方策でも，十分な効果を発揮させるためには，種々の環境条件の整備などが必要である。

① 機械化 ヒ

② 再利用

③ ソフトウェア・パッケージの活用

⑶ 生産性向上に遅効性の効果を発揮させる方策

これらの方策を導入したことによって，直ちに大きな効果を発揮するという性格のものではなく，その導入によって着実に効果を発揮し，生産性向上に対し安定した効果を与えるものと考えられる。

① 品質管理活動

② 開発環境

③ 要員管理

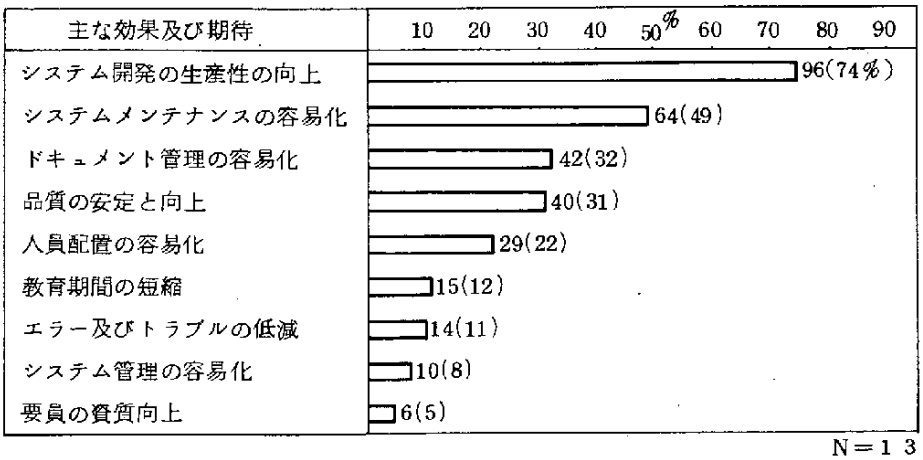
以上について，7.2.1項から7.2.6項でその概要を述べる。

7.2.1 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。

アンケート調査によると，図表7－4のとおり標準化の推進によって，システム開発の生産性の向上をあげるものが多いことから，標準化は，生産性向上のための基本的な手法といえよう。標準化は，開発工程におけるプログラム設計，プログラミング，ドキュメント作成の各段階，用語，フォームシート，コーディングなどを対象として行われている。

図表7－4 標準化の効果



N＝130

7.2.2　組織化

ソフトウェア開発の効率化のためには，固有技術の高度化だけでなく，組織的な対応も重要である。すなわち，ソフトウェア開発を限られた人数（要員）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，活用して最大の組織的効果を確保できるようなマネジメントが必要である。

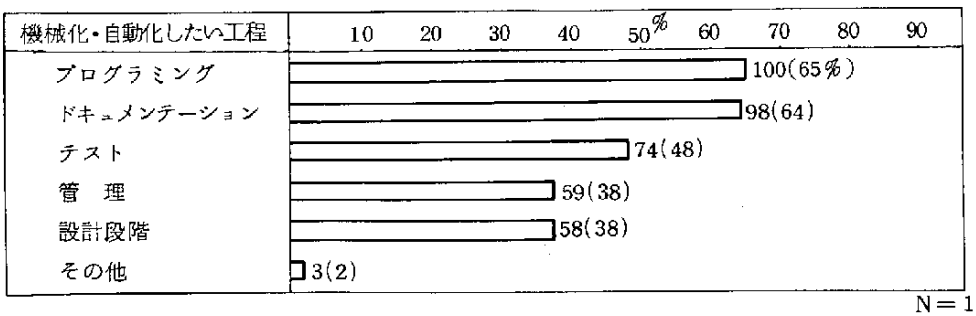
組織化の問題としては，ソフトウェア開発における分業化の考え方，品質検査組織，要員不足をカバーするための対策（外注利用，女子プログラマの積極的活用，在宅勤務制度の導入，エンドユーザ開発の推進などによる開発パワーの増大）などがある。

7.2.3　機械化・自動化

人間の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成することが可能となる。ソフトウェア開発は従来から職人芸として手作業による開発が中心であったが，情報化の進展に伴って，ソフトウェアのニーズの増大とそれに応えるソフトウェア技術者の不足による需給ギャップ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，自動化の導入がクローズアップされてきている。

アンケート調査によると，開発工程の中で機械化・自動化したいと考えている工程として，図表７－５のとおり，プログラミングおよびドキュメンテーションの段階をあげている企業が多い。これらの段階が最も工数がかかり，逆に，機械化しやすい工程であると考えられている。なお，機械化・自動化は，単に生産性向上のみならず，信頼性，保守性，汎用性の高いソフトウェア開発をめざしている。

図表７－５　機械化・自動化したい工程



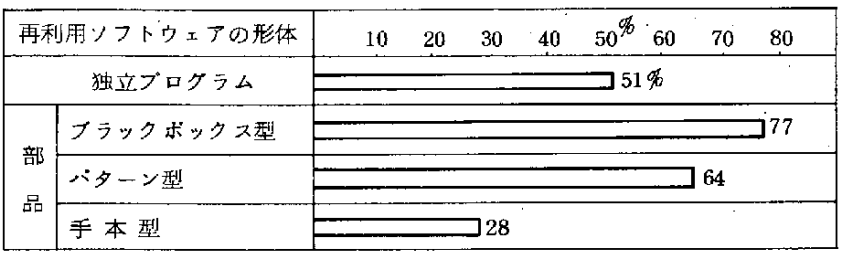
N＝１５７

7.2.4　再利用

ソフトウェアの生産性，品質の向上のための有効な手段として，既存ソフトウェアの再利用がある。その利点としては，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，再利用ソフトウェアの品質が既に検証されていることなどが考えられる。

再利用ソフトウェアの形態としては，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独立プログラムと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作動する部品としてのプログラムがあるが，最近では，プログラムのこのような部品化が注目されている。アンケート調査では，図表７－６のとおり，独立プログラムの再利用よりも部品化に関心をもつ企業が多いことを示している。

図表７－６　再利用ソフトウェアの形態別利用状況



7.2.5　ソフトウェアパッケージの活用

ソフトウェア開発の生産性向上を図るための効果的な方法の１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，自社開発よりコストが安い，開発期間がとれない，自社開発の工数不足，機能・性能がよい，といったような場合に導入が考えられる。

7.2.6　品質管理活動，開発環境，要員管理

これらの方策の導入によって，直ちにその効果を期待できるものではないが，その実施によって生産に安定した効果が期待できるものと考えられる。

(1)　品質管理活動

ＥＤＰ部門のかか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課題であるが，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１ステップである。特に，設計工程の品質向上はソフトウェアサイクルにおける共産性向上に大きく寄与している。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，生産活動全般に亘る作業改善や開発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とする小集団活動である。アンケート調査では，約８０％の企業が品質管理活動に取り組んでおり，そのうちＥＤＰ部門がその活動に参加している企業は５０％である。ＥＤＰ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と考えられる。

(2) 開発環境

ソフトウェアの開発は，知的で根気を要する作業である。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存している状態を脱しきっていないのが現状である。ソフトウェア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努めることが必要である。また，これは，優秀な要員を確保するためにもすぐれた作業環境の整備が重要である。アンケート調査では，多くの企業が作業環境の配慮のポイントを採光，換気，机のレイアウト，騒音などにおいている。

(3) 要員管理

ソフトウェア危機の１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。アンケート調査では，多くの企業が何らかの形で教育訓練の体系的実施，モラール向上など処遇改善策などを講じている。

7.3 ソフトウェアの運用・保守段階の方策

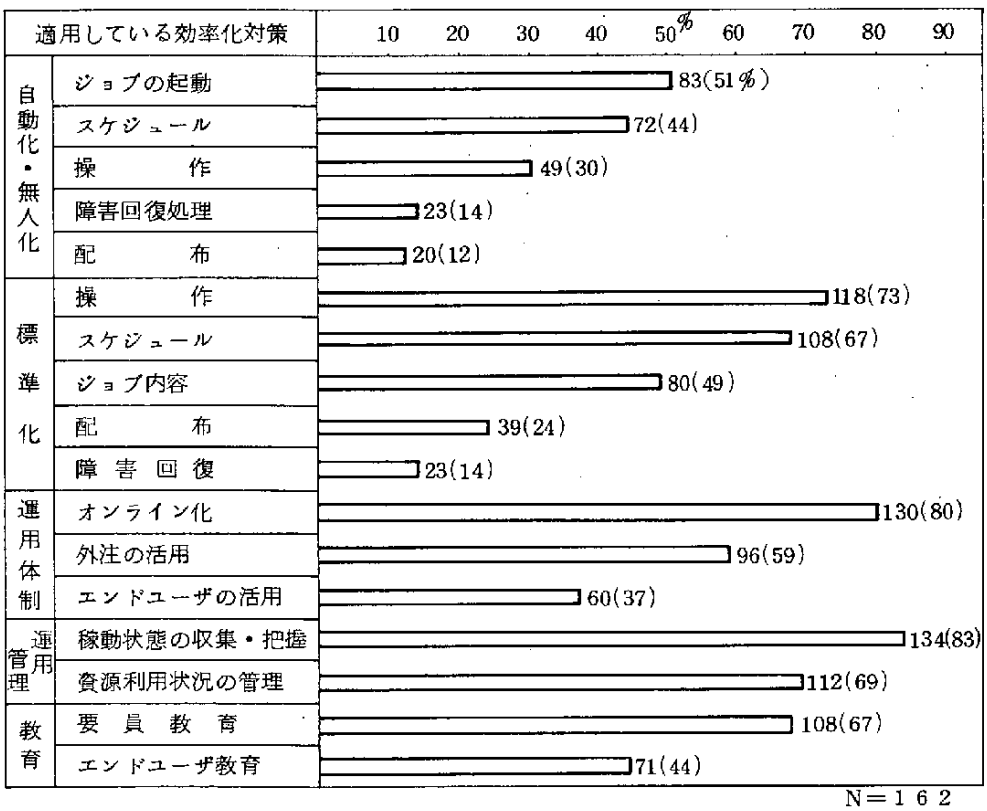
7.3.1 運用の効率化方策

現在のコンピュータの稼動状況は，２４時間フル稼動のユーザ（25.6％），および，２０時間程度まで稼動させているユーザ（２５.６％）で過半数を占めており，相対的に長時間使われている。このためのオペレーション体制も１

～３シフト制の割合が各々３０％程度とほぼ同じになっている。つまり，１直制で１０～１５時間，２直制で１５～２０時間，３直制２０～２４時間といった運用パターンが考えられる。

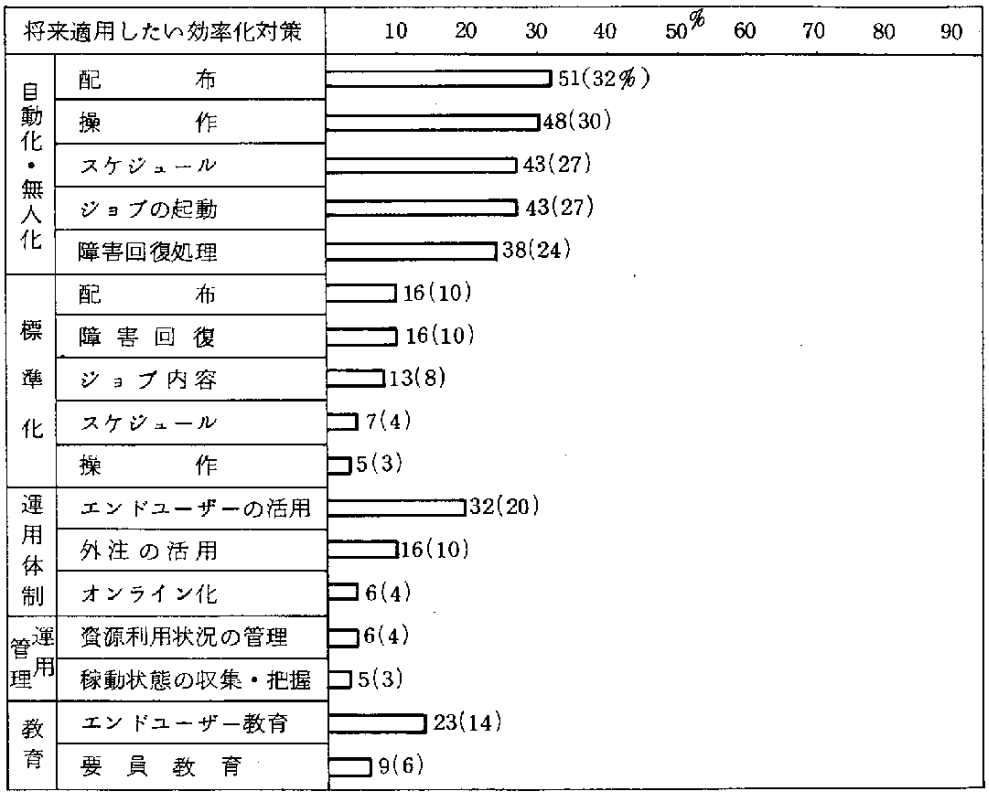
このような運用状況を，効率化の側面から，現在行われている方策として自動化・無人化，標準化，運用体制，運用管理，教育についてみてみると，図表７－７のとおり運用管理面での効率化対策が比較的進められており，稼動状態の収集・把握は８３％のユーザにおいて，また，資源利用状況の管理も６９％のユーザにおいて効率化などがとられている。つぎに，運用体制の面では，オンライン化および外注の活用がすすんでいる。標準化の面では，操作，スケジューリング，ジョブ内容の標準化が比較的進んでいる。教官の面では，委員教育が中心となっているが，エンドユーザ教育に効率化の焦点を当てているユーザも約４割みられる。自動化・無人化については，ジョブの起動，スケジュール以外はまだ対策が進んでいない状況にある。

図表７－７　現在適用している効率化対策



以上が現在行われている運用面での効率化対策であるが，将来適用したい効率化対策としては，図表７－８のとおり，自動化・無人化を進めたいと考えているユーザがより多くみられる。自動化・無人化以外では，運用体制面でのエンドユーザの活用や教育面でのエンドユーザの教育を考えているものがいることを示している。

図表７－８　将来適用したい効率化対策



N＝１６２

7.3.2　保守の効率化方策

保守作業について，全体の作業工数に対する割合を職種別にみてみると，図表７－９のとおり。ＳＥおよびプログラマーが平均５３.８％，管理者およびオペレータがそれぞれ平均１３.８％，１３.１％の工数を保守作業に振り向けている。この中で，注目されるのは，ソフトウェア開発の主力となるＳＥおよびプログラマーが平均で５３.８％，つまり総工数の半分以上を保守作業に費されていることとである。保守作業の割合が７０％以上となるユーザは，

全回答企業の4割近くを占めており，多くのコンピュータ・ユーザがバックログをかかえる要素をもっていることを示している。

図表７－９　職種別の総工数の中で保守作業の占める割合

職種＼割合(件数)	10%	20	30	40	50	60	70	80	90	100	平均
ＳＥ及びプログラマ	11	9	21	15	15	23	26	10	15	6	53.8
オペレータ	51	25	13	2	1	3	1	2	1	1	13.1
管理者	62	27	10	6	2	2	4	1	0	0	13.8

N＝１５１

保守の種類別では，完全化保守の割合が高く（平均５３.３％），修理保守及び適応保守は比較的少ないものとなっている。

また，保守を作業工程別にみてみると，図表７－１０のとおり，プログラムの修正，プログラムの追跡，プログラムのテストなどの作業工程に多くの工数を費しており，保守における効率化は，これらの点に焦点を当てて具体的な方策の展開が必要であろう。

図表７－１０　保守の工程別の作業工程の割合

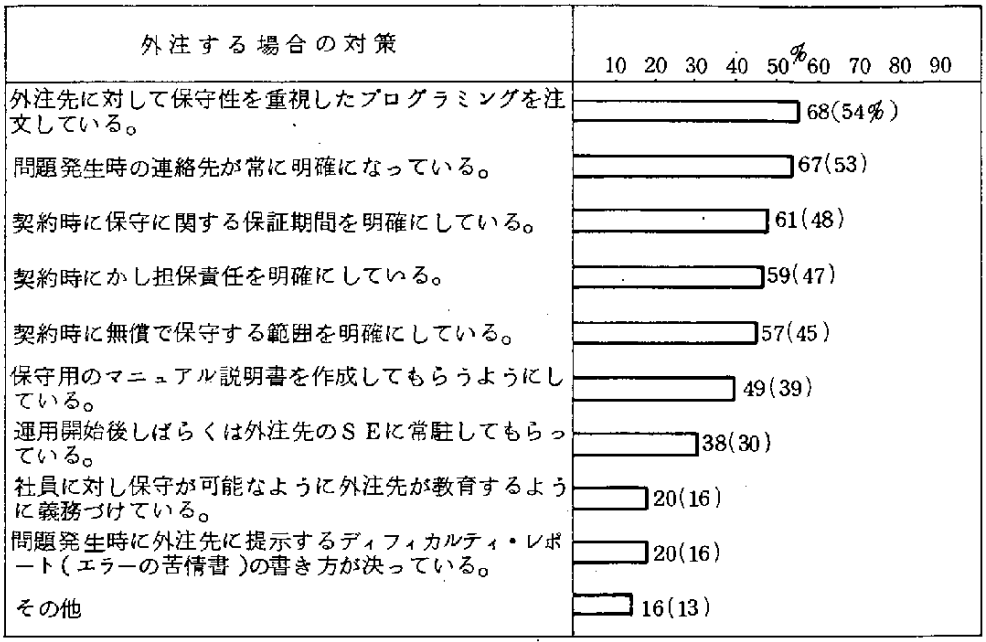
作業工程＼割合(件数)	10%	20	30	40	50	60	70	80	90	100	平均
ドキュメントを調べる	77	31	21	10	3	2	0	0	0	0	16.9
ドキュメントを作成する	86	40	13	2	0	0	0	0	0	0	12.7
プログラムを追跡する	40	45	37	14	6	2	5	0	0	0	23.9
プログラムを修正する	36	53	39	11	6	5	0	2	0	0	22.3
プログラムのテストを行う	43	52	42	9	5	1	0	0	0	0	21.4

N＝１５２

ところで，ソフトウェアの開発は，自社開発と併せて，外注して行われる場合も多いが，外注する場合の対策としては，①外注先に対して保守性を重視したプログラミングの注文，②問題発生時の連絡体制の明確化，③契約時の必須事項として，かし担保責任，保守に対する保証期間，無償保守の範囲

など，それぞれ約５０％のユーザが実施しており，ソフトウェアの保守をいかに効率よく，コストをかけずに行うかにユーザが苦心していることを物語っている。

図表７－１１　ソフトウェアを外注する場合の対策



Ｎ＝１２６

ソフトウェアの保守を行う場合の問題点として，①要員の問題，②エンドユーザ関連の問題，③応用システム関連の問題，④その他の問題について，アンケート調査では質問しているが，要員問題については，保守要員の交替や保守要員の数に深刻な問題があるとしてあげている。また，エンドユーザとの関連については，エンドユーザ組織の交替やシステムに対する不理解を問題としている。その他の問題としては，保守作業の効率の悪い点や保守工数の見積りの問題が深刻であると指摘している。

昭和61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
Tel（254）5840