経情協47－1

コンピュータ情報システムの開発における

システム・ライフ・サイクルのマネジメント

昭 和 48 年 3 月

財団法人　日本経営情報開発協会

この報告書は昭和47年度における日本小型自動車振興会から小型自動車競走法に基づく小型自動車等機械工業振興資金の交付を受けて作成したものであります。

ま　え　が　き

「システム開発委員会」は、我が国企業がコンピュータを用いる情報システムの開発に当たって感じている問題点の解決の方向を探求するために、（財）日本経営情報開発協会に設けられた研究会である。

その主旨は、主として米国で発達しているシステムズ・マネジメメトの概念と技法を、我が国企業の実情に合うものにして、1つの典型的ガイドラインを作成することによって、コンピュータのユーザーである多くの企業にとって有用な参照資料を提示しようというものである。

メンバーは、我が国において代表的な大規模システムの開発を、みずからの手で中心人物となって進めてきた体験者を主とし、昭和47年4月から研究が行なわれてきた。方法としては、①共通の議論の場をつくるため、文献（Rubin 他著、Handbook of Data Prccessing Management，全6巻、Brandon Press & Auerbach, 1970）の第1巻および第6巻を通読する。②担当者からの上記文献の部分的紹介にもとづき、各自の印象・感想・体験などを、フリーな座談形式で討議し、記録する。③報告書に盛り込むべき主要なトピックを選定し、それまでの討議内容を参考にしながら、委員が分担して第1次原稿を作成する。④第1次原稿をプリントして全員に配布し、批判・補足・修正などを加える。といった順序で行なうことにした。こうした委員の討議にもとづくメインのストリームを補強するために、その他の文献調査、我が国企業へのアンケート並びにインタービュなどを行なった。

研究会として残念なことであったのは、年度半ばにして委員のうち2名が海外転勤、他の2名がこの研究会のテーマそのものの大規模システムの開発責任者として急拠多忙になり、さらに他に1名短期ではあったが海外出張する

という思いがけない事態で、このため委員会はいったん壊滅状態になったが、その後をうけてよくこの報告書完成にまでこぎつけたのは、各委員の推挙によって加わって頂いた作業委員の方々と、慶応義塾大学工学部管理工学科川瀬研究室の応援のお陰である。また、協会事務局の御苦労も大変であった。

こうした事態のため、最終報告書の段階で、各項目間に多少のアンバランスが見られるし、内容的にも、ごく一部の委員だけの意見がそのまま出ている調整不足の部分もある。しかし、上記のような問題意識にもとづいて構成されたこの報告書は、今後のシステム開発関係者にとって、それなりの参考になると信じている。この報告書を「たたき台」として、さらに多くの経験、知見を集積することによって、まさに「ガイドライン」の名にふさわしいハンドバックが生み出されるにちがないと期待する。

本報告書の構成

第Ⅰ部「システム・ライフ・サイクル概説は、本報告書でとりあげたアプローチの基底を考察し、フィロソフィカルな議論をふくんでいる。もちろん、ある程度の理想論であってそのフィロソフィがすべて本書に反映されているわけではないが、一応の目的意識の整理には役立つであろう。

本書の主要部分は第Ⅱ部「システム・ライフ・サイクル」と第Ⅲ部「システムズ・マネジメント」である。第Ⅱ部は、システム・ライフの作業内容ないしはその構造を、フェイズに則して解説したものである。ここで、マネジメントすべき対象の特性が明らかにされ、プロジェクト・マネジメントやシステム・マネジメントを担当するマネジャーの手引きとなる。また同時に、より詳細なステップ内で述べられる注意点（おとし穴）は、経験をつんだ一流専門家の貴重な方法論をすこしでもこの専門領域の共有財産として位置づ

けるためのフレームワークでもある。本報告書では、委員の方々の知見の開陳という点で、この後者のねらいは一部分達せられたと思うが、前者のねらいから見るといくつか触れられなかった側面もあって、ユニバサルな「ガイドライン」としては未完成といわざるを得ない。そうした前提でみて頂ければと思う。

第Ⅲ部は、システム開発というプロジェクトが置かれる環境の整備と、それを前提としたマネジメントとをとり上げたものである。プロジェクトが異なる専門を持つ多くの人々による、組織全体にわたる長期の活動である以上、その効率的運用と作成されるシステムの品質とに、こうした周辺条件が大きく影響するのは当然であろう。本書では、第1章でプロジェクト・マネジメントそのものを少し詳しくとり上げたが、これは一部の例外を除いて我が国にまだ本格的なプロジェクト・マネジメント機能が定着していない現状に対する問題意識を反映したものである。

また、プロジェクトをめぐる諸条件のうち、とくに重要であると思われたトピックスをえらんで、第2章以下にとり上げた。もちろん、それ以外にも人事問題、予算問題、設備問題、ドキュメンテーション等主要な問題はあったのだが、研究期間の制約から主観的にふるい分けざるを得なかった。

第Ⅳ部は、以上にまとめられた各委員の研究・討議の結果を別の角度から補足する目的で、我が国企業の実態にアプローチを試みた結果をまとめている。アンケート対象範囲のかたより、インタービュ訪問件数の少なさなどを前提としてみて頂きたい。

なお、付属資料として、米国における考え方と実務への適用の全貌を、それぞれコンパクトにまとめた文献を訳出した。こうした発展途上の分野の例として、用語の不統一も若干見られるが、概念そのものには一貫性があると思う。

参　考　文　献

◎ Rubin ,et al.;Handbook of Data Prcessing Management, 全6巻、
Brandon Press & Auerbach, 1970.

◎ Brandon and Gray ; Project Control Standards, Auerbach, 1970.

◎ Hartman, et al. ; Information Systems Handbook,Philips, 1968.

◎ Benjamin ; Control of the Information System Developmem Cycle,
Wiley, 1971.

システム開発委員会委員名簿

（五十音順・敬称略）

委員長	松田武彦	東京工業大学工学部教授
委員	稲田伸一	鉄道技術研究所・システム研究室・主任研究員
	井上義祐	新日本製鉄㈱・情報システム部・企画二課長
	大島昭	日本放送協会・経営情報室・副主管
	竹下享	日本アイ・ビー・エム㈱・DPビジネス計画マネジャーアシスタント
	木村幸信	神戸商科大学・経済研究所助教授
専門委員	籃原義信	新日本製鉄㈱・情報システム部・調整掛
	内橋勤	日本アイ・ビー・エム㈱・プログラミング担当アドバイザリーSE
	金沢孝	慶応大学工学部大学院
	笹間宏	鉄道技術研究所・システム研究室
	槻木公二	同上
	辻新六	慶応大学工学部大学院
	中村善太郎	慶応大学工学部講師
	長森靖彦	日本放送協会・経営情報室
	藤森聡二	鉄道技術研究所・自動制御研究室
	安井清享	日本アイ・ビー・エム㈱・DPインストレーション・プログラム担当シニアSE
	脇田和郎	新日本製鉄㈱・情報システム部企画二課・掛長

目　　次

Ⅲ　システムズ・マネジメント

Ⅳ　わが国におけるシステム・ライフ・サイクルの実態

I　システム・ライフ・サイクル概説

I　システム・ライフ・サイクル概説

1. システム・ライフ・サイクルとは

システム・ライフ・サイクル（ＳＬＣ）とは、ある1つのシステム開発作業を全体として把握する概念である。それは、主としてコンピュータを用いた経営情報システムについて、そうしたシステムの必要性が感知されて、検討が開始される段階から、システムが具体化されて活動を行ない、やがて次の新しいシステムにとって代わられるまでを、人間の生涯にみたててライフ・サイクルという見方をあてはめようとするものである。それによって、作られるシステムの持つ品質を良いものとし、開発作業のマネジメントを容易ならしめようとするものである。

この考え方は、コンピュータという文明の利器を、企業経営に役立てようという努力の中から生れてきたものであって、主に米国において多くの経験と研究が蓄積され公表されている。米国について世界第2位のコンピュータ利用国である我が国の場合も、その技術的・経済的背景は米国に共通する点は多い。したがって、このＳＬＣという考え方は技術的・経済的な面から言えば、当然我が国の企業に導入すべきものであり、事実、意図的な導入や自然発生的な提唱によって、多くの企業で採用されている。しかし、それが本当に効果を発揮するためには、日米両国に大きな差がある社会的背景に対する適切な考慮を払う必要がある。多くの識者が、形だけのＳＬＣ概念の〝輸入〟に対して危惧を抱いているのもこのためである。

ＳＬＣの内容は、第Ⅱ部以降で詳述されるので、ここでは改めてとり上げないが、その考え方の基盤は一種の企業原則であり、さらに辿るとするならば、解析（細分化）原理である。いわゆる西欧文明の中で発展してき

た自然科学、およびその影響を受けている社会科学のひとつの基本的態度として、どんなに複雑に見える現象でも、それを多くの要素に細分化してみれば、比較的単純な法則が成り立つので、それら要素相互間の関係に沿ってその法則を総合化することによって複雑な現象が説明できるというのである。現象の説明は、より意図的・目的的に予測、設計（方策）へと演繹されて行き、人間の行動指針を与えるということになる。

ＳＬＣにおいては、システム開発作業の全生涯をいくつかの段階（フェイズ）に分けてとられ、さらにステップ、タスク等へと細分化を行なう。細分化された要素相互間の関係は、情報のコミュニケーションおよび作業間のチェック・ポイントの体系によって構造化されている。ここに、巨大な全体の統一的な見方と多数の要素の部分的な見方という、一見矛盾する２つの観点が総合されるのである。こうして、各要素作業の専門化を計って効率の向上をめざすと同時に、専門化された異質の努力が共通の目的に向かって統合されるという体制が完成する。

細分化された各要素について、比較的単純な法則性が成り立つことを認識すると、その要素作業の内容を客観的に記述し、成果を数量的に予測することが可能になる。これがストラクチュアル・スタンダードとパフォーマンススタンダードの２つを構成する。産業革命以来、もの（ハードウエア）の生産活動において進められてきた標準化の概念は、ここに、システム開発作業という知的生産活動に対しても適用されることになる。

工業生産活動の分野で標準化の思想を形式的に輸入した我国においては、標準で人間をしばりつけることは人間の創造意欲、作業意欲を失わせ、人間性をそこなうものだという批判が強い。たしかに、画一化という、もっとも安易なやり方をとる標準化においては、「・・・・してはならぬ」といっ

た表現の集合によって、作業の非人間化がおこなわれるかも知れない。しかし、より高いレベルの標準化の目ざすところは、「少なくとも・・・しておけば、最低限のパフォーマンスは確保できる」状態を、その作業の担当者に指示することである。そこでは標準は人間の創造性を上から押えつける制約要因ではなく、むしろその上に立って能力をフルに発揮できるような、しっかりした下支えとなるべきである。

ＳＬＣの概念は、従来主として、コンピュータ情報システムの開発について発展してきた。しかし、より一般化した見地から言えば、科学技術の巨大プロジェクトについても、プロジェクト・マネジメントという立場から、同様な考え方が発展してきている。さまざまな学問分野での知見を交流して、いわゆるインターディシプリナリー（学際的）な進め方でプロジェクトの開発を効果的ならしめるためにも、そのフレームワークとなるＳＬＣ概念の、今後のより一層の確立に努力しなければならないであろう。

2. ＳＬＣコンセプトを必要とする背景

a. システム開発プロジェクトの大規模化

企業経営そのものの巨大化および高度化からくるコンピュータ情報システムへのニーズと、それに応えようとするハードウェア、ソフトウェア両面でのテクノロジーの進歩は、お互いに相手を刺激し合うという形で急速にふくれ上ってきた。トータル・システムへの意欲は、それを処理できるだけのハードウェアの進歩を促すし、オンライン技術の具体化は広域にわたる情報ネットワークの開発意欲をかき立てることになる。

企業内におけるデータ処理システムの形態も、その初期においてよく見られたような、すでに確立されている個別業務における人手による処

理。計算をたんに機械化すればよいといったものから、企業の全部門に影響を及ぼすような横断的経営情報システムへと変化してきた。

オンライン・リアルタイム・システムの適用分野は、最初にそれに対するニーズが発生した生産工程のプロセス・コントロールや全国的規模にわたる販売。取引情報のコミュニケーションなどに止まらず、経営者の意思決定。計画策定プロセスへの援助といった高度の精神作業面にまで及んでいる。

こうした、コンピュータ・アプリケーションに対する要求の多様化と、それに対処するための技術の進歩という2つの要因が、さらに相互補強的な効果をもって、システム開発プロジェクトは大規模化の一途をたどっている。もはや、事情によく通じた1人ないしは数人のプログラマーのグループが、職人的。手工業的なやり方でシステムを作り上げていく時代ではなくなってしまった。

上述の、ニーズの多様化から最新のテクノロジーの駆使に至るまでの幅広いスペクトラムに対応するには、多勢の、それぞれ異なる側面を得意とする専門家たちの、長い期間にわたる協力を必要とする。

かくして、システム開発プロジェクトは次のような3つの状況を生み出し、その各々がここで論じているＳＬＣ概念を必要とする前提を構成するのである。それは、

イ．システム開発作業の長期化

ロ．訓練され、専門化された要員の不足

ハ．プロジェクト・マネジメントの困難化

であり、こうした問題の解決に有力な手がかりを与えてくれる権威あるガイドラインの確立が望まれるゆえんでもある。これらの項目を個々に、も

う少し詳しく見ていくことにしよう。

b. システム開発作業の長期化

システムの全寿命中に占める開発期間の割合は大きくなる一方である。当研究委員会メンバーの一致した見解として、この割合が５０％、すなわち、３年かけて開発したシステムは、せいぜい開発・導入後３年ぐらいしか稼動しないというのである。こうした比率の増大は、企業環境・経済状勢の変化が激しいこと、技術革新の急速なこととによる、開発されたシステムの運転寿命が短縮していることもあるが、開発作業の長期化も直接の原因である。

前項に述べたように、企業内でコンピュータの性能・特長に対する理解が普及するにつれて、いろんな部門からいろんな要求が出てくる。型にはまった単純反復作業のコンピュータ化とちがって、そうした要求を出してくるユーザーの真のニーズおよびリクァイアメントを把握することは容易ではなくなり、システム開発の初期に行なわれる業務分析、予備設計の重要性が増してくる。

米国および我が国における幾つかの苦い経験の教訓として、こうした、システム開発作業の前半にくるフェイズに十分な時間と努力を投入することが、プロジェクト全体としての成果を向上させるものであることが判っている。しかし、これは、ただ漫然と、このアナリシス段階での時間の経過を待ち、アナリストを完全に放任しておくことが好結果をもたらすという意味ではない。むしろ、時間と努力の投入が、よくマネージされた環境で行なわれてはじめて、プロジェクトに好影響を及ぼすということなのである。

この意味で、われわれが考えるＳＬＣ体系の中には、このシステムズ・

アナリシスの方法論（Methodology）が含まれることが必要である。複雑な状況下で日夜意思決定を迫られているユーザー部門のマネジメントたちの、必ずしも整理されていない断片的・個別的な希望や要求を、直接的。間接的な諸調査を通じてさぐり出し、より具体性をもち、斉合性のある、一群のシステム・リクアイアメントにまとめ上げるという作業は、従来、システム設計担当者の個人的・主観的な思考展開を彼の頭の中で行なうという形をとっていた。ここでMethodologyというのは、そうしたとらえどころのない作業過程をより詳細・具体的に記述することによって、担当するシステムズ・アナリストに行なうべき作業手順のガイドラインを与えるとともに、そのガイドラインに支えられた足場に乗って彼に創造能力発揮の機会を与えようとするものである。

また、プロジェクト・マネジメントの担当者は、必ずしもシステムズ・アナリシスの専門家でないから、作業過程の詳細な分割と相互関連をマイルストン（チェックポイント）を介して与えられて始めて、この知的で創造的な作業のコントロールが可能になる。

ここでは、全サイクルの始めにくる業務分析。予備設計フェイズにおける意義を強調したが、詳細予備設計フェイズ以降についても、ＳＬＣのもつ意義は同じである。ただし、こうした後期フェイズになるほど、標準化は以前から試みられているので、参考となるデータは社内的にも揃っているに違いない。それらによって開発プロジェクト全体の計画・統制が適切に行なわれるならば、①多くのプロジェクト案の中から、企業にとって望ましいプロジェクトの選別により信頼性が高まる、②選ばれ、着手されたプロジェクトの巨額の開発費用を計画どおり、もしくはそれ以下に押える可能性が高まる、といった２つのメリットが生じ、プロジェクトの大規模

化に伴って生じる開発フェイズと稼動フェイズとのアンバランスの負担が、企業にとって少しでも軽くなるであろう。

c．訓練され、専門化された要員の不足

システム開発が、従来と同じように個人プレーで行なわれるとすればどういう事態になるかは容易に想像がつく。プロジェクトの大規模化を招いた要因のため、開発担当者は一方ではユーザー側の複雑な業務体系から生ずるニーズの理解に努めねばならず、一方ではそれをコンピュータ・システムの適用で解決するため、ハードウェア・ソフトウェア両面にわたっての最新の技術知識を駆使しなければならない。もし、彼が事態をすべて1人で掌握できるという前提で、彼のなすべき作業のいくつかを、彼の監督下において他人に分担させるとしても、それは彼の能力にさらに加えて、部下に対する指揮・指導能力とリーダーシップとを要求するだけになる。こうしたスーパーマンがどこの企業においても、そこで開発するプロジェクトの数だけ存在するとは考えられないのである。

また、とくに我が国によく見られる状況であるが、こうした才能を持った個人がつくり上げたシステムがいったん導入されたら、その後環境の変化に応じてシステムにとり入れられるべきマイナー・チェンジは、すべて彼の手を借りなければならなくなることである。作られたシステムが余りにも作成者の個人と密着しすぎていて、他人には改善はおろか内容の理解すら許さないというケースも少なくない。しかし一方、わが国の経営環境のもとでは、そのシステム開発者も年功序列制に組み込まれて、管理者としての道を歩きはじめ、同じセクションで役付きとなったり、システム部門外へ計画的にローテーションされたりする。急場しのぎに、別の担当者が、前任者の作ったシステムをいっさい参考とせず、もういちどふり出しにも

どって旧システムの機能にとって代る新システムを、白紙の状態から作り上げようと努力する。こうして、システム作りについてのスキルは、いつまでも特定個人の財産ではあっても、組織の財産として定着するわけではない。

こうしたスーパーマンによるシステム開発にとって代るべきものは、全体として1つの目的に向かってコオーディネートされた、多様なスキルをもつスペシャリストたちのプロジェクト・チームによるシステム開発である。欧米の近代産業の思想的基盤であった分業による能率向上という考え方を、システム開発という知的作業に適用するのである。

ハードウェアやプログラミングといったテクニカルな面に必ずしも精通していないが、マネジメントの能力は保証されているプロジェクトマネジャーが、それぞれ自己の持場での専門知識を持つスペシャリストの集団を動かして大規模システムの開発を効率的に進めうるためには、ＳＬＣの考え方に示唆される、スタンダード体系が必要である。

スタンダード体系は、①プロジェクトの中に現われるフェイズ・タスクのすべてについて、その作業内容を具体的に規定したストラクチュアル・スタンダード、②作業に関する時間とコストのインプット・アウトプットを量的に規定したパフォーマンス・スタンダード、③諸フェイズ相互間、プロジェクト相互間、ユーザーとプロジェクト、などのコミュニケーションを実体的に確保するためのドキュメンテーション・スタンダードの３つから成る。

ストラクチュアル・スタンダードの整備によって、プロジェクトの中に未熟練スタッフを投入して、ＯＪＴ（On the Job Training）的な訓練を施すことが可能になる。スタンダードは彼に、各タスクにおいてとるべき手順を示してくれるので、若干のベテランの指導さえあれば、プロジェクト全体にほとんど迷惑をかけることなく、新人が腕をふるう余地ができるのであ

る。また、このスタンダードの背景となるMethodologyの絶えざる改訂と改善によって、必要人員が必ずしもプロジェクトの規模や数に比例せず、より少な目に押えることができるようになろう。

そうした改善の結果は、パフォーマンス・スタンダードに盛り込まれなければならない。マネジメントはそれによって、どんなスキルを、どの時期に、どれだけ必要かのプランを作成することが可能になり、これは長期の人事計画にも反映されるであろう。

適切なドキュメンテーション・スタンダードの整備は、作成されたシステムが作成者の手を離れて独立(personnel-free)できることになる。これによって可能になる職務のローテーションは、システム担当者にとっても昇進の機会をもたらし、よいインセンティブを与えることになろう。

d. プロジェクト・マネジメントの困難化

前項に述べたごとく、プロジェクトが巨大化すると、1人のプロジェクト・マネジャーがそのプロジェクトの全貌を詳細に至るまで把握し、適切な指示を与えうるとは考えられない。しかし、もし適切なストラクチュアル・スタンダードによってマイルストン(チェックポイント)が設定されていれば、マネジャーが個々の詳細にわたって熟知する必要はなくなってしまう。

開発期間が長期化するにつれて、有効なリソース(資金、時間、人員、設備)の使い方がますます重要になってくるが、このためにはそうしたマイルストンを基準として、パーフォーマンス・スタンダードの利用とメインテナンスを計ることによって、効果的なプランニングが可能になる。

3. システム・メインテナンスとＳＬＣ

ＳＬＣの考え方が、あるシステムの寿命の全体にわたって関心を持ち、各段階での作業を有効ならしむるものであることは先に述べたとおりである。ここでは、第Ⅱ部において触れることのできなかった（研究期間とメンバーの多忙さとの関係で）メインテナンス作業について、ＳＬＣの見方から簡単に触れておくことにする。

ライフ・サイクルという概念が導入されていない場合には、システム開発はとかくシステムの完成・稼動開始をもって終結したと考えられやすい。しかしその後もシステムのパフォーマンスと、そのシステムが置かれた環境の変化とにたえず注目し、必要に応じて手直しを加えるというメインテナンス業務が無視されてはならない。ＳＬＣの概念においては、こうしたメインテナス業務の位置づけが、その重要な場所を占めている。

システム稼動開始後のパフォーマンスをチェックするためには、事前に体系化されたパフォーマンス・スタンダードという評価基準が確立していなければならない。マネジメントが、そうした基準の設定であるプランニングと、基準にてらして実績のチェックを行なうコントロールという２つの役目から成っているという経営学の考え方は、システム・マネジメントやプロジェクト・マネジメントにおいても例外ではない。そして適切なプランニングは、ＳＬＣの考え方によるスタンダードに立脚して行なわれるものである。

かりに、そうした評価基準にてらして、当該システムが不満足であると判定されたら、そこからメインテナンス業務がスタートする。この、メインテナンス業務のマネジメントに当たっては、メインテナンスがある意味でのシステム開発であり、単独のプロジェクトとして考えるべきであること

が強調されている。もちろん、新規開発とちがって、着手時における情報量は豊かで不確実性は少ないし、あくまで手直しであるからそれほど大量のリソースを投入するわけに行かないのは当然である。しかし、ＳＬＣの有用性は、本質的にはメインテナンス・プロジェクトにも、まったく等しくあてはまるのである。

メインテナンス・プロジェクトが新規システムの開発よりも情報量の多い状態でスタートし、少ないリソースで短期間に終結できるためには、そのシステムの当初の開発にあたって、ＳＬＣ概念が採用されていて、同じフレームワークでメインテナンスに取り組むことが肝要である。同じフレームワークを保証するのは、一般的にはシステム開発部門全体にわたって行なわれるべき方法論（Methodology）のスタンダードと、ドキュメンテーションのスタンダードとである。また、個別には、そのスタンダードに則して作成されている、そのシステムの当初開発時のドキュメントの保証である。

とかくわが国の企業においては、当座のシステム作成に心急ぐのあまり、直接にメリットの感じられない標準化されたドキュメントの整備を省略しがちな傾向がある。とくに、いわゆる名人肌で仕事をする専門家に、そうした態度をとる人が多いようである。

またマネジメントの側においても、そうしたスタンダードの押しつけが、創造意欲をそぎ、モティベーションに悪影響を与えるとして、それほど強制しない例がしばしば見られるようである。しかし、そうしたスタンダードの遵守は、システム開発担当者にとっての規律（discipline）であるとして、断固徹底させる習慣をつくりたいものである。直接の効果はすぐに現われないとしても、メインテナンスの段階でその価値が十分に発揮されるであろう。

比較的早くから標準化、文書化の習慣がついている米国企業においても、

第2世代のハードウェアから第3世代への転換にあたって、多くの会社がドキュメンテーションの不備で苦い経験を持ったといわれている。我が国の場合は、第3世代のハードウェア導入を機会に、はじめてトータルといえるシステムを開発しているケースが多いので、まだそれほど表面化していないようであるが、やがて現在の大規模システムが、次の世代のハードウェアへの転換をおこなう際に、ドキュメンテーションの完備・不備の差がはっきり出るに違いない。

メインテナンスは、不確実性が少ないし、前の経験が生かせるという意味ではプランニングが立てやすいが、メインテナンスの程度については、単なる部分的手直しから、大がかりな変更に至るまで、どこが適当かの判断は新システムの領域設定やプロジェクト選択と同様に厄介である。とうぜん、予備的分析の部分に対するウェイトも、新システム開発の場合と同様にかけておくことが必要であろう。

Ⅱ　システム・ライフ・サイクル

Ⅱ　システム・ライフ・サイクル

1.　システム設計に先立っての業務分析と予備設計

コンピュータ白書'72によれば、わが国のコンピュータ設置セット数は１９７２年９月末現在１１,２３７セットに達し、総額１兆円を突破し、５年後には、およそ４倍になることが予想されている。今後、ますますコンピュータが色々な分野において活用されることになろうが、このような時代にあってこそ、コンピュータを如何に有効に、経済的に活用するかについて真剣に考察される必要がある。

従来、この種のレポートにおいてはシステム設計＝コンピュータ利用のシステム設計という形で表現されたものが多く、システムの中でコンピュータが分担すべき範囲が決定されたあとの記述にその多くが費やされている。

この章においては、新しいこころみとして、システムそのものの概念から解きおこし、全体システムの中で、その目的や性格に最もよくマッチした形にするために、どのような判断基準にもとづいて、コンピュータの守備範囲を設定すべきかに重点をおいて述べるつもりである。

1.1　システムとコンピュータ利用のシステムについて

1.1.1　システムとは

システムとは何かについて、ウエブスター辞典は次のように定義している。

System：① a complex unity formed of many often diverse part subject to a common plan or serving

common purpose ② the structure or whole formed by the essential principle or fact of science or branch of knowledge or thought

要するに各々独立した多くの目的を持った要素を、一つの目的のために役立てるように、科学的な客観性に基づいて、方向づけをするものが、システムであるということができる。

したがって、システムの果たす役割りは、ある一つの全体目的を効率よく達成するために、以下に述べる、いろいろな構成要素を如何に効率よく組みたてるかということになる。

1.1.2　システムの範囲と位置づけ

前項においてシステムについての一般的な概念を述べたが「何をもってシステムとするか」については「そのシステムが成り立っている環境」「システムに何が期待されているか」によって、そのシステムの範囲が決定され、位置づけが変わると考える。以下、それぞれのパターンにおいて、システムがどのような態様になるかを考えてみた。

1）システムの対象とする業種

例えば、官公庁、商社、サービス業、製造業を例にとってみても、各々の業種において「システムに期待されるもの」が異なるはずである。極めて大ざっぱな見方をしても、官公庁にあって、システムがカバーする範囲は長期、短期にわたる国の諸施策の立案と実施であろうし、商社であればメーカーとユーザーの中間にあって商品の回転率向上を如何に実現するかということになる。同様にサービス業にあっては顧客を対象にその待時間を如何に短くしてサービス向上に寄与するかということであろうし、また製造業においては設備を中心とした

投資効果の追求であろう。

2） システムの対象とする発展過程

また一つの業種であっても、それが発展期にあるか成熟期にあるかによって、システムの範囲と位置づけはことなるものである。例えば製造業を例にとれば発展期つまり売り手市場にあっては最大生産を志向するシステムが設計されるであろうし、成熟期にあってはいわゆる需給バランスはくずれるわけであって、如何に販路を拡大するかということで販売に関するシステム開発にその重点は移行するはずである。

3） システムの対象とする企業の特殊事情

更に同種の企業であっても、規模の大小、その企業の歴史的な背景、要員事情、立地条件、その後の環境の変化に対する対処の仕方等についても、システムの性格はことなるであろう。

以上、「システムに何を期待するか」ということはケース・バイ・ケースにより異なるけれども、そのケースで決められた役割りを、効率よく果たし得る仕組みをつくることが重要であると考える。

1.1.3 システムとコンピュータ利用のシステム

前項においてシステムの範囲と位置づけについて述べた。そしてシステムとは、その目的ないしは期待されるものを効率よく達成させるための仕組みであることを理解した。

今後はこのシステムをどのように開発していけばよいかについて、その取り組み方を考えてみたい。その過程においては、いままで述べてきた全体システムの中でコンピュータが分担すべきシステムの範囲は、どのような基準で判断すればよいかについて明らかにしたいと思うが、その詳細は後の節にゆずることにする。

ここでは、特にコンピュータを利用したシステムを開発しようとする際の留意点を指摘するにとどめたい。

勿論、コンピュータはシステムを設計する際の有力な手段である。しかし次に示すような欠点をもっている。一つは、相当長期の開発期間を要すること、二つは、硬直性、つまり一度出来上ってしまうとそれの変更の為めに多大な時間と労力を要することである。したがって、この欠点をカバーするために、システムの前提条件を、どのように設定するか、システム・ライフをどれ位の期間に見積もるかが、システム検討時での最重点項目でなければならない。

1.2 システム検討時に考慮すべきシステムの構成要素

1.2.1 構成要素

システム検討に際しては、システムの目的を最も効率よく経済的に実現するために、以下に示す、各構成要素の与件が明確にされなければならない。

1) 設　　備

大型化、連続化、自動化　(無人化)

2) 人　　間

判断の高度化を狙うか、省力化か、業務のルーチン化を狙うか

3) 組　　織

属人的管理か組織的管理か、分業か協業か、責任と権限はどうするか

4) スペース

倉庫、ストックヤード

5）時　　間

判断の緊急性、処理の迅速性

6）情報および情報処理

情報の性格および量、情報処理のスピード、正確度をどの程度保証するか

7）情報処理機器

コンピュータ、端末機器、回線等

8）そ の 他

1.2.2　システム開発に際してのスタンス

1）現状改善か新システム開発型か

現状改善的なスタンスであれば設備、組織等が既に出来上っているわけであるから、コンピュータを導入する場合も、それがなければ動かないということにはならない。この場合には、コンピュータは単に道具として、従来、人間が行なってきた業務を肩代わりするという範囲にとどまるであろう。

これに対し新システム開発型ということであれば、コンピュータでなければ出来ないという部分を中心にして、その他の部分、つまり組織のあり方とか、設備の配置および稼動体制とか、或いは人間の仕事の仕方等々がすべて考えなおされるという形になる。この場合には、コンピュータがむしろ主体制をもちその他の部分を規制することになるわけであり、ここではコンピュータの果たす役割りも、前の現状改善型に較べて当然、変わるはずである。

2）全体システムか部分システムか

この場合の全体システムとは、例えば製造業の、受注、生産、出荷

に関する全機能を、トータル・システムとして設計するケースを指し部分システムとは、その中の一つの機能を対象として、サブシステムを設計する場合をいう。この場合もそれぞれ、システムを設計する際の考え方、および、コンピュータ・システムの果たす役割りはことなってくる。

以上、コンピュータを利用したシステムを開発するに際してのスタンスを述べたが、一概にどちらがよりベターであるということは出来ない。それは、これからシステムを設計しようとする企業が、おかれた状況、歴史的な背景、今後の動向等を勘案して、どのスタンスによるかが決定されるべきものである。しかし、システム・ライフ・サイクルにおける一つのチェックポイントとしては、今から設計、開発しようとするシステムが、どのスタンスに立っているかは明確にしておく必要がある。

1.3 システム検討のための業務分析と予備設計

システムを開発し実施する過程を大きく分類すると次のようになる

- マスタープラン作成のための検討
- マスタープランの作成　(基本計画にもとづく共通目標の設定)

各サブシステムが分担すべき範囲の検討

- 各サブシステム計画のブレークダウン
- 各サブシステムの設計

以下、順を追って各段階で何が行なわれるべきであるかについて、考えてみたい。但し、ここで述べることは、システム検討に際しての、大規模でオーソドックスなものゝ考え方である。しかし、実際に個々のケースにつ

いてシステムを検討する場合は、その規模、アスタンスの相違により、必ずしもこの手順によらなくてよいケースもある。この場合は、それぞれのケースに適合するように以下の内容をアレンジすればよい。

1.3.1　マスタープランの作成のための検討

システムとは、前述したように設備、人間、組織、スペース、時間、情報および情報処理、情報処理機器、その他から構成される。

したがって、この段階では、これらの各要素を組みあわせ、企業の目的を、全体として最も効率よく、経済的に達成させるために、どのようにバランスさせるかの作業が行なわれなければならない。

具体的には「要員はこれだけに抑えたい、したがってコンピュータ・システムとしては、ここまでコントロール出来なくては困る」、とか「設備の能力はこのようになっている、これを効率よく稼動させるためには、コンピュータ・システムとしては、ここまで必要になる」といった形で相互に調整されなければならない。この作業はシステムの最終的な姿を確定するものとして極めて慎重に、繰りかえし、関係箇所との間で討議されなければならない。

この過程を経て、各要素の位置づけ、分担範囲が決定され、目標値が設定されることになる。

特にコンピュータ・システムを志向する場合は次の諸点を十分考慮し、その位置づけと範囲が決定されるべきである。

コンピュータ・システムの目的を明確に

「コンピュータ・システムで何をなすべきか」の目的がはっきりしていなければならない、今後、単なる計算の代行機としての段階を経て、コンピュータを経営の道具として利用する段階にすゝめば、進む程、マネ

ージメントの要請が何であるか、何が期待されているかということが明確に整理されるべきである。

経済的であるかどうか

コンピュータ・システムを導入するについては、システムの開発、ハードウェアの装備、ならびにシステムの運営のために多額の投資が必要である。

したがってコンピュータ・システムを導入し、その位置づけ、分担範囲を決定するについては、定量的、定性的、メリット、デメリットを較量し、ペイするかどうかがその判断基準にたらなければならない。

定量化出来るメリットとは要員の削減効果、設備の稼動率、歩留の向上等であり、定性的メリットとは、品質の向上ーユーザー・サービスの向上、等であるがこれらが予め整理されコンピュータ・システムの事後的な評価の基準とされ、導入に際しての指標とされるべきである。

コンピュータの威力が十分発揮されること

コンピュータのもつ一つの大きな特色はデータを処理する能力がずば抜けて速いということである。したがって、この能力を十分発揮させるためにはこれに見合う、業務が選定されなければならない。

管理サイクルと一致しないようなコンピュータ・システムは意味をなさないし、また、コンピュータ・システムは分担する範囲が管理レベルの不統一を招くような業務であれば、その業務は対象として適当とはいえない。

前提条件が明確に定義されていなければならない

コンピュータはその処理スピードと、正確性においては秀れた特性をもっているが、変化に対する柔軟性、適応性という面では人間の頭脳には

はるかにおよばない。

したがって前提条件が不明確な業務、前提が明確になり得ない業務はコンピュータ・システムの対象にすべきではない。

以上、コンピュータ・システムをどのように位置づけるか、その分担範囲はどういう基準で判断すべきかについて述べたが、特に強調したいことは、あくまでも目的に見合ったシステムであるべきこと、および、コンピュータ・システムの分担範囲は必要最小限度に抑えられるべきであって、バランスとか競争とかいう心理的要素だけで事を処すという態度は戒められるべきであるということである。

1.3.2　マスタープランの作成

大型システテランとは、各要素の分担する範囲と業務量、スケジュール等を明らかにし、それが全体として、斉合性が保たれ、システム全体が実現されることを保証するものである。

1）マスタープラン作成の意義

大型システム（開発期間2～3年、要員数百人年程度以上）の開発プロジェクトを効果的、効率的に推進するためには、大型システムのある部分についての詳細設計に先立って、次の事項をまず検討し確認することが必要である。何故なら大型システムの全体像が描かれないままに、ある部分の詳細設計を行なっても、他の部分とのつながりが保証されていないため、次々と完成してくる他の部分との間をうまく連結するためには、かなりの調整、変更が不可欠となり、多くの部分から構成される大型システムであればある程、その努力は、ほとんどエンドレス・エフォトとなるからである。

検討、確認すべき事項

a　システムについて

○システムの必要性

○システムのカバーする範囲

○システムに期待される機能とシステムの狙い

○外部環境および外部システムとの連係設計の基本思想

○システム・ライフ

b　システムを構成するサブシステムについて

○システムは、どのようなサブシステムから構成されているか

○サブシステムはシステムの中でどのような機能を果たし、そのためどのような設計思想により、どのような基本構成をもつか

○各サブシステムは相互に、どのように連係しているか

c　システムおよびサブシステムの開発スケジュール

○各サブシステムの開発について、その優先順位、時期と開発に必要な資源（要員、コンピュータ等）配分について、総合的に調整された全体スケジュール

○全体スケジュールより、他との斉合性を保証された各サブシステムの開発スケジュール

このような事項を検討、確認するために作成する基本計画を、マスタープランとよぶことが出来る。

2）マスタープランの役割とそのための要件

マスタープランは前段に述べたように、大型システムの開発導入を、全体として効果的、効率的に行なうための基本計画であり、各サブシステムの開発導入はマスタープランに基づいて推進されなければならない。

換言すれば、マスタープランは、それに基づいて各サブシステムの開発が実際に行なわれるためのものでなければならない。そうでなければ単なる文書に終わってしまうものであり、一つのプランとしての価値は、実行を通じて初めて、実現されることを銘記しなければたらない。

マスタープランがプランとしての役割りをはたすためには、実現可能性と規範性の二つの性格を具備することが必要である。

a　実現可能性

次の二点について、十分、考慮されなければならない。

○システムの内容について

・現実のレベルから考えて、実現可能なレベルのものであるかどうか（中間レベルの暫定サブシステムが必要かどうか）

・システムの前提となっている諸条件は整備されているかどうか

・サブシステム間の連係が、マスタープランに基づく、サブシステムの開発段階で、基本的な変更を必要としない程度に十分、確認されているかどうか

・サブシステムの内容が、サブシステムの開発段階における詳細設計のガイドラインとなる程度に、具体的に記述されているかどうか

・システムライフについて十分な考慮がなされているか

○開発のスケジュールについて

・開発の期間、資源（要員の質と量、コンピュータ等）の配分は適当か

・システム開発の前提条件は必要な時点までに準備されるかどう

か

・スケジュールの内容（期間、必要な要員、コンピュータ等の資源）が、開発段階における、詳細スケジュールにより基本的な変更を必要としない程度の正確度をもつかどうか

b　規範性

実現可能性を与えられたマスタープランはサブシステムの開発段階における行動の規範としての機能を果たさなければならない。

サブシステムの開発が、マスタープランから遊離して行なわれるのであれば、マスタープランの目的とする、システム開発の効果的、効率的推進と、インテグレーテッドなインフォメーション、システムの実現は極めて困難となる。従ってサブシステムの開発段階では、何よりもマスタープランを規範として行動するという経営思想と共通認識が必要であり、同時に、各サブシステムの開発、推進が、その内容とスケジュールにおいてマスタープランに沿って、行なわれているかどうかの確認と調整が不可欠である。

3）組織および進め方

システムの規模により、これを検討するチームの規模もまた、大小いろいろな形が考えられる。

先ず、極めて大規模なシステム、例えば、製造業における「販売、生産事務システム」というようなシステムの検討にあたっては、各分野の代表者から構成される社長直属の大プロジェクト・チームが必要である。この場合には、各機能別グループの他に、調整グループを設け、システムの最終目標を達成させるべく、各グループの相互関連において、目標を与え、それの進捗についてコントロールするはたらき

をもたせなければならない。

その他、システムの規模と性格により

- 兼任のチーム
- ユーザーとシステム部門から成る単独チーム
- ユーザー部門の中だけのチーム

というように分けることができる。これらのチームは「どういうシステムが本当に望ましい」を考えながら、現状の調査、問題点の分析、その解決方法を整理、検討していくことになる。

1.3.3 各サブシステムのブレークダウン －DPプランの作成－

前節においてコンピュータ・システムの位置づけ、および分担する範囲を明らかにするまでの手順を述べた。

ここでは、コンピュータ・システム（以下、D．Pシステムと呼ぶ）に限定して、設計に際しての態度、設計手順におけるチェックポイントを考えてみたい。

DPシステムは3つの段階を経て完成する、1つは検討と設計の段階であり、次はその作成であり、最後はその運営である。第1章では、上記のうち、検討と設計の段階を分担する。

第2段階のDPシステムの作成については、第2章および第3章においてまた、第3段階の運営については、第4章、第5章において、その詳細が述べられる。

検討と設計の段階は更に次の手順に分類することが出来る。すなわち(1)現在のシステムを分析すること (2)モデルを作成すること (3)モデルを検討すること (4)新しいDPシステムを提案することである。

- 現在のシステムを分析すること

いかなるシステムの設計にあたっても、現在、何がどのように行なわれているかを十分、理解することなくしてシステムの設計を行なうことは不可能である。現在のシステムを理解するためには、システムの対象となる業務に携わっている人間にヒアリングすることがその主な手段となるが、それの実行にあたっては、あらゆる階層の意見を幅広く収集し、正しい認識が出来るようにしなければならない。ヒアリング技術についてここで述べることは省略するが重要なことは、実際にこのシステムを利用ずる部門から遊離してはならないということである。

○モデルを作成し検討すること

現在のシステムの分析が終われば、これから作成しようとするシステムが「何をしなければならないか」「種々の要件をどの程度満足させなければならないか」が理解出来るはずである。モデルの作成に際してはいくつかの代替案を準備し、各案について経済性、実現可能性、効果性の面から種々の検討が加えられ最適案にしぼられるべきである。

要するにこの段階での仕事は建築家が新しい家を設計するのによくにている。まず依頼主が家族の希望や計画を話すのに耳を傾ける。収入や家族構成と年令を調べ、現在の住まいがどんな趣味、趣向を持っているかを調べる。居間、書斎、台所については特に質問をして念入りに調べる。デザインに対する家族のこのみを知ったら、建築家は希望を予算に合わせる手助けをしなければならない。寝室を独立させるために日よけのついたテラスを犠牲にすべきか空調設備をいれるためには家の景観はあとまわしにしてもよいか、こういったことを考えて予算内におさめ、家族全員に満足のいく設計仕様をきめる。この仕様は重要であり、この正確性は解となる設計の最終的効果を左右するものである。

（IBM Data Processing Techniques ＳＯＰの考え方参照）

○新しいＤＰシステムを提案すること

この段階での仕事はＤＰシステムについて記述することである。この記述はＤＰプランとなり、システムの実現を保証するものとして、その規範とすべくトップの承認を得ると同時に、関連部門にオーソライズされなければならない。

以上の各段階においてＤＰシステムの設計者は質の高い、創造的な努力が要求されるが、その際特に留意すべき点を列挙することにする。

(1) 情報コストの意識は十分か

ＤＰシステムはそれの設計、開発、運営のために膨大な費用のかかるものであることは前述した通りであるが、ＤＰシステムの設計者は「このシステムから得られる情報がはたして有効であるかどうか、十分にペイするものであるかどうか」を常に意識する必要がある。

(2) 企業の目的達成という大目標からはずれていないかどうか

ＤＰシステムは企業の目的を達成するためのもの、ないしは企業の経営方針を決定するについて、それをサポートすべきものである、したがってＤＰシステムの設計者は、システムの目的を正確に理解し、それに必要な情報は何かということを見誤らないよう心掛けなければならない。

(3) システムが企業の体質、特に現在の管理制度に合致しているかどうか

(4) ＤＰシステムが企業におけるトータル・システムとして把えられているかどうか、サブシステム間の相関性は十分考慮されているか

(5) 企業をとりまく環境の変化に対する順応性はあるか

ＤＰシステムは、非常にロジカルなものであるだけに硬直的なものであるが、企業活動は生き物であるだけに、それをコントロールするＤＰシステムには状況の変化に対する可撓性が要求される。ＤＰシステムの設計者はこの矛盾する要求に応えるため、システムの設計に先立っては考えうる仮説をたて、それの前提条件を整備しておかなくてはならない。前提条件のたて方の良否がシステムのライフ・サイクルに与える影響は極めて大きいといわねばならない。

ちなみにＤＰプランとしてうたうべき項目は次のように整理すべきであろう。

① ＤＰシステムの目的

② 当ＤＰシステム案にいたるまでの検討経過

③ ＤＰプランの基本方針

(1) ＤＰシステムの前提条件

(2) ＤＰシステムがコントロールすべき対象範囲

(3) ＤＰシステム設計上のポイント

④ ＤＰシステム設計に要する諸コストおよびメリット

⑤ ＤＰシステム設計、開発に要する人員と質

⑥ ＤＰシステムの設計、開発推進スケジュール

なお、この点につき、日本経営情報開発協会主催の「渡米システムズ・アナリシス専門研修団」（第２回＝１９７１年）のテキスト（Brandon Applied Systems Inc.編）では、これを次のように述べている。

2. EDPシステムの設計

第1章においてコンピュータシステム（以下DPシステムとよぶ）は目的に見合ったものでなければならないこと、およびそのシステムが分担すべき範囲は必要最少限度に抑えられるべきものであることを述べた。第2章では、第1章による検討の結果をうけて、DPシステムの導入が決定されたことを想定し、それを実現するために、DPシステムの詳細設計と、コンピュータを選定するについて、それぞれどこにチェックのポイントがおかれるべきかについて述べることにする。

DPシステムの設計ならびにコンピュータの選定については、これからDPシステムを導入しようとする企業がどのようなスタンスにあるかによって、それぞれチェックポイントのおき方にニュアンスの相違があるはずである。

例えば、DPシステムを新規に導入しようとする場合なのか、或いはある程度の経験と実績に基づいて、さらに拡大、発展させようとする場合なのかバッチシステムを目指すのか、オンラインシステムを志向するのか、特にコンピュータの選定については、コンピュータメーカーの提供する既存のハードウェア、ソフトウェアの技術によりDPシステムの設計が可能な場合とユーザー（DPシステムを導入しようとする企業）がそのシステムに必要なハード、ソフトのメディアを決定し、コンピュータメーカーにその開発を依頼ないしは共同で開発しようとする場合等、色々な条件の違いにより、検討の仕方において力点の置く位置が異なるはずであるが、以下にのべる項目については、どのような条件のもとにあっても最少限、必要と思われるものについて整理したものである。

2.1　詳細予備設計

2.1.1　詳細業務フローの設定

ＤＰシステムの設計にあたって先ず第一になすべきことはフローチャートの作成である。

このフローチャートはＤＰシステムの設計グループ内において、これから開発しようとするＤＰシステムの内容について認識を統一するためのものであり、対外的にはこれから具体化しようとするＤＰシステムについて、たとえ相手方が専門的知識をもたなくても、その概要を理解し、それについての意見を述べるための資料として活用されるものである。

したがって、このフローチャートは次の諸点が明確になっていることが望ましい。

1）企業の全体システムが概観出来、その相関関係が理解出来ること、およびその中でＤＰシステムの分担する範囲が明確に定義されていること。

2）マン、マシンのインターフェースすなわち人間のはたす機能とＤＰシステムの果たす機能が区別されていること。

3）必要なインプットの種類、タイミングが理解出来ること。

4）システムにおける人のながれと物のながれについて斉合性がとれており運用可能なフローとしてまとめられていること。

2.1.2　入出力予備設計

如何に秀れた機能をもつＤＰシステムであっても、インプットデータの正確性が保証されなければ真に有効な情報をうみだすことは出来ない。一方インプットデータの作成、牧拾が最大限に完備していても、それが正しくファイルされ要求に見合うレポートとしてアウトプットされなけ

ればＤＰシステムの目的は達成出来ない。

したがってＤＰシステムの設計における入出力の検討は、最も重要なファクターとして取り扱われなければならない。

この段階でシステムの設計者が行なうべき検討項目を列挙すれば次の通りである。

- ○必要Ｉ／Ｏのアイテムの洗い出し
- ○情報としてのアイテムのまとめ
- ○ＥＤＰＩ／Ｏと人間系Ｉ／Ｏ再検討
- ○Ｉ／Ｏ種類の概要設定
- ○Ｉ／Ｏオペレーション手順の概要
- ○Ｉ／Ｏフォーマットの予備設計
- ○インプットデータチェック条件仮設定
- ○アウトプットデータ編集条件仮設定
- ○Ｉ／Ｏリンク系統予備設定

アウトプットについてはＤＰシステムからアウトプットされたレポートを、いつ、だれが、何のために活用するかの検討が必要である。

- ○アウトプットされたレポートをいつ使うか

企業活動には、その中で行なわれる業務の性格により一定の管理サイクルがある。計画－実行－評価というサイクルの中で、それぞれに必要とされる情報は、長期にわたる情報、期サイクルの情報、マンスリー、ウイークリー、デーリー、時々刻々の情報というように多種多様である。

これらの情報が必要なときに提供出来る体制を整えることがＤＰシステムの使命である。

「タイミングを失したアウトプットは紙くずの価値しか持っていない」

○アウトプットされたレポートを誰が使うのか

企業は色々な階層の人間から成りたっており、ＤＰシステムに期待するアウトプットは一様ではない、トップの経営者、中間管理者、実務の担当者は、それぞれ自分の分担する管理スパンの範囲において最も有効な、使いやすいレポートを欲するはずである。したがってアウトプットの設計にあたっては、誰が使い、どのレベルの情報を欲しているのか見きわめる必要がある。

○アウトプットされたレポートが何に使われるか

アウトプットが何のために使われるかによってもアウトプットの性格はことなるはずである。例えばプランニングのために過去から将来にいたる傾向をつかみたい場合、計画と実績を対比したい場合、等、それぞれの用途により、データがカバーしなければならない範囲、密度はことなるはずである。

以上、必要とされるアウトプットの質と量の洗い出しが終われば、何を誰がどのようなタイミングで何によりインプットすべきであるか、データファイルのメディアとしては磁気テープがよいのか、ディスクか又はドラムがよいのか、その他で秀れたものがあるのかの検討が可能となり、アウトプットのメディアとして何が最も効率的であるかの予備設計が出来る。

特にコンピュータの負荷を計算するについては、インプットデータの正確性を保証するためのチェックに要する部分が大きいと思われるので、以下、これに関連してインプットデータの正確を期すためにどのような方法が考えられるかについて述べることにする。

1）インプットの原始帳票は単純化しなければならない

インプットに必要な原始帳票の種類は必要最少限度に抑えられるべきであり、データのソースは一元化されるよう整理されなければならない。また帳票のレイアウトは、なるべくシンプルにし、データの記入、タイプミス、パンチミスの発生を未然に防ぐよう工夫されなければならない。

2）データのコード化

コードとは数字あるいは英字で各種の名前を出来るだけ短い一定のケタ数により、系統的に簡略化した符号のことである。

データをコード化することは、単にデータをコンパクトにするという意味の他に次のような機能をもっている。すなわち各業務の体系や関連ないしは管理の意図を表現するためのものである。したがって、それの、検討、設定にあたっては　①全体的な共通性をもち体系的であること　②コードの追加等が出来るようにコードの体系に弾力性をもたせること等の配慮が必要である。

3）コンピュータによるチェック

原始データは人間が採取、インプットするものである限りエラーをゼロにすることは不可能に近く、この種のデータがコンピュータの負荷におよぼす影響、アウトプットの精度に与える影響は極めて大きい。したがってエラーデータについては処理に入る前にコンピュータによるチェックとリジェクトが必要である。

コンピュータによるチェックの方法としては次のものが考えられる。

a　テーブルチェック

予めインプットの可能性のあるコードのテーブルを準備しておき、

それにマッチングしないデータがインプットされたときはリジェクトする方法。

b　関連チェック

メインになるデータに対して、ロジック的に矛盾するサブデータがインプットされた場合リジェクトする方法。

c　アイテムチェック

データレングス、Alpha Numericチェック等。

4）オートコーディング

マニュアルによるインプットを避けるため、オートコーディングの手法を採用することもエラーを削減する一つの方法である。

オートコーディングとは、ジェネレートすべきデータを論理に基づいて準備しておきその条件にマッチしたときのみ該当するデータを付加する方法である。

5）インプット機器の活用

通常、インプットの媒体としては、カード、紙テープが使用されるが、これによればパンチャー、タイピストの作業を要し、ミスの防止、省力化という見方からも問題なしとは言えない。これをカバーするものとしてOCR（Optical Character Reader）MICR（原始帳票上に磁気文字を印字し、それを読みとり装置が読みとって演算処理を行なうもの）音声によるインプット手段等が開発されている。これらの手段は、これから開発しようとするシステムが対象とする業務にfitする限り、積極的に活用すべきである。

2.1.3　ファイルの予備設計

DPシステムにおけるファイルの設計はシステムの性能を決定づける

といっても過言ではない。ファイルがシステムそのものの処理方式に大きな制約を与えるものである以上、それの設計にあたっては、正に「これから開発しようとするシステムに何が期待されているのか」ということが検討のベースにならなければいけない。

したがって以下に示す各項目について、色々な角度から検討が加えられ、最良の仕組みがあみ出されなければならない。

① そのファイルを何の目的のために使うか

② そのファイルはどういう使われ方をするか

③ そのファイルの使われる頻度はどれ位か

したがって、

④ ファイルの内容として、質的にどのようなデータをファイルしておけばよいか量的にみて、どれ位の幅をカバーし、ボリュームはどれ位になるか

⑤ 上記の機能を満足するためにどのような機器が最適であるか

以上がファイル設計上のチェックポイントであると思われる。

ＤＰシステムのファイルは、なるべくコンパクトであることが望ましい、何故なら迅速なレスポンス、コンピュータにかかる負荷を考えた場合の経済性から見てはるかに有利であるからである。

一方、必要な情報を必要なときにアウトプットしたいという要請に１００％応えようとすれば、カバーする情報の種類、範囲は広くならざるを得ない。

これら相反するメリット、デメリットを如何に調整し、その中で最良な方法を如何にして見つけ出すかが、この段階における、システム設計者の役目である。

ファイルの予備設計の段階でシステム設計者が行なわなければならない主なイベントを整理すれば次の通りである。

- ＥＤＰに記録させるべき情報の評価と再検討
- ファイルアイテムの仮設定
- データ量の推定
- メディアの決定
- ファイルの構造
- キーコードの設定
- アクセス方式の標準化と割りつけ方式
- 情報のライフサイクルとファイルの再組織化方式
- Historical File, Recap File
- レコードの予備設計

以上に示した各項目は最終的にファイルをどのように設定するか、コンピュータの構成をどのように決めるかについて欠かせない検討テーマである。一方、これらの項目は以降に述べる処理すべきデータ量、そのファイルが使われる頻度、使われ方、ないしはオペレーショナルな面等と密接な関連をもっている。したがって、ＤＰシステム設計におけるファイルの設計は、決して単独に行なわれる性格のものではなく、これら関係する他のイベントの検討と歩調を合わせつゝ、スパイラル状のステップをふみながら最後の決定に導かれるものであることはいうまでもない。

以下、順を追って説明を加えたい。

1）ＥＤＰに記録させるべき情報の評価と再検討

前項において必要な情報の洗い出しが終わっていることに基づき、こ

こではその情報がＥＤＰに記録させるべきものであるかどうかの評価がなされなければならない。コンピュータは大量のデータを迅速に処理し、情報を生みだすこと、および少量のデータであっても、それの処理について、極めて正確なこたえが要求されるものを扱かうという点において人力の及ばない能力を発揮する。したがってその評価の基準は次のように考えてよいと思われる。

①　量的にみてそれがコンピュータ処理にふさわしいものかどうか。

②　質的にみてコンピュータによる処理が要求されるものかどうか。

であるが、ここで質的にというのは、イメージとして「リピート性に富む業務を繰り返し処理するために使用されるマスターファイル」、「コンピュータによる厳密なチェックを必要とし、ハンドによるミスが重大な損害をひきおこすようなデータ」等をさす。

2）入出力の条件と記録情報のグルーピング

上記の検討に引き続き記録情報をグルーピングする作業が必要である。記録情報のグルーピングとは、これから開発しようとするシステムの性格すなわち、オペレーションのサイクル、アクセス量、アクセス方式等を勘案し、ファイルの設計について　①どの情報と、どの情報は同一のファイルに持つべきであるか　②ファイル間にデータの重複はないかの検討ならびにチェックを行なうことである。

入出力の条件とはいつ、誰が何をインプットし、いつ、誰が何のために、そのアウトプットを利用するかの確認を行なうことである。

3）ファイルアイテムの仮設定、データ量の推定、メディアの決定

記録情報のグルーピングが終われば、ファイルアイテムの仮設定が可能となる。ファイルアイテムの仮設定が出来れば各アイテムのサイズ

により、１レコードの長さが推定される。１レコードの長さにデータ数を掛ければ、ファイルのボリュームが推量されるから、その量に見合うメディアとして、何が適当であるかについて色々な代替案をうみだすことが出来る。最終的な決定にあたってはアクセス方式、その他の制約条件を加味しなければならないが、メディアを決定するについて、データの量という問題は重要なファクターの一つである。尚、１レコードの長さをfixされた形にもつかVariableの形にしてデータボリュームの節約をはかるかは、そこで行なわれる業務の性格により判断されなければならない。

データ量の推定の仕方については後節で述べることにする。

4）ファイルの構造、キーコードの設定、アクセス方式の標準化と割り付け方式

ファイルアイテム、レコードレングス、データボリュームの検討が終り、これにマッチするメディアの選定が終れば次はファイルの構造を如何に設計するか、すなわち、データの配列をどのように工夫し、処理のながれとの関連でrandom accessfileにするかsequencial fileにするかの検討を行ない、それに見合うメディアを選択しなければならない。特にランダムに処理しようとする場合は、サーチキーの決定に際し十分な配慮が必要である。

１ファイルについては、１アクセスの方式に統一することが望ましい、オペレーション上の混乱、ならびにプログラム上の繁雑さを防ぐためである。

アクセス方式の標準化については、記録情報のグルーピングの段階で予め考慮される必要があろう。

5）情報のライフサイクルとファイルの再組織化方式

ファイルには適時Updateされ累積されるものと、内容のメインテナンスにとどまるもの〻二種類に区分出来る。特に前者については、ある時点でファイルの再組織化をしなければ、コンピュータにかゝる負荷はデータボリュームの増加に比例して大きくならざるを得ない。ファイルの再組織化を行なってよい時点は、データないしは情報のライフサイクルの終了した時点すなわちそのデータ情報へのアクセスが完全になくなる時点であるが、この線をどこに求めるかは、そのデータを使用する部門の価値判断とコンピュータの負荷増によるコスト高という利害の調和点をどこにおくかによって決定される。

一定のライフサイクルを経過したデータは再組織化されたのち、コンパクトな形にされ、事後の統計、分析用データとして保存される。保存用のメディアとしてスペースの許すかぎり、磁気テープを用いるのが一般的である。

6）Historical File, Recap File

historical file とは例えば当日のアウトプットが翌日のインプットになるという形態のファイルを指す、前述の適時Updateされ累積されるファイルは、この形態に属するものが多い。このファイルは、当日の処理が加わったという点で前日までのファイルとはその内容をことにする。このことはＤＰシステムの処理フローを設定する際の一つの与件であり、トラブルに際して、それのリカバーの手順を考える上で必要な検討テーマである。

2.1.4　コンピュータプロセスの予備設計

前項まではある業務を処理するについてコンピュータが分担すべき部

分を明確化し、それのI/θについて概要を決め、それがどのようなメディアによって処理されるべきかについて考えてきた。ここではそれらをうけてコンピュータプロセスの予備設計を行なう。

ここで検討すべき項目は

○ジョブの流れの分析

○マルチプロセス

○オンラインジョブ設計、オフラインジョブ設計

○コントロール部分と業務ロジック

○各種パラメータのテーブル化　　である。

1）ジョブの流れの分析

ジョブの流れはその業務の時間的制約、すなわち、どういうインプットがどういうタイミングで行なわれ、いつアウトプットが必要かということから決定されなければならない。コンピュータはある時間帯を設定した場合、おのずとその限界があり、限られた能力の中で如何に効率よく処理するかが、ここで検討すべき問題である。

業務の時間的制約は、そのシステムが何を目的としているか、それに応えるためにどの程度のシステムにするかによって異なるけれども、発生するレコード単位に即時にレスポンスが期待されるもの、例えば座席のリザーベーション、プロダクションプロセスコントロールというような業務はオンラインリアルタイムシステムにより処理しなければ応えられないものであり大量のデータを対象とし、処理のサイクルがそれ程シビアーでないものについてはバッチシステムによる処理に適しているといえよう。いずれにしろ要求されるタイミングに合わせてアウトプットを行なうためにはジョブを如何にうまく組み合わせるか

についての熟慮が必要になる。

ちなみにオンラインリアルタイムシステムを志向するについては設計、開発運用すべての点において事前に十分な検討がなされなければならない。何故ならば一度そのシステムを作り上げると容易に変更することが出来ないこと、それの開発、運営に要するコストは極めて大きく長期的な見通しが必要とされること等の理由による。オンラインリアルタイムシステムを導入するについてはコンピュータの利用という面でかなりの経験が要求される所以でもある。

尚、実際の開発に際しては　①トラフイック量とＣＰＵとの関係によるレスポンスタイムの測定　②各端末とコンピュータを結ぶ回線の問題　③オペレーション体制等、通常のバッチシステムの設計とは比較にならない広範でしかも質の高い検討が必要とされるところである。さて、ジョブのながれの分析とは要求されるタイミングに合わせてアウトプットを行なうために、ジョブ別にどういう順番で処理していくか、ジョブストリームとジョブストリームとの構成をどうするかということの検討を行なうことである。

これらの検討結果をまとめたものがＤＰシステムのフローチャートであり、これにより、プログラムの規模、構成、処理すべき内容等について、その概要を把握することが出来ることになる。

2）マルチプロセス

マルチプロセスとは同一の処理装置を共用し、２つ以上のジョブを同時に並行して処理することをいう。マルチプロセスの利点は、コンピュータシステムの資源（記憶装置、チャンネル、制御装置、入出力装置）についてより高い利用可能性を追求出来ること、資源の使用に

融通性をもたすことが出来ることおよび複数のジョブを同時に実行することによってスループットを上げることが出来ることであり、ジョブの流れを分析するについて効率化をはかるために考えられたコンピュータの一つの使い方である。

但し、これの実際の活用については常に優先順位にしたがってジョブが処理されるということから互いにＣＰＵ占有率の高いジョブを並行させることは得策ではなく、CPU占有率の高いジョブとその他の資源を多く活かせるジョブをうまく組み合わせることによって、はじめて、その威力を発揮させ得るものである。

3）オンラインジョブ設計、オフラインジョブ設計

これもジョブの流れを効率化するために考慮しなければならない、ジョブの分け方の一つである。オンラインジョブとは前段階のジョブが終わらなければ、次工程のジョブが開始出来ないという性格のものを指し、オフラインジョブとは、例えばあるデータをテーブル化したファイルのメインテナンスとはある帳票を所定のフォーマットに合わせてプリントするというような業務をさし、各々が独立したジョブとして完結し得る性格のものをいう。

この両者を如何に組合せて、タイムラグを小さくするかがジョブの流れを分析する際の一つのポイントである。

4）コントロール部分と業務ロジック

特にオンラインシステムの場合は相当多数のモジュールが寄り集まって一つのシステムを構成しており、その間にいろいろなコントロール機能が錯そうしている場合が多い。このようなケースにおいて個々のモジュールにすべてのコントロール機能、例えば通信回線のコントロ

ールとか、各種端末からインプットされるメッセージ及びデータのチェック、振分けといった機能を分担させることは得策ではない。したがって、これらのコントロール部分を一括して設計し、その下に個別のアプリケーションをぶらさげる形をとる方が、はるかに効率的である。

これにより、各アプリケーションプログラムはどの範囲のスペックをカバーすべきかが明確になる。

コントロール部分と純粋に業務ロジックから成りたつ部分を分離して設計することは、ジョブの流れをスムーズに行ない、開発ならびに運用面で効率化をはかるための一方法である。

5）各種パラメータのテーブル化

パラメータとは作業をどういう手順で行なうかを指示するためのものである。したがってジョブの流れが複雑になればなる程、これをコントロールするパラメータの管理は容易ではない。したがってこの繁雑さを避け、スムーズなジョブの流れを保証するため、パラメータのテーブル化をおこない、そのテーブルを個々のジョブステップが参照出来る形にして、その間のデータの受渡しを可能にしようとするものである。

2.2　定量調査と新システムデータ量の見積

2.2.1　ＥＤＰ設計と定量調査

ＤＰシステムは必要な情報を必要なタイミングでアウトプット出来るものでなければ意味がないことは繰り返し述べてきたところである。しかし如何に高性能、大容量のコンピュータであっても（勿論、経済性を

考慮した上でのことはあるが）その中で処理し得る量にはおのずと限界のあることはいうまでもない。したがってこれから開発しようとするシステムが、どのようなインプットを行ない、どのように処理し、どんなアウトプットを行なうかの定量的な裏づけをすることはＤＰシステムの設計上、最も基本的な検討項目である。

ここで検討すべきイベントは

○ＥＤＰ設計にあたえるデータの量

○新システムにおけるデータ量の予測

○システムピークの考え方　であるが以下、順に説明を加えたい。

1）ＥＤＰ設計にあたえるデータの量

データ量を測定するについては色々な側面からこれを検討しなければならない。すなわち　①ソースデータの量　②コンピュータの中で実際に処理される量　③アウトプットの量である。

① のソースデータの量については、バッチシステムの場合、インプット要員、及びインプットのタイミングに影響するところが大きく、オンラインの場合はそれがトラフィック量になる。

② については、コンピュータの能力をどれだけに見積るかを決定するについて欠かせない要因であり

③ についてはアウトプットのタイミング、管理レベルとの兼ねあいで見直される必要のあるものである。

また、業務の性格上、量的には多いが処理時間に余裕のあるもの、量的には少ないけれども、ある一定の時間内で処理すべく非常にタイミングのシビアーな業務等色々なパターンがあり、それぞれの業務に見合った定量化の検討が必要である。

2）新システムにおけるデータ量の予測

これから開発しようとするシステムについてデータ量を予測するについては、先ず3つのチェックポイントを設けるべきである、すなわち　①現在のデータ量を把握し、それがどのように処理され、どのようなタイミングでアウトプットされているかについての考察を行ない、それがピーク時においてコンピュータ能力の範囲内にあるかどうか、②過去の実績から、その傾向を把握して、データ量の分布ならびに伸び率を見定めること、　③業務の性格が変化することによってコンピュータにかかる負荷増分を見積ること　　である。

先ず第一に、現在のデータ量を把握するについては、事務量調査の手法として実績記録法、自己記入法、ワークサンプリング法等いろいろな手法が紹介されたが、要するに事務担当者がオペレーションの遂行に費やした時間、仕事の出来高、仕事のサイクル稼動率、余裕率等をデータとして集め、それを分析することによって、インプットの量処理量を把握することを目的としたものである。尚システムピークの考え方については後述する。

第二に、データ量の分布および伸び率を見定めることは、システムの規模および、ライフサイクルを設定するについて極めて重要なキー項目である。

なぜならデータ量の分布、伸びの見積りのあまさがしばしばコンピュータ能力をオーバーフローする事態をひき起し、さっそく長い期間多くの費用をかけて開発したシステムの寿命を縮めることになりかねないからである。

データ量の分布については勿論、業務の性格によりことなることは

いうまでもないが、その他に景気の変動によって処理すべき量に与える影響、季節的変動、または例外要素などをどの程度システムの中にとり込むかの問題である。

データ量の伸びについては外的な要因に影響されるところが大きい。例えば鉄鋼メーカーの販売、生産事務システムというようなものについて考えた場合、粗鋼の生産規模が1億2千万屯のベースとそれが1億5千万屯になった時点での注文件数の処理量は当然ことなるはずであって、これらの要因をどのように勘案するかがシステムライフを設定する際には重要な問題である。

第三に、特にバッチシステムについて言えることであるが、システムを使い込んでいく過程において、不備な点の修正、性能アップのためのメインテナンス又は管理レポートのあらたな要求等、処理量の増加という形で、コンピュータにかかる負荷の増分を見込んでおく必要があろうと思われる。

以上3点は、DPシステムの設計において、どのような規模にするか、システムのライフサイクルをどのように設定するかについて、データ量という側面からみた場合の欠かせない検討項目である。

尚、システムライフサイクルに影響を与えるものとしては、システムそのものゝ基本構想に反するような社会事情の変革があるが、ここではそれについて触れるところではない。

3) システムピークの考え方

DPシステムは、一時的にもせよコンピュータ能力以上のものを処理することは不可能である。したがってコンピュータシステムを導入するについては最も経済的にかつ最も効率よく稼動させるためにピー

クをどのように処理するかが重要な検討課題である。

オンラインリアルタイムシステムを実施するについては、ある時点におけるトラフィック量のピークがシステムの規模Ｍａｘ能力を規制することになろうが、バッチシステムにおいては必ずしも、これらすべてをコンピュータシステムでカバーしなければならないというものではない。例えば１ケ月の中で４回のピークがあるとすれば、これをＤＰシステムでカバーするか、運用面で平準化する方法を採るかは、システムユーザー部門とシステム設計者との間で、十分話し合いが行なわれるべきである。

また、１日、１週間、１ケ月、各サイクルの中で、それぞれどこにピークがあるかをおさえることは、他の業務との関係で、コンピュータシステムの稼動スケジューリングを行なう際に必要な情報である。

2.2.2　入出力データ量

前項ＥＤＰ設計と定量調査においてはデータ量の把握という点について主にシステムライフサイクルという見方から、そのチェックポイントを考えてみたが、ここでは具体的に或るシステムをユーザー部門との間で設点するについて要求に見合ったタイミングでアウトプットを提供するために、ＤＰシステム設計に際して、どこに焦点をおいて検討すればよいかを述べることにする。

先ず、第一は入出力のデータ量の把握であり、第二は内部処理を行なう際のファイルの構成の仕方であり、第三は処理時間すなわちＣＰＵの能力の問題である。第二、第三については以下に項を分けて説明する。

入出力データ量は業務の性格により、ことなることは前述した。例えば人事管理システムというものを例にとれば、ソースデータとしては膨

大な量が予想されるが実際に処理し、アウトプットする量としてはある資格、条件に該当したもののみに限定されるということが出来る。また前例の鉄鋼メーカーの販売、生産事務システムというような性格の業務は販売部門が当日、受付ける注文内容のすべてがチェックおよびスクリーニング等の処理の対象になるものである。更にオンラインによるプロダクションプロセスコントロールシステムにおいては、物の流れのスピード、量、工程数等により、それぞれから発生するデータ量、メッセージ量は決まってくる。

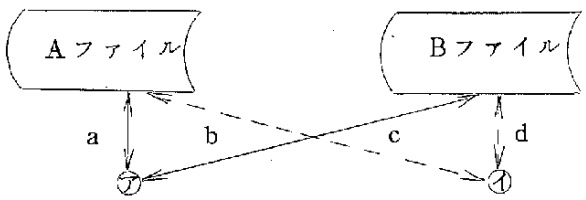
このように、各業務の性格により、インプットされたものすべてを処理するのか、例外的に特定のデータのみを処理すればよいのか、目的とするところにしたがって入出力のデータ量はことなるはずである。

ＤＰシステムの設計にあたっては先ず、これらの量の把握が的確でなければならない。

2.2.3　ファイルアクセス量

ここでの検討テーマは、上記データを処理するについて、実際に使われるファイルを如何に分散してもてばアイドルタイムを最少限におさえることが出来るか、ワークファイル、コンパイラーといったファイル、どのように割りつければハンドリングを少なくすることが出来るかという問題である。

例えば、下図のようにＡ．Ｂ．２つのファイル構成になっており、ア、イ、２つのジョブからアクセスされる形を想定してみよう。



このような形の場合、aのアクセスとcのアクセスが同時に競合するようなファイルの構成は望ましくない、何故なら、aのアクセスタイムの間だけcのアクセスは待たされるからである。

このような現象を防ぐためには、ジョブ別ファイルのアクセス量、ファイル別アクセス量とコンテンション、それぞれのピーク量等、すべての検討にまなって、スループットを早く行なうための工夫が必要である。

なお、これらの検討の結果、欠陥が発見されたときは、ファイル設計にまでフィードバックされなければならない。

2.2.4　プロセス量

入出力の量が決まり、ファイルのアクセス量が決まれば、各ジョブのステップ数によって、処理に要する時間の推定が可能であるが、この時間は、ＣＰＵの能力如何にかかっている。システムに要求されるタイミングに応えるために、現有能力で不足するのであれば、要求にマッチするよう、グレートの高い、ＣＰＵの発注が必要となろう。

3. システム詳細設計とプログラミング

前段階で、ＤＰシステムにコンピュータの使用がきまり、機種の選定が行なわれているので、その機種に合わせ、コーディングが可能なように最終的詳細設計を、システム・アナリストもしくはプログラマが行なう。その際に、詳細設計に必要な情報がすべて用意されているか確認し、サブシステム間のインターフェースに特に注意する。そして、ＤＰシステムの設計担当者、もしくはユーザー部門の担当者承認を得る。コーディングに入る前に、ユーザーよりの変更とか、新しい要求等は受けつけない。つまり凍結する、ということをする必要があると思う。そうしないと、詳細設計の終わり、かつコーディングが始まった段階で、変更してくれなどということが出てくると、開発の予定が狂ってしまう。もし要求が出てきた場合には、それはためておいて、初期の設計のものが完成し、動き出してしばらくしてから、変更を考えるか、あるいは期日が遅れるという条件つきでの変更要求の受けつけは考慮する。

詳細設計が終わるとプログラミングに入るわけだが、プログラミングの仕事としては次のようなものがある。

- ○論理の分析・設計
- ○コーディング
- ○机の上の検査
- ○テスト・データの作成
- ○アセンブリまたはコンパイレーションとテスト
- ○オペレーションのための作成
- ○最終のドキュメンテーション

3.1 詳細設計

3.1.1 ファイルの詳細な内容と構成

大体前段階で、どのようなファイルが必要になり、かつそのファイルはどのような媒体に置くか、というようなことがほぼきめられているわけだが、この段階でいま一度、テープに置くファイル、あるいはディスク、もしくはドラムに置くファイルについて、きちんとしたレコードの編成と、それからレイアウトを作成する。その結果、ファイルの仕様書をつくるわけだが、その内容には次の項目を含む。

アプリケーション名、ファイル名、ファイルの識別(identification)、レコード名、ファイルが使われるプログラムのリスト、ファイルをつくり出すプログラム、レコードの長さ、ブロック化因数、保存周期、編成の型、記録モード（テープ）、フィールドのレイアウト（フィールド名フィールドの略号名、フィールドの長さ、小数点の位置、データの型、最大最小値）。

このようにして仕様書ができたら、次に、各プログラムに使われるファイルと各ファイルが、使われるプログラムの前後対照表を作る。これによって、ファイル名とかファイルの識別のきめ方、およびレイアウトへの変更等が統制できる。

○各テープ、およびディスクについては、標準のラベルを使用するようにする。テープのラベルについては、日本工業規格が存在するので、それも参考にするとよい。

3.2.1 入出力の書式

カードとかＯＣＲの用紙等、入力のレイアウトを、標準の用紙を使い作成する。その内容としては次のことを含む。

アプリケーション名、プログラム名、予想枚数、フィールドのレイアウト（フィールド名、フィールドの大きさ、データの型、位置）等。

印刷出力（プリントアウト）のレイアウトを定め、標準の用紙に記入する。各フィールドに対して、フィールド名と例を記入し、かつフィールドの区切り（長さ）が明確にわかるようにする。

各印刷出力は、それを識別する適切な見出しをつける。見出しに、たとえば社名とか、報告書名とか、報告書番号、ページ番号等が含まれる。

そのほか、コンピュータにかける入力出の伝票用紙を設計するとか、あるいは印刷の用紙を特別に設計するといったような仕事、いわゆるフォーム・デザインの仕事がかなりここで必要になることがあるし、また、フォーム・デザインばかりを専門に担当している人も存在している。

3.1.3　機器や回線に関する要求の再確認

一応、どのようなファイルが必要で、そしてまた入出力についてのレイアウトとか、あるいはフォーマットというものがきまったわけで、それに基き、一体どのような装置が必要かということをここでもう1回確認することが望ましい。まず、処理に必要な入出力装置とか補助記憶装置の場合にはどのぐらいのスペースが必要とか、あるいはアクセスタイムがどのくらいなきゃいけないというようなことの検討も含まれる。それから処理するための主記憶装置の大きさ、特殊機能の必要性の確認というようなものを行ない、さらに、現在ついてないような装置で必要なものがあればここで要求する、ということが出てくる。また、ハードウェアに伴ってソフトウェアのほうも、特別な要求をここで出すということもあり得るかと思う。

カードでインプットするような場合は問題ないが、カードでなく特別

な装置を使ってデータを入力する。あるいは入力データを準備するといったような場合には、それに必要な機器をここで再検討しておく必要があり、もし特別な機能を要する場合にはそれに関する設計とか、あるいは性能の検討というようなことが必要になる。

オンライン・システムで端末を使ってるような場合には、端末装置についての同様な作業が要求される。特に銀行の端末だとか、あるいは座席予約の端末とか、特殊用途の端末の場合、あるいはいわゆるコントロール・システムのような場合で化学工場におけるプラントのオートメーションに使うような計測機器を端末につないでいるような場合だと、そういうもののさらに詳細な検討、場合によっては、検討をし、それを特別につくってもらうというようなことが必要になってくる。

さらに、オンライン・システムの場合だと、前の節で検討したような入力を、またピーク時だとかその他の要素も加味して最も経済的な、効率のいい回線網はどんなものかということを、ここらでもう1回検討してみることが望ましいわけで、そのためにシミュレーションを行なうとか、あるいは解析的手法を使うことがある。

このように個々の装置だとか、あるいは回線網についての検討が終わったあとで、大きなシステムとか、あるいは複雑な処理をするシステムについては、ここらあたりで全体のシステムの評価、トータルなスループットとか単位あたりどうなるとか、あるいは付属している機器はどのように効率的に使われるかというようなことを、これもまた解析的手法とか、あるいはシミュレーションとか、そういうものを使って検討することが望ましいとされている。

効率の検討の要素、チェックポイントをいれた方がいいのではないか、

またこれは誰が行うか？

3.1.4　プログラムの論理設計

プログラミングというのは、大体、業務分析を行ない、現在どういう問題があるかということを識別したあとで、その問題をどうやって解訳するかという解決案をつくり出すわけだが、その解を今度は実際にコンピュータでやってる場合にはどういうふうな手順で、どういう論理でやったらいいかということをきめるのが、プログラムの論理設計であり、さらに、論理設計したものを機械のことばになおしていく、いわゆる翻訳の作業がプログラミングであるということがいえると思う。

○プログラムの論理設計には次の２つのやり方がある。

①　まず主要な流れ（メイン・フロー）を先に書いて、それから枝分かれする部分を書く。

②　各ルーチンが独分にプログラムされ、コンパイルされテストできるように、論理的な部分、すなわちルーチン単位に問題の解を分解する　いわゆる積木方式のプログラム（モデュラー・プログラム）。

○ここでは、積木方式のプログラムについて述べる。使用するプログラミング言語によって、論理設計の詳しさのレベルが異ってくる。ここでは、アセンブラ語を使用した場合について述べることにする。公正準言語を使った場合には、以下に述べるような形ではなく、もっと簡単にすることができる。

○積木方式のプログラミングというのは、主流（メインライン）と呼ばれる１つのルーチンにより制御されるプログラミングの組み立て方である。その場合に、作業の割り当ての決定は、主流ルーチンにより行なわれ、それは処理ルーチンの機能に関知しない。何らかの理由で、

あるルーチンが修正もしくは除かれる場合には、他の処理ルーチンには何らの影響をおよぼさない。そして新しい処理プログラムを追加するには、主流プログラムの中の分岐の行き先を1つ増してから、その処理プログラムを入れればよいわけで、要するに追加というのは簡単になる。この方式の利点としては、次の3つがある。

○二重のコーディングが避けられる。

○多くのモジュール化ができる。

○システム保守が容易である。

○積木方式をやってく上でのおもな設計基準というのは、

○わかりやすさ

○修正しやすさ

○プログラム組立ての標準化

である。

○このようなことを達成するには、次のような手順をとる。

1）モジュールの流れ図作成 － ルーチンの主要構成要素と構造の全体図、それから必要に応じて詳細図を書く。

2）各ルーチンの詳細な記述 － ルーチンの目的、ルーチンによって処理されるデータ、およびプログラム論理の各ステップの説明を書く。

3）プログラミング上の規則 － 標準のラベルの使用、標準のドキュメンテーション手法に従う。

○モジュールの設計は次のように行なう。

1）プログラム・パラメータの決定

2）要求、要素、機能を列挙する

3）処理ルーチンの論理的順序の決定、主流の設計。

4）１ページ大の全体図の作成

5）組織図のような多くのモジュールよりなる図の作成

○主流ルーチンは、

○ハウスキーピング

○入力レコード

○処　　理

○出力レコード

○ジョブの終了（エンド・オブ・ジョブ）　の部分より構成される。

○主流ルーチンの作成に関しては次のような約束がある。

○主流ルーチンは、同じレベルの他の処理ルーチンにデータの流れを向けない。

○必要なら、処理ルーチンにより処理のより低いレベルに入る。

○複数の処理ルーチンに共通な入出力機能は主流ルーチンにより制御される。

○すべての共通な区域は主ルーチンの一部として定義される。

○処理ルーチンに関する約束は次のとおりである。

○プログラムの各論理セグメントの別個の処理ルーチン。

○各処理ルーチンがそれ自身で完全である　－　そのルーチンに限って必要な区域は自分の中に含んでいる。

○セグメントの外の決定がセグメント内の処理を決めず、セグメント内の決定は、セグメント外の処理を決めない。

○各ルーチンは閉じたサブルーチン、ルーチンの入出力は、特定の前後のセグメントに決して従属しない。

○ただ1つの処理ルーチンにしか影響しない入出力機能は、そのルーチンで行なってもよい。

○各セグメントは、他のセグメントの介入を避けるため自分自身のハウスキーピングを含む。

○上位モジュールに渡す処理結果は、上位モジュール内の区域かレジスタに置く。

モジュールの大きさとしては、わかりやすく短時間で書け、完全にテストできる程度の大きさが望ましい。そして大きな高尚なモジュールをつくろうとするよりも、むしろプログラムの構造の計画を練るのにより多くの時間を費したほうがよい。モジュールの大きさの例として、たとえばアセンブラでは200命令、COBOLでは50ステートメントぐらいとするのが望ましい。

リアル・タイム・システムでは次のことに注意する。

○複数個のトランザクションを同時に処理することがある。あとから入ってきたトランザクションが前のが正しく分岐する前にセットしたスイッチをこわさないようにする。そのためには、再入可能コードの形をとらなければならない。

○プログラム間のリンケージ（作業区域のアドレス、種々の状態表示の受け渡し）。

○コミュニケーション・プログラムと処理プログラム間のリンケージ。

○入出力論理が複雑 ― 制御キャラクタの書き換え、可変長レコードのための制御。

○プログラムの論理設計の段階において、あとで大幅な修正の必要が生ずるのを避けるために、次のようなことを考慮しておくことが大切

である。

① 既存または計画中のデータ・ベース。

② 他のプログラムとの関係、データの互換性。

③ エコーのコントロール

④ 障害対策（チエックポイント・リスタート、リラン、エラー回復）。

⑤ 機密保護。

⑥ プログラムの拡張性。

⑦ プログラムのテストと保守。

3.1.5 オペレータへのメッセージ

○オペレーション中に人間の介入と、それによる誤りをなくすために、メッセージは少ないほうが望ましいが、どうしても必要な場合には、できるだけわかりやすく使いやすく、効率的なメッセージを考える。そしてメッセージの形については日常語の分に近い形にするかコード化するかが考えられるが、その種類とか使う人によって異なってくる。新しいメッセージのきめ方とか、特にコード化した場合にコードの統一を管理するようにする必要がある。そしてメッセージ自体とその意味、それが出たときの処置等をリストした一覧表をつけることが必要である。できればハードウェア、プログラムおよび入力データのエラーに対するメッセージもここで標準化しておく必要がある。

3.1.6 プログラムの部分単位とその結びつけ

プログラムの中でサブルーチンが使われる場合や、いくつかのモジュールに分けてプログラムが作成される場合には、プログラム・リンケージやインターフェースに約束された形のもの — つまり標準のリンケージや標準のインターフェースを使うようにする。

なお、サブルーチンには開いたサブルーチンと閉じたサブルーチンが存在する。サブルーチンの名前、入り方や出る形、その呼び出し列、リンケージとして特別の役割りを持つレジスタ、保存区域（save area）、マクロ命令の形などが標準化されるべきである。

1つのプログラムが一度に主記憶装置に入らないときには、区分化（segmentation）が行なわれる。プログラムの根本（root）の部分とオーバレイ・セグメントとの結びつけは、アセンブルやコンパイルするとき、リンク・エディットのとき、ジョブ・エントリーのとき、またはタスクの実行時に行なわれる。

モジュール間の実行時の形態には、単純、オーバレイ、動的逐次（dynamic serial）、動的平行（dynamic parallel）がある。

サブルーチンやサブ・プログラムを逐次再使用可能（serially reusable）や再入可能（reentrant）の形につくることがある。また、実行中に自分自身のプログラムの中に再入できる再入可能プログラムのことを、再帰（recursive）プログラムということがある。

記憶域の割りつけには、共通に使うスペースを固定的に割りつけたり、動的に割りつけたり、スペースが足りないときは、オーバレイやロールアウト／ロールインを行なう。最近では仮想記憶装置（virtual memory）も一般に使われようとしている。

マルチプログラミングが効果的に実行できるように、たとえば目的プログラムが使える主記憶装置のスペースの大きさ、入出力装置や補助記憶装置の台数の制限、入出力の媒体はテープやディスクとして、そのためSPOOLを行なうことなどを配慮する。

3.1.7　ファイルの編成と入出力アクセス方式の選択

しばしば使用するオペレーティング・システムやプログラミング言語によって利用できるファイル編成が異なってくる。ファイル編成により、下記のようなさまざまなアクセス方式が使われる。

○順次アクセス

○直接アクセス

○索引順序アクセス

○区分アクセス

○テレコミュニケーション・アクセス

○仮想記憶アクセス

○緩衝域は一つのことと複数のことがある。緩衝（バッファリング）の手法としては、　○単純緩衝、　○交換緩衝、　○連鎖緩衝、　○セグメント緩衝、　○動的緩衝がある。

緩衝域とプログラムの間のレコードの受け渡し方法、すなわち、転送モードには次の4つがある。移転モード、位置指定モード、データ・モード、代替モード。

3.1.8　流れ図の作成

プログラム作成前には、プログラムに要求されていることが完全に明確になっているかをチエックして、あとでやり直しがないようにすべきである。そのために流れ図というものが大いに役に立つ。よく考えて論理流れ図を書き、そして詳しいブロック・ダイヤグラムというのはきわめて複雑なルーチンの場合にだけ書くことにする。流れ図のおもな目的は何かというと、次の4つがある。

○プログラムのインターフェースを明らかにすること。

○実行時に行なわれることをシミュレートしてみること。

○可能性のある異なる解を評価し、最上のものを選択する助けとなること。

○コーディングを容易にし、かつ誤りの修正や保守を楽にすること。

注意深く考えて流れ図の標準をつくり、それを厳守しないと流れ図は往々にして粗末になりがちである。そして、プログラム開発において流れ図作成においてかなりコストがかかり、また、次の段階（コーディングとデバッギング）に要する時間に大きな影響を与えるので、その有効性に注意すべきである。

流れ図の種類とレベルについて見ることにする。

流れ図にはシステム流れ図とプログラム流れ図がある。後者はさらに、

○全体的プログラム論理図（マクロ流れ図）

○ルーチンまたはモジュールの詳細図（ミクロ流れ図）

がある。場合によっては、モジュールの関係を示す図（プログラム構造図）をも作成することがある。

ここでいうマクロ流れ図は、プログラム全体を積木的部分に論理的に分けて表現したものであり、ミクロ流れ図は、それを見れば完全にコーディングができる程度に詳しく書かれたものである。しかし、ミクロ流れ図の場合に注意しなければならないことは、不必要に詳しくし過ぎないということで、それは全体の論理の流れがぼけないようにするためである。

流れ図記号の書き方については、「ＪＩＳ　Ｃ　６２７０－１９７０情報処理用流れ図記号」を参照。この規格には

○流れ図記号

○流れ図中の記述

○結合子（connector）

○横　線

○二つ以上の出口

○同種類の媒体の反復表現

が規定されている。こうした記号や書き方の標準化と同時に、流れ図を書くのに使う紙の大きさをも統一し、またそれのファイルのしかた等も標準化することが望ましい。

流れ図を書く上での２、３の注意を述べることにする。

論理の流れが下の方向または右の方向に向っているときには矢印は不要である。流れ図記号の中の記入は、できるだけ簡略化し、数学的記号をも利用する流れ図記号のそばにプログラムの中に使われたラベルを書くことにより、プログラムのリスティングとの対応をはっきりさせることができるし、また、マクロ図とミクロ図があるときには、それらの対応をうまくつける。他のページに続く場合には、出口の結合子にページを書き込む。

流れ図を作成したあと、コーディングを始める前に、厳重にチエックすることが必要である。そのチエックする場合に、どういう要領でどの点をチエックするかということを定めたガイダンスがあると好都合である。そのガイダンスの中には、たとえば標準のパラレル記号が使われているかどうかとか、上に述べたような規則が忠実に述べられているか、といったようないわゆる形式的な問題、もう１つは、論理設計が誤ってないかということを、もう１度注意深く検討することが望ましい。

3.1.9　流れ図の自動作成

流れ図をきれいに書き上げること、およびすでに存在する流れ図を修

正する、つまり保守の仕事をするということはかなり時間もかかりめんどうな作業であるが、最近では、流れ図を計算機で自動的に作成させるためのソフトウェアが、内外のシステム・メーカーやソフトウェア会社によって提供されているので、もし可能ならそういった自動的に流れ図を作成するソフトウェア（ドキュメンターの一種）を使用することを検討する価値がある。自動的に作成される流れ図の中には大きく分けて次の二種がある。

1）特別の言語でコーディングされたものを入力とするもの。たとえばＩＢＭのFLOWCHART　1360

2）アセンブラ語、FORTRAN　COBOL　PL／1等で書かれた原始プログラムを入力とするもの。

たとえばApplied Data ResearchのAUTOFLOW

前者の場合に、流れ図に含まれる記号のそれぞれの機能と他との関係を記述する簡単な言語があり、その言語で流れ図の各記号、およびその記号の前後関係を書いて、それをカードにせん孔して計算機に入れると、流れ図が自動的に編集され、印刷されるようになっている。この言語はごく単純で、機械の特性とは関係なく容易に書くことができる。

流れ図の自動作成により、プログラムのドキュメンテーションの作成および保守に要する時間や全体としてのプログラムの作成の労力を減らすことができる。しかも、統一された形の流れ図の作成、すなわち流れ図の標準化が達成できる。

3.1.10　決定表の使用

ごく最近では、決定表に対する関心が一部では高まっており、流れ図を使用すべきか決定表を使用すべきかという議論がかなりあるので、こ

こで決定表について若干触れておくことにする。

こまかいプログラム論理を表現するのに決定表（decision table）を使ってもよい。

ある時点で存在する条件によって処理の流れが選択される場合には、多数の分岐する流れを呈示し、かつ説明するのに都合がよい。システムの定義のために、流れ図より効果的で、簡潔で、かつ完全なことがある。

決定表の標準化の問題については「ドキュメンテーションの標準化」P２２６～２３０を参照。

決定表の使い方には次の３つのレベルがある。

１）複雑な論理過程を示すドキュメンテーション。

この場合には流れ図の表現を補うため、もしくは論理表現の唯一の方法として用いる。

２）決定表より直接にコーディングされ、相互参照される。決定表が記憶装置内に実際につくられ、処理の間にプログラムによって使われることもある。

３）決定表そのものが原始言語を構成し、それを入力として、COBOLやFORTRANのプログラムに変換する前処理プログラムが用意されている場合もある。

決定表と流れ図の比較は、W. Hartmanほか著の“management Irformation Systems Handbook”に書かれていて、その結論だけを要約すると次のようなことになる。

○単純の場合やシステム・アナリストが完全にわかってるときは流れ図を使う。

○複雑な場合や入力の情報が不完全の可能性があるときは決定表を

使う。

○事情により要求されるときは、まず決定表をつくり、それから一つ以上の流れ図をつくり出し、最も適当なものを選んで使う。

○現状分析をプロジェクト・チームがマネジメントに説明するときは、決定表のほうがよい場合が多い。

3.2 コーディング

プログラムの設計がすむとコーディングに入るが、このやり方を標準化すると、誤りを減らし、他人にも見やすいし、流れを追いやすいし、流れ図との相互参照にも好都合になって、誤りの検出、修正、および変更や変換が楽になる。また、他人への引継ぎや教育もしやすくなり、ひいては品質の向上となり、しかも管理がしやすくなるという可能性がある。

3.2.1 言語の選択と効率的な書き方

コーディングに入る前に、どのような言語が一体利用可能であるかどうか、それらの言語のつくられた目的、長所、短所、使いやすさ、標準化の動向等を考えて、最も適切な言語を選択し、かつ当該プロジェクトに使用するサブセットをきめる。このようなサブセットを定義した標準マニュアルを用意しているユーザーも少なくない。

アセンブラ語の使用は最小限にとどめ、やむを得ず使用する場合には何らかの規定を設けておくことが望ましい。

命令やステートメントの標準的な使い方、共通な名前、共通に使う手続等をきめておく。

レジスタの割り当てループのつくり方などから見て、目的プログラムの効率を上げる方法があれば、それも規定しておく。ただし、通常は、

初めから目的プログラムの効率を上げるほうに没頭するようなことは避けるべきで、あくまでもプログラムの論理を正確に命令のステップに翻訳することを第一義とすべきであり、最後の段階で部分的に、あるいは場合によっては全体的に効率を上げるということを考える。

これらをまとめた手引書を作成しておくと好都合である。

3.2.2　コーディングの形態

よりよいコーディングをいうのは次のような場合をいう。

- 処理時間を最小にするもの
- 保守が容易（変更が容易）
- 信頼度が高い

○プログラムはできるだけ小部分、もしくは大よそのきめられた大きさに分け、それぞれ可能な限り独立にコーディングして、別個にテストできるようにしておく。初期設定（initialization）またはハウスキーピングは必ずそれぞれのモジュールで行なうようにする。

原始プログラムの形態やその中に書き込むものをきめておく。アセンブラ語の場合には次のような事項である。

○見出し、日付け、ページ番号
○左右のスペース、左側の揃え方
○上下のスペース、行間のとり方
○小モジュールごとにまとめて前後空白行
○小モジュールに見出し
○作業スペースや定数を1個所に
○注釈と備考（キーとなる命令に説明を入れると論理を追いやすい。）
○パラメーターの説明、スイッチのセットに説明

○前後関係がつけやすいようなくふう

○流れ図との対応

コーディングに対する態度としては、高尚な技巧よりも簡明さ、誤りの検出や修正のしやすさを第一とする。処理時間対コア・スペースやファイル・スペースの問題に遭遇することがあるが、どちらが重要であるかをそのつど考えコーディングを行なう。大多数の場合には、最も単純なコーディングが最も早く最上の結果を生み出す。

次のような点を考慮する。

○プログラムの意味がわかりにくいときは、必ず注釈で補うようにし、読みやすいプログラムをつくり上げる。

○正常な状態ではオペレータが介入することはできるだけ少なくする。

○記号の（symbolic）のラベルや記号のアドレスは単なる一連番号のようなものを避けて、たとえ限られた文字数内であっても、できる限り意味があるもの、あるいは意味がわかるような記号　たとえば、処理の名称やデータ名の略したもの　を使う。

○飛び越しステートメントには分岐の条件と行き先をわかりやすく注釈で説明するようにする。

○レジスタや共通な区域の使い方を統一し、一貫性のあるものにする。

○手書き文字の書き方を統一しておくことが、キーパンチの誤りを減らすのに非常に役に立つ。

○コーディングが終わったならば、机の上の検査を十分に行なう、その場合に他のプログラマにチエックさせることが望ましい。また、流れ図の場合と同様に、チエックのしかたをまとめたガイダンスがあれば一番よい。

3.2.2　制御カードの作成

オペレーティング・システムを使う場合に、制御プログラムへの指示としての制御カードを作成する必要があるが、それに必要なジョブ制御言語（Job Control Language）の書き方をできる限り標準化しておくことが望ましい。そして標準化した書き方の手引書をつくっておくとよい。

○プログラムごとに一々制御カードをつくるというのはたいへんめんどうなので、できる限りＯＳの中にカタログしておいて使うことが望ましいし、また、制御カードを生成するようなジェネレータを利用しているユーザーも多数ある。

3.3　誤りの検出と除去

プログラムのコーディングが終わると次の仕事は、そのプログラムをアセンブルまたはコンパイルし、それから用意したテストデータでもってそのプログラムが正しいかどうかテストすることになるが、できるだけむだな機械の時間を減らすために、机の上で十分チエックし、コンパイルとかアセンブリ時においてプログラムのリスティングをやはり注意深く検討し、いわゆるクレティカル・エラーはできる限り除いてテストすることが望ましい。また、テストに対しては十分な計画や周到な用意が望ましい。

3.3.1　デバッギングの手法と補助手段

デバッキングのやり方を統一化し、かつプログラムに含まれるあらゆる機能とすべての論理の流れをテストするようにすれば、テストの効率が上がり、かつ正確度を高めることができる。また、スケジュールの作成や必要とする機械時間の見積もりもしやすい。そしてできればプログ

ラマがオペレーションに立ち会うことはせずに、オペレータに完全にまかせてテストをするようにすべきである。

デバッキングの方法としては、　○コア・ダンプ、　○テープ、ダンプ、　○操作卓の表示の読み取り、　○重要な記憶域、　○作業区域、○プログラム・スイッチ、　○指標インデックス、　○指標レジスタ、○入出力バッファの印刷、　○中間結果の出力、　○各命令の実行の追跡（trace）、　○論理的流れの追跡（flour trace）等がある。大量のコア・ダンプやテープ・ダンプは避けたほうがよい。

最近では、プログラミング言語に効率的なデバッキングの補助手段が含まれているものがあり、これらの有効性、長所、短所等の検討を行なうべきである。もしそのようなものがない場合には、論理上重要な点において、中間データの一部または全部を印刷して出すようにする。

主流ルーチンの完成前もしくはテスト用の機能を含めるために、処理モジュールをテストする際に必要なシミュレータを開発することがある。

カード・デックがせん孔されたらよくチエックして、明らかなせん孔ミスは全部見出すように心がけるべきである。往々にしてミスパンチのためにむだなテスト時間を費していることが少なくない。

3.3.2　テストのための設計と計画

デバッキングのことについては、設計図からコーデイングを通じて十分に配慮されていなければならない。テストがしやすいように、できる限り独立な小部分に分けてコーディングを行ない、中間結果の置き場をまとめるようにする。コントロールの流れを調べたり、中間結果を取り出すようなテスト用のルーチンを組み込んでおくこともある。

テストをいくつかの部分（ケース）に分け、それぞれのケースの対象、

目的、テストされる機能、テスト・データをまとめた仕様書をつくり、承認を得るようにする。あらゆる場合をもらさずにテストするため、チェック・オフ・リストなどを用意しておくとよい。

3.3.3　テスト・データの作成

テスト・データは次のようなものがある。

○実行時に起こり得るすべての条件をシミュレートするようなもの。

○プログラムの中のすべての経路を実行させるもの。

○テスト・データの作成にユーザー側が助けとなるような場合には、その援助を得るようにする。

○テスト用の入力データを生成するプログラムや中間結果や最終結果を見やすい形に変換するプログラムを作成することもある。最終のテスト結果は、入力データとともにきちんと保存しておく。そして、テスト・データの変更や追加がなされる場合には、それらを確実に行なう必要がある。

3.3.4　テスト工程の設定と管理

部分ごとに独立にテストして積み上げていくテスト工程の設定と積み木方式のテストが望ましい。テスト工程としては、サブシステムのテスト（単位テスト）、システム・テストがあり、さらに回帰テスト（regression test）が行なわれる。

テストの目的、内容、手続、ルーチンやサブルーチンの組み合わせ、入出力データ、コントロール・カード、スケジュール等をしるした仕様書を作成する。テストの進め方の一例としては、次のようなものがある。

○アセンブリやコンパイルして見出された誤りを直す。

○各ブロックにつき、おもな論理を簡単なケースの人工データでテストする。出力が正しければ次の論理の型をテストする。

○より複雑なケースをテストする。

○全般的論理、たとえば、ブロック間のコミュニケーションをテスト

する。

○エラーの発見やその他の理由で修正、追加が必要になったときに、前の工程まで戻ってテストを行なうこと ―回帰テスト― を行なうことがある。

システム・テストでは、システムの構成要素間のインターフェースが正しいか、要求された機能を持っているか、性能は設計目標に合致しているか、などのテストする。そしてまた、システムがその前の段階で正しくチエックされているかどうかをここで確かめる。このシステム・データの量が多いばかりが能ではなくて、いかにして少ないデータであらゆる面をテストするかということが肝心であり、そしてあらゆる例外と可能性が少しでもあるような誤りを含んだデータを使ってテストする必要がある。

誤りが見出されてそれを修正する場合には、目的プログラムのパッチだけですまさないで、原始プログラムを必ず訂正する。

○テストの結果は厳格に検討、評価され、承認を受ける手続が確立されているべきである。

○テスト・ランの回数やエラーの型を記録しておくとよい。

3.3.5　リアル・タイム・システムのデバッギング

リアル・タイム・システムのテストは入力が直列的に入ってくるバッチのシステムのデバックとは根本的に異なり、はるかに複雑で困難な場合が少なくない。そしてリアルタイム・システムでは、　1.入出力の要求を処理し、　2.必要なアプレショナル・プログラムの経路を守る必要がある。マルチプログラミングが行なわれる場合、おのおののプログラムの使用可能な状態を制御プログラムが把握していなければならない。

リアル・タイム・システムのテストのステップとしては次のようなものがある。

1）一時点で1つだけの変数を入れて、データを扱う論理をテストする ― 入力メッセージをシミュレートする特別なプログラムをつくることがある。

2）各モジュールが個々にデバッグされた後、それをつなぎ合わせて呼び出し列やプログラム間のリンケージをテストする。

3）処理プログラムをよくテストしたら、システムに端末をつないで、そこからメッセージを入れる。入出力のスナップショットをとる、テープ・ログをとる。

4）端末のオペレータの種々のエラーに対処できるかテストする。

5）ピーク時の負荷を処理する能力 ― 過負荷（overload）が生じたときにどうするかを調べる。シミュレートするソフトウェアを使う。

システム・テストでは、一般的条件は異常条件がまれにしか生じない条件をテストする。そして種々のトランザクションの相互反応を調べ、また、相互ロックアウトの状態をテストする。

解析モードで走らせることを可能にするやり方もある。この場合には、必要な管理プログラム、またはデバッギングプログラムがコアに読み込まれ、プログラム実行の途中で必要な任意の情報がコア内の区域に記録される。

3.4 対話型システムや仮想記憶装置を利用したプログラム開発

最近ではタイム・シエリング・システムの発達により、対話方式の端末を使ってプログラム開発に効率を上げているケースが多くなってきている

し、また最近はいくつかのメーカーが仮想記憶装置を提供するようになってきているので、これらを利用した場合のプログラムの開発についてごく簡単に触れることにする。

3.4.1　ＴＳＳが提供する機能

ＴＳＳが提供する機能としては次のようなものがある。

○原始プログラムの生成または修正（ＥＤＩＴなる指令）

○会話方式でのプログラムの作成、デバッグ、実行

○デバッグのための補助手段 — たとえば、プログラムの途中で割り込みを起こして、中間結果を見る便宜（ＴＥＳＴなる指令）、記号名（symbolicname）のままでデバッグ

○データ・セットのコピー、マージと結合、テストのためにデータ・セット内のスペースの割り当て

○プログラムをテスト・ライブラリよりプロダクション・ライブラリに転送

○バッチで使える主要言語（COBOL，FORTRAN，PL／I）の会話式実行、サブルーチンの呼び出しと連結が容易

○バッチで使えるデータ・セットの形式（format）の使用

○他のユーザーを妨害することなく新しいプログラムをオンラインでテストする安全性（security）

3.4.2　ＴＳＳの利点と効果

ＴＳＳを使った場合、ターンアラウンド時間が短いので、一つの仕事にプログラマが集中でき、待ち時間をむだに費したり、他の仕事をかけ持ちするようなことがなくてすむので、プログラムの生産性を上げることができ、継続的かつ集中的にプログラムの作成が可能で、プログラム

開発と実行の間に人間とシステムとの相互反応が早くなり、作成時間が短くなり、費用を減らすことも場合によっては可能となる。またバッチ処理で起こりがちな不必要なリストや膨大なコア・ダンプが減り、アプリケーションをより少ないマンパワーでより早く導入できるのと、プログラムの変更が容易かつ短時間で可能になる。

例として、Data Processorの１９７２年５月号に載っているが、アメリカのSikorsky Aircraft CompanyではＴＳＯ端末４０台を使ってプログラム開発をやっているが、それを使ったためにプログラマの生産性が２００上がった、つまり３倍の効率を出した。従来は既存のプログラムの保守にあまりにも多くの時間を費し、新しいアプリケーションのプログラミングは少ししかできなかったけども、ＴＳＳを使うことにより、変換をやりながら新しいプログラムの作成が可能になったと報告されている。

3.4.3　仮想記憶装置の利用

仮想記憶装置を利用すると、次のような利点があるといわれている。

○実記憶容量による制約が少なくなり、

— プログラマの生産性が上がる

— 新しい大きなプログラムの開発の期間が短くなる

— プログラムの拡張や維持が容易になる

○実記憶域の管理の面からいうと

— 実記憶装置全体を有効に使える

— 同時に多くのプログラムの実行が可能となり、システム資源の効率的な利用ができる。

○システムのスケジューリングが容易になる。

○小さなシステムで大きなシステムのバックアップが可能になる。

○実記憶装置の容量の異なるシステムでプログラムの使用が可能になる。

3.5 プログラムのドキュメンテーション

プログラムのドキュメンテーションには、最初の入出力関係の仕様書から流れ図およびプログラムの論理構造の説明とかいったものがすべて含まれるわけだが、そういったものをすべて作成し、かつテストしている段階に論理的な誤りが見つかれば、そのつど流れ図を直すということもあるが、最終的には、テストが完了した段階でもう1回プログラムのドキュメンテーションを見直して完成させる必要がある。

3.5.1 プログラムのドキュメンテーションの内容

プログラムのドキュメンテーションは下記のような内容を含んでいるか、あるいは、もし含まれない場合にはどこにそれが書かれているか、どこを見たらいいかということを明示しておくことが必要である。

○プログラムの目的、機能、方法の全般的記述。

○プログラムの入力、出力、使われるファイルの記述。

○プログラムの論理を示す流れ図（diagram）。

○指示の出力メッセージの記述。

○コーディング関係の資料、たとえば、コンパイルやアセンブルのリスティング、記憶装置内部の印刷、使われる行列か表の記述。

○テストの計画。

○プログラム・テストを指示。

○操作指示。

○上記は新しいシステムを開発する通常の場合であり、複雑なシステ

ムや既存のシステムを拡張もしくは修正する場合には、これに異なった内容となる。

次のこともドキュメンテーションの形と内容に影響をおよぼす。

○決定表による論理の表現。

○高水準言語の使用。

○アプリケーション・パッケージの利用。

○汎用ソフトウェアの利用。

3.5.2　プログラム・ドキュメンテーションの例

プログラム・マニュアル

目　　次

4.2　表と手法

5.0　リスティング

5.1　機械語に翻訳後のリスティング。

5.2　ラベルと記号の表

5.3　命令のリスティング

5.4　スイッチのリスト

6.0　プログラム・テストの計画

4. システム導入の準備と運用

4.1 機器設計計画

4.1.1 まえがき

機器の設置計画は、よく理解され、またかなり経験がある問題であると言われている。しかし、特にリアルタイムシステムの場合は、機器設置計画は前もって十分検討しておく必要がある。

まず、システムが新しく導入されるシステムであるか、前のシステムを取替えるシステムであるかということによってかなり問題が異なってくる。

新システムの場合に設置計画者が往々にして忘れがちなことは、計算機の償却期間は大体5年程度であり、5年後に新しいシステムが導入されることを想定して、あらかじめ取替えるべき新しいシステムの場所を確保する配慮をすべきである、ということである。

計算機システムを導入すると、業務の流れが変わり、しかもいったん業務が計算機中心で流れだすと途中でその業務を打切って人手に切替えることが非常にむずかしい。そのため、ある時期には、2つのシステムを同時に働かせなければならないことが生ずるからである。経済的な面で、完全にフリーな場所をあらかじめ確保することは困難である。
しかし、少なくともシステムを導入する前に次に置替えるべきシステムの場所を想定しておくことは必要である。

次に取替えシステムの場合には、上述のように第1期のシステム導入時に十分配慮したにもかかわらず、種々の原因から設置場所を遠く離れた場所にしなければならぬ事態が生ずることがある。例えば、本社ビルが他の必要性から移転する場合、それに伴ない計算機システムを移さねば

ならぬことが考えられる。

（実例－1）　ＮＨＫのトピックスの取替えシステムは、場所を放送センターに移さねばならない。

（実例－2）　国鉄のマルス計画の場合、第4期までのシステムは、1つのビルを上へ上へと継ぎ足して8階まで作った建物で収容していたが、ついに収容しきれず、国立に移転した。

4.1.2　計算機室機器配置

(1)　計算機室内の機器配置

機器配置の計画を立てる場合に考慮すべき基本的制約事項を列記す；

1.　床の重量制限

2.　利用できる空間

3.　柱の位置

4.　許される最大のケーブル長

5.　オペレーションや保守等に必要な空間

次に、計画の際に考慮すべき基本的な事項として、第1に、各機器の重量と、それを配置する場合のバランスに特に注意しなくてはならない。

第2に、見通しや作業性の問題から柱の位置に注意すべきである。

第3に、業務の流れをよく分析して、オペレータが働きやすい機器配置をとること。

第4に、メインテナンスの空間を十分に取っておき、特にメインテナンス作業がオペレータの操作を妨げないよう配慮すべきである。

第5には、一般に計算機システムが導入された後、システムの使用範囲が拡張されるに従って、特に端末機器、ディスク、テープ等の台数が少しずつふえていく。このことを考慮した余分な場所をとっておく

必要がある。

計算機室内の機器配置の1列が図4－1である。（114頁参照）

細かな機器設置の問題点は、各メーカーから出されている色々なインストレーション・ユーザー・マニュアルの中にも、注意すべき事項が述べられている。システムを設置する場合には、このようなマニュアルがかなり参考になる。

建物計画を立てるときに、機器の搬入の計画をあらかじめ考えて建物の間取りや入口などを考えておかねばならない。機械の設置場所が、高層ビルにある場合などは、荷物用エレベータで運び込めない事態もおき、非常に大きな問題となる。

(2) システムによる計算機室の環境の違い

(1)では標準的な機器配置について述べたが、リアルタイム・システムの場合には、機器配置についての配慮は、各システムごとに異なり、一般的なことは言えない。しかし、現場でプロセス．コントロールとして使われる計算機の配置には注意が必要である。これは、計算機が現場の非常に挨の多い場所や、温度、湿度の高い場所で使われることがあり、配置の問題は別の観点から考える必要がある。

4.1.2 その他のスペース

(1) カード・テープの保管場所

次に計算機の入出力である、カード、テープ、ディスク、プリンタ用紙等の問題を考えておかねばならない。特にテープやディスクの場合は、重要なデータの内容がオペレータの誤操作により壊されることがある。このため、保管の場所を十分注意して分けておくとか、何らかの安全処置を講ずる配慮を、まず場所的に考えておくべきである。

またセンター業務の場合のように、多くのユーザーが各自のカード、テープ等のデータを保管する場合がある。この場合は、保管場所を計算機室の近くに設けておくことは、センターのバッチ業務運営上、非常に重要なこととなる。

また、このようにして保管されたユーザーのデータを、いかに効率よく計算機室内に運び込むかという問題について建物の配置計画の段階で十分考慮しておくことが大切である。

(2) オープンパンチ室

大量のデータをパンチしたり、ユーザーが自分でデータやプログラムをパンチする、オープンパンチ室をどこへ設置するかという問題も、スペースをかなり必要とするので、あらかじめ建物の配置計画の時点で考えておかねばならない。

(3) 要員・管理者のスペース

計算機のオペレータだけでなく、サービス・エンジニアやメンテナンス・エンジニアのための場所を計算機室の近くに確保しておくことは重要である。計算機がダウンしたような場合、回復時間を問題にすると、メンテナンス・エンジニアがどこに居るかにより、MTTRがかなり違ってくるからである。

また計算機システムをスリーシフトで動かす場合、要員の休養場所や寝室等も、準備しておく必要がある。

4.1.3 各種コーティリティ

(1) 電源設備

バッチシステムの場合は、主として電源の安定性の問題だけ考慮すれ十分である。信頼度があるほうがよいのは当然であるが、バッチ

処理の場合には、一度電源が切れても再度ジョブを最初から流せばたいていの場合は十分である。そのため電源設備に多くの費用をかけることは、望ましくないからである。

オンラインシステムの場合には、電源ダウンによるサービスの中断は大きな混乱を引起すため、十分な配慮が必要である。一度電源が切れると、分散している各端末に電源ダウンによる故障であることを周知徹底させる対策立てねばならない。

また、オンラインシステムの場合には、電源の瞬断を非常にきらうため、あらかじめ対策を立てておく必要がある。

一般的に電源として考えておくべきことは、電圧と電圧変動の問題、周波数と周波数変動の問題、消費電力の問題である。

オフィス内で計算機を使用する場合は、最近電力事情がかなりよくなっているため、直接計算機にオフィスの電源を供給しても問題は少ない。しかし、工場内で計算機を使用する場合かなりのピークロードがかかる場合や、電力の使い方が変則的な場合に、周波数波形の崩れや、電圧の瞬間的変動などの問題が起る。このような場合は、別途の方策が必要である。

電源の問題は電力会社と計算機メーカーの機器設置エンジニアと、十分協議してあらかじめ調査しておくことが、後にトラブルを少なくするために重要なことである。

外国から輸入した機械の場合は当然であるが、地域的な６０サイクルと、５０サイクルの周波数の違いと機器とのマッチングに対する考慮も必要である。

オンラインシステムの電源設備の一例として、国鉄のマルスシステム

の場合は、2系統から並列に電源を供給している。さらに、両電源とも切れた場合のために、自家発電装置を備えていて、外部電源が切れた場合自動的に自家発電装置が働くようになっている。しかも、外部電源を重いフライホイルをつけたモータジェネレータを介することにより、自家発電に切替える時の瞬断もできるだけ少なくなるよう考慮している。

(2) 照明・空調・その他

照明は、計算機室内では、オフィス程度の照明で十分である。しかし、メインテナンスする時に、明るさのバラツキがないような照明器具の配置が望ましい。

空調は、最近のコンピュータメーカの売込みによると、あまり考えなくてよいということではあるが、空調をした方がよいのは当然である。ただし、プロセスコントロールなどでオンラインで使用する計算機の場合には、この問題を十分考慮して計算機の機種を決定すべきである。

このほか、電話などの問題は、通常のオフィスと同じに考えてよい。またオペレータを減らすために、オペレータ室と機器室との間にＩＴＶを設置することも効果のある方策の1つである。

計算機の機種によっては、空調との関連などで、色々なコーティリティを配置しなければならないことがあるため、機種を決定した時にチエックしておく必要がある。

4.1.4 オンラインシステム特有な問題

(1) 回線

オンラインシステムの場合は、非常に多くの回線を使うため、この品質、及び回線をどのように手配するかということを、あらかじめ機器設

置のときに考えておく必要がある。特に日本では、通常のユーザーは、電々公社の回線を専用回線あるいは加入回線としてリースするため、回線にかなりの費用がかかることが多い。また回線にオンラインシステムを接続する場合、電々公社の厳しい規格があるため、計算機メーカーのスペシャリストと相談してチエックしておくことが必要である。

(2) 端末機器室

オンラインシステムの場合、回線の端末につながる端末機器、すなわち銀行の窓口機器、国鉄のマルスの切符販売機器、その他各種ディスプレイ装置などを置く端末機器室の問題がある。これは計算センターと違って非常に多くの個所に分散し、空調などに費用をかけられない。そのため、機種の選択をする時、空調の必要性その他について、十分調査する必要がある。　　また端末機器は現場で使用するため、騒音の問題を考慮して、タイプライタなどの音が、通常のオフィスの仕事を妨害しないような機器室の配置が望ましい。

なお、プロセスコントロールの場合には、端末に種々の測定装置を用い、しかもこれは対象とするプロセスの特性に依存する。このため、計測技術の問題に属することではあるが、やはり、オンラインシステムの1つの問題として、設置する時あらかじめ十分なる調査が必要である。

4.1.5　非常時対策

これまでは、あまり問題となっていないが、火災あるいは地震などの非常事態の場合の対策を考えておくことは非常に重要である。計算機室内には、非常に重要なデータバンクであるディスク、磁気テープが保存

されており、これらが火災で焼失すると、場合によっては企業の活動が停止することすらある。

この対策として機器室とこれらのデータライブラリとの間に防火シャッターを設けるとか、データをたくわえている室の火災予防として、スプリンクラーは、後の処置に困るため、研究段階ではあるが、炭酸ガスを充満する方法などが考えられている。現在のところ、これに明確な対策がないため、非常に重要なデータは、あらかじめ二重にコピーして、建物の違った場所にたくわえるという処置を講ずることも今後考えるべきであろう。

日本では地震が多いため、ポイント．ロードが地震の時どのような影響を受けるかについては、建築の分野でも十分解明されていないため、これについて十分な余裕をとっておく必要がある。

4.2 受入試験

4.2.1 中央機器の組み立てテスト

メーカーは中央機器、すなわちＣＰＵおよびそれの周辺機器をあらかじめ工場内でユーザーのシステム構成に合わせて充分にテストをした後、各機器の接続を一度切ってユーザーへ持ち込みそこで再び組み立てテストを行なう。このテストはメーカーが主体になるのでユーザーとしてはメーカーの「テスト完了通告」時点で受け取ればよい。

この組み立てテストの期間はシステムの大きさによって異なるが、大体数日から２週間程度を見込んでおけば充分である。

4.2.2 端末機器の単体試験

端末機器のテストもハードウェア上の問題はメーカー側で充分検討されておりまた数も多いので、ユーザー側としては考える必要はない。

ただ、サンプル的に取り上げてメーカーの試験項目に合わせて再確認する程度のことはしておいた方が良い場合もある。また、これを回線に結んだ時に種々の問題が生じるが、これは別のところで考える。

4.2.3　端末シミュレータと結合試験

受入試験の段階で中央機器と端末機器をつないで同時にテストする時があるが、これは回線のトラブル及び中央機器、端末機器の種々のトラブルが重なり合うので、一体どこにハードウエァ上の問題があるかを見つけるのはかなり難かしい。そのため受入試験の前に、まず中央機器から回線を通して端末シミュレータと言うものを使ってテストが行なわれる。この手順は第4－1図に示されるように3つの段階がある。(114頁参照)

まず最初の段階はソフトテストと言われるものでＯＳと通信制御と端末シミュレータをすべてソフトで組み、通信制御プログラムのテストを行なう。

その次の段階は通信制御装置ＣＣＵをつなぎ込み、このＣＣＵから出ている端末回線をハード的に接続する。それからテストプログラムを使用して実際の回線は使わずにＣＣＵのところでショートカットしたような形でのテストを行なう。

その次の段階は実際にMODEM　すなわち変調装置をつないだ状態で回線を延ばし、その回線の先で2回線を接続してテストプログラムを使用してテストを行なう。最終的には点線のところの端末機器をつないでテストを行なうことなる。

このように試験を段階的に行なっていくことによってハードウェア上の接続あるいはシステム上のトラブルは個々に発見することができるので、非常にトラブルのシューティングが楽になる。また回線が非常に多い

場合にはいろいろの回線の組み合わせについて同様のテストを行なう。

4.2.4　ＯＳ、コンパイラ、ユーティリティのテスト

通常のバッチ業務でのＯＳであるとか、コンパイラ、ユーティリティのテストは１号機でなければ、すなわち今まで数台ないし数十台他のユーザーによって使われたシステムであれば受入試験をする必要はない。

しかし、普通ＯＳはユーザーの周辺装置のシステム構成に合わせてシステムジェネレーションを行なうことが多いので、非常に特殊な端末機器の構成である場合には、一応典型的な業務の流れに対してユーザー側でテストプログラムを用意してＯＳ、コンパイラ、ユーティリティのテストを行なった方が良い。

オンラインシステムの場合にはＯＳの一部改造とか特殊なユーティリティあるいはテストプログラムなどを作るので、これらを合わせて受け入れの時点にテストできるよう準備しておけばオンラインシステムのインストレーションを非常にスムーズに行なうことができる。

4.2.5　ユーザープログラムの組み合わせテスト

バッチ業務の場合にはジョブと言う形でユーザープログラムは流れるので、ＯＳ、コンパイラ、ユーティリティとユーザープログラムとの組み合わせテストは、通常のＯＳの種々の機能を使用する１つのテストプログラムを通すことによって行なうことができる。

しかし、オンラインシステムの場合はユーザープログラムとＯＳとの間に非常に細かなコミュニケーションがあるので、これは受け入れテストと言うよりむしろシステムの稼動の全体のテスト、いわゆるシステムテストを行なうことになる。もっともオンラインシステムの場合には受入試験の段階ではユーザープログラムの開発が完全に終了していることは少

なく、各機器を設置してからユーザープログラムを開発することが多い。従って、ユーザープログラムの組み合わせテストを行なうよりはプログラム全体の完成後のシステムテストまでこのテストを延ばすことが多い。これはまたオンラインテストについても同様で、前述の端末機器のテストのみを受入試験の段階で行ない、完全なシステム的なオンラインテストはユーザープログラムの開発後に行なわれることになる。

4.2.6　製作と監査の分離

このような受入試験を行なう時テスト項目をメーカーに一任することが多いが、これはやはりユーザー側の問題である。製作側すなわちメーカーで試験項目を作った場合にはプログラムテストと同様、どうしてもそれまでの製作側としての経験だけが盛り込まれたテスト項目になり、受け入れ側として必らずしも満足のいかないテストになることがある。従って、ここで製作と監査というものの性格を明確に分離して監査側から見た受け入れテスト項目を充分に確立しておくことが大切である。これにはシステムを受け入れる前にあらかじめ数人の人間をメーカー側に派遣してシステムの特徴、ハードウェア上の種々の問題などをチエックして、受け入れ側でテスト項目を作ることが必要である。

4.2.7　プログラムの試験

システムの建設チームからエンドユーザーへの受け入れの段階にはプログラムテストを行なう。

バッチシステムの場合には前述のようにユーザープログラムが単体で走る場合が多くあまり問題はない。

オンラインリアルタイム．システムの場合は多重プログラムをユーザーのレベルでも採用しているので、プログラム作成時においてシステムにお

ける動作のあらゆる論理的な可能性の組み合わせを予測することは全く不可能である。すなわち、一つのユーザープログラムがその実行をすべて終えるまでコントロールを保つことが少ないし、同一プログラムを同時に使うリエントラント型に作られている場合もあり、さらに入力も一般に多様でランダムである。従ってテスト時には予想しない多くの誤りが発生することが考えられる。

オンラインシステムでは単純な誤りもプログラムの複雑な動作の組み合わせの陰に隠れて、さらにそれがタイミング関係によってある複雑な時系列的な論理関係が成り立つ時にだけ発生することがある。　しかもこのような誤りはその発生するタイミングが生じるチャンスが非常に少ないため、バッチプログラムと異なって再度実行した時正常に通ることもあって誤りの再現性が非常に少ない。

従って、プログラムテストはかなり長時間行なう必要があり、また実際にシステムが稼動すれば誤りを分離することが困難になるので、オンラインリアルタイム．システムでは必要なあらゆるテスト手段を含むように設計の段階からシステムが組まれていなければならない。

4.2.8　テストプログラムの効用

さて、コンピュータシステムとはハードウェアとソフトウェアとの混合体であり、テストをするための種々のテストプログラムが必要となる。このテストプログラムは機器を受け入れた後に作るのではなくて、その前に、メーカーがまだ機器の製作を行なっている段階に同様の他の機種を使って各種のテストプログラムを用意しておくことが大切である。

ＯＳは必らずしも個々のオンラインシステムに適するものではなく、これを若干改造したり、場合によっては全く新しくシステムに合わせてＯＳ

を作ることがある。このような場合、管理プログラムとユーザープログラムは同時に開発されることになるので、システムのテストはシミュレータを使用して行なう。まず管理プログラムをテストする時はユーザープログラムのシミュレータを作成し、またユーザープログラムをテストする時は管理プログラムのシミュレータを作って相互に完全にデバックする。勿論このシミュレータは実際のものに比べて機能は簡略化されているが、このシミュレータのもとで相互にテストする必要がある。

次にデータをジェネレートするテストプログラムが必要である。これはテストの初期の段階でデータの正確さと速度を上げるために模擬的な入力のトランザクションをテープあるいはディスクに作成するためのものである。これによって一つ一つのトランザクションを人間が端末機器より入力してテストするよりもずっと短かい時間で正確にプログラムのテストを行なうことができる。これをファーストタイム・シミュレーションと言い、テストの能率、受入試験の能率をあげることができる。

その他にマクロ命令をテストするプログラムであるとか、あるいはデバッグ用のプログラムを作っておく必要がある。プログラムの誤りの発見を容易にするため、各種のダンプ、トレース、あるいは一般にスナップショットと呼ばれているデバッグ用のルーチンが既にOSの中にも用意されてはいるが、やはりユーザーの特殊な業務に合った個有のダンプ、トレース、スナップショットのデバッグ用ルーチンを用意しておくとテストの時間をかなり短縮することができる。

また、テスト結果は大量のダンプリストのような形をとる場合もあるので、あらかじめこれらを分析するプログラムを準備しておく必要もある。

4.3 移行計画

4.3.1 移行計画の必要性

開発の進められてきたシステムがいよいよ実施準備の段階に入る時、このシステムの実際の業務を担当する部署が中心になって使用するに際し種々の問題が生じる。すなわちシステムを使用する部門における受け入れ態勢の準備と言う点を充分考慮する必要があり、これらの最終的なエンドユーザーと密接な関係を保ちながら一時的にも現在の業務の流れを低下することなく、新しいシステムがその活動上の諸機能を果すことができるようにシステムの移行計画を開発することが大切になる。システムの移行計画は情法システムにとって本質的な意味を持っている。すなわちシステムの開発や設置には多くの組織や機関が参加、関連しており、効果的なシステム移行計画を作成するにはユーザーのトップレベルの管理者が全面的に参加しなくてはならない。時にはシステムに置き換えられようとする手作業とは別個の全く新しい業務の方式を計画しなければならないこともあり、この場合には新旧両システムを並行して動かすことができるので、システム移行に関する混乱は最小限に食いとめられる。しかしまた、移行計画が適切でなければかえって混乱が大きくなることもしばしば起こる。

新しいシステムが移動してシステムのオペレーション開始が宣言された時、システムは開発から運用へ移ることになる。コンピュータシステムのような複雑なものではこの切り換えはターン・キー的な一度のものでなく、ある期間をかけて徐々に進行するものであり、これがコンピュータシステムにおける移行計画を非常に重要なものとしている。

すなわち、最初からある程度のシステムパフォーマンスを達成するには、開発者とユーザーが共通の目的を持つパートナーとして共同作業を通じてシステムの移行に伴なう種々の問題を解決することが大切であり、システムの完成自体もいままでより長い期間が必要となる。

この移行計画の期間と言うのは、今までのデータ処理システム、または手作業によるシステムが次第に消滅して、それにかわる新しいシステムが出現するまでの一定の期間であり、この期間に業務の流れが低下しないようシステム移行計画がシステム開発作業の一環として立案されなければならない。

要するに移行計画とは、コンピュータシステムが使用される前の状態から、最終的な定常運用へ移るまでのトランジェントな段階の組織、データ、運用などにおける計画であることを充分認識して設計段階から考えておく必要がある。

4.3.2　移行と組織変更

(1)　業務組織変更に伴なう諸問題

移行に伴なう人間の組織の変更は、直接業務に従事している人たちの異動に関係してくるので、日本のような社会ではあまり公然と打ち合わせや討議が行なえないような性質を持っている。一般には人事部門や組織管理部門とコンピュータシステムを受け入れる対象業務部門、及びコンピュータを運用する部門の管理者らによって打ち合わせが行なわれることになる。この問題は事前に評価した組織面、要員面の改善が、実際にシステムを設計してみて予測どおり行なえるか否かを検討することになる。

組織について言えば、新システムは本質的あるいは大規模な制度や手

続きの改善を伴なう場合があるので、そこには組織上の職制の変更を伴なうのは当然であって変更内容、変更時期、またはその方法について組織幹事の専門部門との打ち合わせが必要である。

また要員面については、機械化による人員の削減の問題や要員の質の変更の問題が生じてくることが多く、その実施時期や実施内容については最終的に人事上の問題になってくる。一般的に新しいシステムが稼動し始めても、その組織や要員が従来のままであれば、それはコンピュータシステムを導入した意味が全くないことになるので、当然大なり小なりこう言う変更の問題は生じるものである。

マネジャーとして忘れてはならないことは、こう言う組織や人事に伴なう変更を行なう時には当然組合との問題が生じてくると言うことをあらかじめ認識して、充分な手配をしておく必要がある。このような問題が生じたためにシステムの稼動が当初の予定よりもかなり遅れ、そのためにせっかくのシステムが有効に利用されなかった事例も多くある。

(2) ユーザーによるシステム評価

移行時にはユーザーが当然システムを直接自分で使用するようになるので、ユーザーから見たシステムの評価がなされることになる。

ところが開発側としては、完全にシステムが稼動していると考えていても設計段階におけるユーザー側の要求を誤解していたり、あるいは完全な形で現在の業務を置き換えることができなかった場合、それに対するユーザーの拒絶反応を引き起こす可能性がある。このような問題は主として移動時における教育の問題と関連しているので、単なる組織変更の問題とだけ見ないで、教育の問題と関連づけて考えなければなら

ない。特に日本の場合にはコンピュータシステムを導入する場合従来の筆記作業からタイプライターとかキーを扱うような作業に切りかわってくるので、アメリカにおけるよりもこの問題はより深刻な形となってあらわれてくることが多い。

(3) 並行作業

前述のように移行計画と言うのはシステムが全く使用されないゼロの状態から定常作業に移るトランジェントな状態であるから、業務の性質によっては従来の人間を中心にした組織の運営を行ないながら、もう一方ではコンピュータシステムが完全にユーザーの満足いくレベルに達するまで、並行的に同様の作業を同時に進行させなければいけないことが往々にして起こり得る。従ってこの場合、暫定的な要員の手配を考えておかなければ切りかえに伴なう混乱を大きくする原因となる。

4.3.3 業務の引き継ぎ

(1) オリエンテーション

移行計画と言うのは、別の見方からすれば開発チームからエンドユーザーへの業務の引き継ぎと言うことにあたるので、まず最初にユーザーに対するオリエンテーションの計画をたてる必要がある。オリエンテーションはユーザーにとってシステムに関するあらゆる今後の学習やトレーニングの母体や基礎を作ると言うもので、ユーザーのシステムに対する基本的な考え方を確立するもとになる。この点でオリエーションは訓練とは別ものである。

オリエンテーションの具体的項目としては

- システムの目的とするところの解説
- 組織上、運営上の変更事項の列挙

- 新しいシステムの設置運用における責任部門の明確化
- 移行、設置計画の解説と移行時における中間業務の明確化
- システムとの交換すべき情報の説明
- 新しいシステムのコスト面、利益面の説明

(2) 移行計画の立案の主導権

移行計画はシステムの開発側、マネジャー、エンドユーザーの共同の作業で立案されなければならないが、指導権は最終的には使うユーザーが持つべきである。これは用務の内容を良く知っていると言う点で、移行時における種々の諸問題を摘出することがユーザー側でなされることが望ましいと言う観点から言えるのである。

(3) ユーザーとオペレータの仕事分担

移行計画の段階では、定常作業と異なるため通常のオペレータの仕事とユーザーの仕事が完全に分離できないことが往々にして生じる。従って移行時におけるユーザーとオペレータとの仕事の分担というものがやはり計画の中で充分考慮されてなければならない。

(4) 移行計画の立案とその承認

移行計画は模然と行なわれるのではなく、充分な計画性に基づいて行なわれ、一応立案文書など明確なドキュメントとして残しておき、これをトップマネジャーに承認を受けてそのもとで計画を実行することが各方面でのトラブルを少なくする要因となるであろう。

4.3.4 移行に伴なうデータの収集、蓄積、交換

(1) データの収集方法

新しいシステムの場合でも、取り替えシステムの場合でも、新しいコンピュータシステムにはデータが蓄積されていない。従ってこのシス

テムを使うためには、徐々に端末あるいはセンターでデータを集積し、システムのデータバンクを作っていかなければならない。

ところがこのトランジェントの状態では、場合によっては過去のデータとつき合わせをするような必要があるが、もとのデータがない場合ではつき合わせの対象がないので、定常運用のプログラムではすべてエラーデータとしてはねられデータが蓄積できないような状態が生じる。従って、このような状態に対処する特別な移行時の立ち上がり用のデータ収集プログラムと言うものを準備しなければならない。これは非常に特殊な事例であるが、一般に初期にデータを集める場合、できるだけ過去のデータをあらかじめパンチカードなどで準備しておき、センターで一括してデータバンクが作れるように移行計画を作ることが望ましい。ただしこの場合、データの収集にあたり大量の人手、または作業を要するので、これに対する配慮をしておく必要があろう。

(2) データコンバージョン

運用開始に伴なっては前述のように初期データを作成する必要がある。現在手作業などで行なわれている業務の各種のデータをコンピュータシステムに入力することのできる形、例えばパンチカードなどに変換することは大変な作業である。この労力を過少評価しないよう注意する必要がある。また、プログラムの作成段階では入力データのフォーマットを変更することが往々にあるので、このデータの変換作業はプログラム側のフォーマットの確立をできるだけ待った方がよい。

このデータ変換に際しては正確さが問題であり、対策としては注意深くチェックする以外にあまりない。また、コンピュータによって自動的に変換できる所と人間の手作業によらなければ不可能な所を見極め

両作業を適切に組み合わせて行なうことも充分考慮しなくてはならない。

(3) テーブル類などの取扱い

移行時には業務の移す範囲が徐々に広がる場合がある。給料計算を例にとると、給料計算を新システムに移行しながら行なう場合には対象とする人の数を徐々に増加させていくことがある。これに伴なってファイルのデータばかりでなく、プログラム内の種々のテーブル類も徐々に変換していかなければならない。通常テーブル類はサポートプログラムで準備されることが多いので、業務を拡張するに従ってテーブル類を何回も書き直していく状態が生じる。このようなテーブル類の書きかえに要する時間は案外多いので、移行計画時にはテーブル類を変換するためのコンピュータ使用時間を見込んでおく必要がある。

4.3.5 移行時の運用

(1) 平常運用との相違

移行時は最終的な平常運用と異なったコンピュータの使い方をしなくてはならないことを充分に認識して、コンピュータの使用時間の問題及び組織運営について、あらかじめ移行時として別の形式で運用などの計画を立てておかねばならない。

(2) 時間帯の割り振り

コンピュータの使用時間の問題をとり上げれば、最終的にあるプログラムの実行をするよりも、ユーザーの訓練であるとか、初期データの収集、プログラムエラーのデバックなどの時間が必要である。プログラムエラーに関しては当然これがなくなった時点でユーザーへ引き渡されるべきものであるが、やはりシステムテストの段階を経ても移行時

に若干の複雑な組み合わせによるエラーの発生はあらかじめ覚悟しておかなければならない。従ってこれが生じた時の早急なる手当てのためのコンピュータ使用時間も割り当てておく必要がある。

また、移行時にはシステムの構成を若干かえて、システム全体の効率を上げるようなチューンアップの問題も生じるので、これに対する時間も用意しなければならない。このように平常運用とは異なって以上の問題にかなり大量のコンピュータ時間を割り振っておかねばならないであろう。

これらは当然移行が進むにつれ低減するものであるから、あらかじめプロダクション．ランと、このような移行に伴なう諸問題を解決するためのコンピュータ時間との割り振りを時間的推移で表わすタイム．チャートの製作を移行計画の一部として入れておくことが望ましい。

(3) 端末機器の設置との関連

また、オンラインシステムの場合には、一度にすべての端末を切りかえることは、人間の熟練度の問題、訓練計画との兼ね合いにおいて不可能なことが多い。従って、バッチシステムで対象範囲を徐々に広げるのと同様に、リアルタイムシステムでも端末の数を徐々に増して移行していくということが生じる。

端末の数が異なることによってオペレーションのやり方が変わってくることも当然あり得るし、オペレータの簡単なキーの取り扱いなども端末の数に左右されることが多いので、このように端末を徐々に増加させることを考えた場合には、この期間におけるオペレーション．マニュアルの整備が重要なことになってくる。

(4) 移行時の運用組織

移行時においてはシステムの開発者と真のエンドユーザーが端末機器及びセンターの機械を共有して、ともに最終的な定常運用が早く行なえるように努力するわけであるから、人的にも、組織的にも、開発者とエンドユーザーが共存できるような運用組織を考えておかなければならない。

4.3.6 取り替えシステムの場合の問題点

(1) 一般問題

取り替えに伴なって大規模にシステムの変換を行なう場合には、新しいシステムを開発するのと同様の費用と期間を要する場合が多いので、この点に関し充分にマネジャー及びトップに徹底させておく必要がある。通常、上層部はすでに一度コンピュータシステムを導入しているので、システムを取り替える場合にはあまり労力を必要としないであろうという観点から、取り替えの期間であるとか、それに要する開発費用を充分に考慮してくれないことが多いので、新システムを導入するよりもかえってシステムの取り替えの時の方が種々のトラブルが生じて、情報システムが失敗することが往々にして起こり得る。

一般に、取り替えシステムの場合にはプログラムの変換からデータの変換、その他端末機器の変更などについて一度経験があるという意味でかなり安易に考えやすいが、むしろ問題は深刻で複雑な場合の方が多いので、これに対する移行の期間も充分に取っておく必要があり、また充分な費用を見込んでおくことが取り替えシステムを設計するマネジャーの重大な責務と言えよう。

(2) プログラム変換の必要性

すでに古いコンピュータシステムが稼動しており、数年後に新しいコンピュータシステムへ取り替える時に生じる問題として、まずプログラムの変換の問題がある。最近はコンピュータ・メーカーからファミリー・シリーズとしてのコンピュータが用意されているので、より上位の機種に対するコンパティビリティはかなり保証されているように言われているが、実際には同じファミリー・シリーズを使ってもかなりの細かい部分でプログラムの変換を行なわなければならない問題があり、また新システムに取り替える頃にはコンピュータの能力はたいてい大巾に増大しているので、ユーザーから新しい機能を追加してほしいと言う要求が当然生ずるであろう。この場合、プログラムの変換の作業がどうしても必要になる。

バッチシステムの場合には、最近はＣＯＢＯＬなどのより高度のレベルの言語を使用していることが多いので、プログラムの変換に伴なう問題は少ないが、オンラインシステムの場合には主として効率やコア容量の点から開発言語としてアセンブラを使用することが多い。
この時にはプログラムの変換にかなり大量の人工と経費を要することを考えておかなければならない。勿論、最近開発されたＰＬ　のように、オンラインシステムでも使えるような高度のレベルの言語もないわけではないが、今後新システムを考える場合、常に取り替えの問題を考慮に入れて開発の使用言語を選択する配慮が、システム開発の責任者としての重要な問題となってくる。

(3) データの変換

また、単にプログラムだけでなく、ユーザーの要求によって機能追加が行なわれた場合、当然データのフォーマットの形式、ファイルの形式などが変わってくることがある。従って、プログラムの変換だけでなく、旧システムのデータをいかに新しいシステムのデータに変換するかという問題も、取り替えの設計段階で考慮しておく必要のある問題であろう。通常は変換用に特別なサポートプログラムを作って、これによって古いファイルから新しいファイルへのデータの変換を行なうが、この時にはデータのコンパティビリティについて充分注意しなければならない。すなわち、古いデータに含まれていないアイテムを追加する時、新しいアイテムを古いデータに詰め込む作業が必要になってくる。　　当然これは人手作業を伴なって行なわなければならないので、このための埋め込み用のサポートプログラムも必要になるであろうし、また、データを準備するための手作業の経費も考慮しておかなければならない。

(4) オンラインシステムの取り替え

24時間稼動しているオンラインシステムの場合には、旧システムから新システムへ取り替える場合にサービスの中断を伴なうことがある。このようなケースは、主として24時間操業を前提とするプロセスを制御するオンラインシステムに生じる問題であるので、サービスの中断が重大な事故を起こしたり、あるいは製品の品質を落したりする事態が生じるかも知れない。従って、このようなオンラインシステムでは中断期間における計画、すなわち中断の期間において一時的に多くの人手をかけな作業を行なうのか、またはそのようなことができないシステ

ムでは、制御対象になっているプロセスがコンピュータ以外の何かの原因で停めなければいけないような時期を見計ってシステムを取り替えるよう計画しなければならない。

在庫管理型のオンラインシステム、すなわち座席予約とか、銀行のシステムの場合には操業が夜中の時間帯で切れることがあるので、この時に切りかえることが可能であるが、それでも一度に全システムを切りかえると種々の点で混乱を大きくするので、切りかえの対象範囲を徐々に拡大していくというような配慮が必要である。たとえば新しいシステムにプログラムエラーがあった場合、すべての対象を一度に切りかえるとエラーによってサービスが中断した時の影響度ははかり知れないものがある。

オンラインシステムの場合には、システムを取り替えた場合、当然端末機器は旧来のものをそのまま使用することが多いので、端末機器からコンピュータシステムへ二重の配線を行なって、ある一時期を期して機器の切りかえを行なうこととなる。これには電話局で新設の局ができた時に切りかえる方式と同じような考え方を導入しなければならない。一方、新システムへ取りかえる時に、機能追加などの要求によって端末機器も同時に取りかえる場合がある。この時は全く、新旧両システムが並行に動いているので技術上の問題はむしろ少ないが、使用するユーザーの側から見た時の問題点の方が大きくなってくる。　すなわち、端末機器の種類が変わったことによって当然それに伴なうオペレーションの変更があり、かなりの訓練期間を経ても古いオペレーションの慣習が残っているために、切りかえ時のオペレーションミスを誘発することが多い。このようなこのような面からも、端末機器の種類を取り

替える場合でもできるだけオペレーション上は従来のものと変更のないような鍵盤配置であるとか、機器の配置を考慮しておく必要があろう。

4.3.7　マニュアルの保存とその整備

(1)　メーカー提供のハードウェア及び基本ソフトウェア

一般にシステムを開発する段階で、メーカーからハードウェア、及びメーカー提供の基本ソフトウェアのマニュアルが渡されている。これが運用段階に入ると、オペレータやユーザーの一部に同様なものが支給されねばならない。かなりの部数を必要とするので、この移行段階の間にこのようなものの手配をしておく必要がある。

(2)　ユーザー開発プログラム

ユーザー開発のプログラムのマニュアルも、当然システム開発の各段階でドキュメントとして残しておかなければならない。ここでは、最終的にアップデートされたものをチェックして、ユーザーに引き渡す時にきれいな形で印刷した最終版を移行段階で準備する。

(3)　テストプログラム及び診断プログラム

さらに移行段階で種々のテストを行なったり、特別なケースに対するテストプログラムを作ることがある。大部分のテストプログラムはシステム設計の段階で作られているのが当然であるが、設計段階で見落しのあったようなテストプログラム、あるいは故障診断プログラムなどを移行段階で作ることはしばしば生ずる。この様なプログラムは作成の期間が限られるので、ドキュメンテーションがおろそかになりがちであるので、この整備をしておく必要がある。

4.3.8　運用手順書の作成

(1)　作成時期と内容

ユーザーから見た場合のコンピュータシステムの運用手順は、当然システム設計の初期の段階に設計の一つの条件として作られるべきである。しかし、開発の途中で、開発側の要求で運用の手順を一部変更することがある。したがって移行の段階に、初期の運用手順書で不備な点に手を加えて改定しなければならない。

また設計の段階では完全にカバーしきれなかったエラーが生じた時の色々な手順についての特殊な手順書が必要となる。これは異常処理対策の手順書ともいえるものである。特にシステムの初期の段階では、プログラムの細かな点までの設計が十分進んでいないため、ある条件でプログラムエラーが生じた時のエラー表示までを、十分設計書の中に書き加えておくことができない。したがって、開発が終った後の移行段階に、プログラムから発生する種々のエラーの一覧表を完備してユーザーに配布し、エラーが生じた時のユーザーの手順もあわせて整備しておく必要がある。

(2)　運用者の熟練度との関連

移行の初期の段階では、運用者も取扱いになれていないため、簡単なエラーであっても一々エラーの処理マニュアルを参照することになる。

移行の後期の段階では、運用者がシステムの取扱いになれてきて、常に起すようなデータインプットのミスなどについてのエラーリポートは、マニュアルを参照せずに、覚えてしまっていることが多い。

この様な時、膨大なるエラー手順書を一々くるのは大変なため、移行の前期の段階と後期の段階では、この様な手順書の作り方を再度チェック

しなければならない。具体的には、移行の初期段階でのドキュメントは、オペレーションのシーケンスや、あるいはアルファベット順に種々のドキュメントを整備しておく必要がある。後期の段階、すなわち定常運用に移る前には、使用される頻度、及びプライオリティに応じて2種に分けた手順書を作っておくことが、以後の定常運用を潤滑にする一つの秘訣であろう。

(3) 手順書の改定手続の設定

移行の段階で、ユーザー側から見た若干の改定はできるだけ移行の期間に洗い出して解決しておくことが望ましい。しかし、定常運用に移ってからもこのような改定を生ずる可能性があるので、移行の期間に手順書を改定する手続を決めておくことが大切である。

というのは、一度定常運用に移ると使用するユーザーの数が非常にふえるので、手順書を改定するということは、システムの運用を混乱させることになる。避け得ない改定に対する手順、すなわち、変更の時期、改定における前後の取扱の変更点、などのように周知徹底するかの問題を組織上明確にしておかねばならない。

(4) 一般的注意事項

マニュアルやドキュメンテーションは、開発段階ではシステム開発者に便利なように書かれているが、必ずしもこれは最終的に使用するユーザーから見てわかりやすいドキュメントであるとは言えない。

したがって移行期間中にユーザーや、オペレータが、マニュアルや運用手順書を使用して、不備な点を指摘させて、それによって使いやすいものに改善していくことが、マニュアル及び、運用手順書の作成の最大のポイントである。

4.4 定常運用

4.4.1 マシンスケジューリング

(1) バッチ処理システム

定常運用で一番大きな問題は、マシンスケジュールをいかに立てるかということである。これはバッチシステムと、オンラインシステムの場合ではかなり異なってくる。

バッチシステムのマシンスケジュールは、日によってワークロードがかなり違うため、運用管理者は、通常1ケ月単位で大きなマシンスケジュールを作る。　　さらにそれをウィークリーベース、ディリーベースで修正していくことによりスケジュールを立てる。

(2) オンラインシステム

オンラインシステムの場合、使用される対象が1つの業務ジョブの場合が多いため、通常昼間の時間帯をユーザーのオンラインシステムとして使用し、この間にデータの収集、アップデートを行なう。夜間の時間帯の使用法は、対象とするシステムにより異なるが、1つは業務の一部として昼間集めたデータの設計をとり、レポートを作成する業務や、集めたデータをバッチ的に変換する作業を行なう。その他、業務とは無関係に計算機の運用の面から、機械の使用効率の統計をとることや、ファイルに空きが生じた時の詰め合わせなどに使用する。

(3) 予備機のスケジュール

オンラインシステムでは異常時に備えて予備機を持っているのが普通である。この予備機は完全に遊ばせておかず、通常はバッチ的業務を行なわせる。この場合スケジュールをバッチ業務として立て

ておくと、オンラインシステムがダウンした時、予備機に切替り、バッチ業務が使用できなくなった場合に、それにどのような影響を与えるかを、あらかじめ解析して、十分余裕を持ったスケジュールを立てておかねばならない。

あまり予備機のスケジュールを効率よく割当てておくと、オンラインシステムから切替った時に、予備機のスケジュールが24時間をオーバーすることがある。バッチ的な業務であるから、その一部を切離すことは可能であるが、一度予備機にマンスリー又はウイークリーの計画を立ててしまうと、いかにバッチ業務といえども、最終リポートの期限があるため、これに抵触することも生ずる。
このため、予備機があれば非常に便利がよいという程度の業務にとどめておくことが望ましい。当然これは費用と危険性との兼合いで判断すべき問題ではある。

(4) 予防保守の時間帯

コンピュータシステムに限らず、全ての機械では当然予防保守の時間帯をとらねばならない。コンピュータの場合には、マンスリーベース、ウイークリーベース、デイリーベースの予防保守の時間帯をメーカー側で指定しているので、これに合わせて業務を決めておく必要がある。

通常デイリーベースの保守は、簡単なテープのヘッドの掃除であるとか、プリンタなどの周辺装置の動作の確認やメカニカル部分の掃除が多く、普通30分程度である。ウィークリーベースでは約3時間から5時間程度の時間をとり、テストプログラムを使って種々のマージナルテストを行なう。マンスリーベースの保守は、通常1日程とり、かなりの部分のオーバーホールを考えておかねばならない。

これらの細かなことは各マシンによってかなりバラツキがあるため、メーカーから出される保守マニアルを参照するか、メーカーから派遣されているシステムエンジニアと相談して、マシンスケジュールの中に予防保守の時間帯を組込んでおく必要がある。

当然これは予防保守であるから、システムの運用が異常になった時には延ばすことは可能である。しかし、これをあまり多く行なうとかえって定常運用を妨げることになる。

4.4.2　ジョブアカウンティング

(1)　バッチ処理システム

マシンスケジュールと関連して、バッチ処理のセンター業務システムでは、ジョブアカウンティングを省くことはできない。通常、センター業務ではジョブアカウンティングによって機械の使用費用を取ることになるので、これが完備しているシステムを使うことによって、チャージの問題を自動化することができる。

普通は、計算機のオペレーティングシステムに付加されている、ジョブアカウンティングシステムによって行なうことができる。
しかし、これだけでは不十分な場合が多いので、ユーザーが若干これを改定して自分のシステムに合わせたジョブアカウンティング．ルーチンを作ることが望ましい。

(2)　オンラインシステム

オンラインシステムでは、ジョブアカウンティング対応するものとして、夜間の時間帯に昼間入ってきたメッセージのジャーナルをもとにして、種々の効率の測定や、トランジェントエラーの回数などの統計をとることにより、これを以後の定常運用の資料とする。

当然これはシステム設計の此階で考えておかねばならない問題である。

4.4.3　勤務体制と作業形態

(1)　勤務体制の問題点

運用面でマネジャーが考えておかねばならぬ問題は、計算機を1日に使用する作業量から算定して、オペレータの勤務を決めることである。

通常計算機の勤務体制はシフト制がひかれており、レンタルベースの料金が違ってくるため、それとの関連で勤務体制を作っておく必要がある。これはオンラインシステムの場合特に問題になり、日勤の時間帯と夜勤の時間帯の作業配分をよく考えておかないと、単に時間的な問題だけで勤務体制を作ると作業の密度が異なるので、深刻な労働問題を引起す可能性がある。

具体的には、オンラインシステムでは、日勤の時間帯には異常が起らぬ限り、オペレータは何もすることがないようにシステムが設計されているのが普通である。むしろ夜勤の時間帯には、テープを取替えたり、統計をとったりということで、時間の割には作業の密度が濃いと考えておくべきである。

(2)　正常運用時のオペレータの作業軽減

上記の問題と関連して、オンラインシステムも含めて定常運用時でのオペレータの作業を軽減するようにプログラムの設計をしておくことは大切である。

バッチ処理の場合には、多くの場合、テープ取替えや、プリンタ用紙の取替えなど、オペレータの定常的な作業は比較的ひんぴんと起る。そのため、最近のスプーリングのようにカードを一時的に積むような形での軽減しか考えられない。

オンラインシステムではシステムの設計時に十分配慮することにより、正常時にはオペレータはほとんど何もしなくてよいように考えるべきである。

(3) 正常時のオペレータ作業の活用

このようなシステムでは、昼間の時間帯ではオペレータは単に温度の低い計算機室の中で坐っているだけになる。そのため、オペレータの部屋を別途に設け、ＩＴＶなどを活用して、その間に後に述べるテープ、ディスクなどの機器管理業務を行なわせたり、あるいは簡単なプログラムの小改修を行なわせるような作業を考えておくと、オペレータを教育して次にプログラマーとして使用するような人事構成を考えることができる。これは日本のような社会で、職種の変更を伴なわなければ給料面での処遇を考慮できない給与体系では、作業員の士気を高める面からも重要である。しかしこの問題は会社の就業規則と関連してくるため、オペレータの就業規則などを設定する時十分配慮しておかねばならない。たとえば、オペレータの業務範囲の中にプログラムの小改修などを突如として飛び込ませると、労働問題と結びつくことになる。

(4) 作業管理

機械化されたシステムであるから、作業の管理もできるだけ計算機のジョブアカウンティングなどに自動的に反映されるように、システム設計すべきである。しかし、テープの管理や、使用したプリンタ用紙の管理など、かなり人間作業にたよらないとできない作業もある。これはあらかじめ記入しやすい形の台帳を準備して、オペレータが使用時間とか、枚数とか、簡単な数値を書込めるようにするべきである。

4.4.4　システムの調整

システム運用時にシステムの調整を行なうことが必要である。

(1)　機器及びシステム機能の評価

システム運用中の機能の評価を客観的に行なうのはむずかしい。しかし、プロジェクト、スーパーバイザは、定期的にユーザーに機器及びシステム評価を求めるべきである。

その場合の評価項目を下に列記する。

1.　システムの出力に対するユーザーの満足度

2.　データ収集と入力方法

3.　企業目的に対する直接的寄与度

4.　企業目的に対する間接的寄与度

5.　現在システムに対する不満足点

(2)　システム機能の改善

プログラマーなどは、オペレーションの運営状況を観察して、効率を高めるための改善方法を見つけることが多い。それは、アプリケーションプログラムの1部分を修正するだけのものから、運用手順の変更を伴なうもの、システム全体に影響するものなど種々のレベルがある。これらの改善は、現在運用している作業を変えたくないという精神的な面や、変更作業の人工の問題などがあり、抵抗を受けることが多い。

このような場合無理に変更をすべきではないが、新しい方式の利点は、十分にデモンストレーションし、又スーパーバイザのほうもそれを十分

検討すべきである。

(3) 運用手順書との関係

なおこのような調整を行なったとき、これが計算機の運用手順を変えるような場合には、3.3.8で述べた運用手順書改定の手続きを踏まねばならない。

4.4.5 機器及び備品管理

(1)テープ．ディスク等の管理

まず計算機を使う場合には、常識的に磁気テープ、ディスクパック、アウトプットのプリント、コンソールタイプライタのシートなどを管理する管理方式を作っておかねばならない。テープ．ディスクはあらかじめそのラベルのところに固有の番号を書込んでこれを管理する形が現在のＯＳではとられているため、簡単な意味での番号管理は自動的に行なえる。しかし書かれている内容に関してまでは管理できないので、これは台帳を作り、オペレータが簡単に記入できるような形で番号管理を行なう手順をオペレーションマニュアルに組込んでおく必要がある。

(2) データセキュリティ対策

次に、非常に重要な基礎データがオペレーション上のミスによって壊れないように、データセキュリティの対策を番号管理と合わせて考えておく必要がある。これは単に番号管理の問題だけでなく、テープ

ディスクの格納場所との関連で、人間工学的配慮が必要である。

その他、データを2重に書込んで、一方はロックしたロッカーに入れておくなどの対策も考えられる。

(3) 消耗品の数量管理

カード、ラインプリンタ用紙、タイプライタ用紙などの使用数量は、オペレーションを行なっていると、月間平均値がわかるため、これを基にして資材購入手続をとる。但し、オペレーションの最初のころは、消耗品の使用頻度がつかみにくいため、あらかじめ多めに購入しておくことが必要である。

この問題はオペレータの台帳管理との兼合で、統計を出すような作業形態を考えておかねばならない。

4.5 異常障害対策

4.5.1 異常障害の分類と一般対策

(1) 端末オペレータの操作ミス

端末機器のオペレータの操作ミスは必然的なものと考えられる。前述のように新システムへ移行し、端末の取換えや操作手順が変更になった場合には今までの慣れによる操作ミスが続出するであろう。そうでなくとも例えば誤った列車の予約切符を発売したり、預金通帳の口座を誤って入力したり、タイプライタのキーを打ち間違ったりすることもある。この種の人間の入力ミスに対してシステム側では、入力手順が正しいか、入力値はある定められた範囲にあるか、などはプログラムによってチェックすることは容易であるが、論理的に正しい入力データであれば誤ったまま処理が進み、回答を見てから誤ったことに気づく場合もある。これに対してはオペレータに入力の再確認を行なわせるエコーチェック方式などがある。

(2) 回線の障害

通信回線は伝送路のS／N、信号レベル変動、周波数特性など伝送誤りを生じる要因は多く、また回路が一瞬切断したりしてインパルス性雑音が発生すると送信中のデータにビット誤りを生じたりする。この誤りを検出するには種々の方法があり、たとえば送信符号を訂正可能な符号としてある程度の誤りなら自動的に訂正させるか、自動的にあるいはオペレータによって再送を行なうことなどがある。

(3) 周辺機器の障害

周辺機器の異常は入出力の終了割込み時の情報によって中央装置に知らされるし、また入出力が終了しても割込み信号が戻ってこない時に

は周辺機器の動作時間をタイマーによって監視チェックして異常処理ルーチンを作動させる。

(4) 中央装置の障害

中央装置に生じる障害には装置そのものの故障や回路の誤動作によるハードウェア的なものとプログラムやデータのエラーによるソフト的なものがあり、この故障は他に大きな影響を与える。プログラムの続行が不可能になった場合、まず処理の中止が可能なプログラムであれば途中で処理を放棄し、またソフト面で障害対策がとれない場合は一たんシステムを停止させ再度プログラムのロートからの再試行を行なうこともある。

オンラインシステムではファイルのメンテナンスやトランザクションの保護に注意し、デュアルシステムやデュプレックスシステムでは正常な機器へコントロールやデータなどを手渡し、機器の構成変更を行なったりする。

(5) オーバーロード

これは負荷が急激に増え過負荷状態となった場合で、通信回線と中央装置との接続を即時性にすれば、過負荷時には処理能力以上の入力情報は全く処理されずに捨てられてしまうことになる。一般の問合わせ応答型システムでは待時式にして、過負荷時には処理の待ち行列を作り入力に対する処理を必ず行なえるようにしてあるが、不当に長い待ち時間にならぬよう制御する必要がある。

(6) オペレータの誤動作

センターのオペレータの誤動作としては機器の操作手順ミス、テープ、ディスクのセットミス、コンソールよりの指令ミスなどがある。オペ

レータの操作はシステムに対して司令部に相当するものであるから人的ミスの完全防止はなかなか困難である。

対策としてはまずマニュアル類、特に異常状態や特殊処置に対するマニュアル類を完備してオペレータが混乱しないようにすることが必要である。誤操作に対してはキーボードの配置に人間工学的な処置や操作パネル面上の手順マーク、不注意によるスイッチ操作の防止のためのカバー等の工夫が考えられる。オペレータの誤操作がシステムに重大な影響を与えるという問題に対しては、例えばプログラムの全体量が増加してもそのような誤りを入力の所でプログラム的にキャッチして処理の重要な所にこの誤ったデータを持ち込まないようにするエラーチェックの方法がある。このエラーチェックとしては2つの方式がある。1つはフォーマット・チェック（Format check）で入力データの様式のチェックを行なう。もう1つはバリディティ・チェック（Validity check）で、これはオペレータの操作やデータに関してその数値や論理性について予想される範囲にあるかどうかをチェックするもので、システムの内容との相関性においても考えられなくてはならない。

(7) プログラムエラー

プログラムエラーは十分デバッグしてテストを行なった後でも、システム稼動後になってある特定のタイミングによって出現してくることが多い。これによって出力の異常だけではなくプログラムの異常となり他のプログラムや大切なデータを破壊したり、あるループに入り込んだままぬけ出して来なかったり、マルチタスクの場合などはデッドロック状態に陥ったりする。このような異常状態は制御プログラムや

ハードウェアによるメモリープロテクションなどによってある程度防止は可能であるが、デッドロックなどはプログラム作成段階でも十分注意しておかなければならない。

4.5.2　異常の記録と処置

(1)　記録と報告

異常が生じた時は後々の異常の原因の追求や診断のため機器の状態を記録しておく必要がある。すなわちコアメモリやレジスタ、トランザクションの内容や端末の使用状態などを磁気テープなどに保存しておく。また生じた異常状態について関係ある箇所に連絡したり、後々の対策のために報告書を作る必要がある。しかし異常状態が生じた時はオペレータは混乱しているので文書にまとめることは無理であるので、一定様式の報告書用紙を準備しておき項目チェックや簡単な記述で済むようにしておかねばならない。

(2)　連絡体制と連絡箇所

オンラインシステムでは通常1ケ所のセンターと多数の箇所にまたがる端末とを結んで運用しているので、種々の異常事態に対してどのような連絡方法をとるかを確立させておかなければならない。このシステムが在庫管理型であるか、問合わせ応答型であるか、データ集配的なものかによって連絡体制も変わってくるし、予約システムとか銀行システムのように対象とするものが人間であるか、あるいはプロセスコントロールのように物であるかによっても変わってくる。通常前者の場合、加入交換システムの電話を使用したり、センター側に異常が生じたなら別系統で各端末一斉に異常状態に応じた表示をして端末オペレータに知らせるようにする。

システムに異常が生じた時、場合によっては重大なプロセスの事故や人命にかかわるようなこともあるので、単に連絡の問題だけでなく安全性の保持についてもシステム設計に関係して考えておかなくてはならない。

(3) 二重系の切離し

二重系の並列システムでは1つの入力に対する処理を両系統で同時に行なってお互いの処理結果が一致するかどうかをチェックする。このクロスチェックの情報が一致しなかったり、制限時間が超過しても他方からの応答がなかった時は異常状態となる。クロスチェック不良の時は入力に応じた模擬入力を発生し、両系統でこの模擬入力に対し処理をして再びクロスチェックを行なう。この時結果が一致すれば一時的障害であるし、結果が再び不一致の時は各系統毎に前回の結果と模擬入力の結果を照合して異常状態の方の系統を発見する。また応答の制限時間が超過したものは明らかに異常状態であり、これらの異常系統の切離しを行なう。

(4) 待機予備機への切換え

主系統が異常となった時、待機系へ切換えるにはまず主系統で実行中であったプログラムを待機系へロードし、また処理途中の各種情報、ファイルの状態などを待機系へ引渡して、主系統はシステム停止の状態になる。切換えそのものは異常状態の検知によって自動的に切換えたり、オペレータによるマニュアル操作の方式による。

(5) ファイルの回復

機器の異常やプログラムエラーなどによってファイルの内容が破壊された場合には直ちにファイルの回復を行なう。ファイルの内容がリア

ルタイムでは更新されず読出し専用であれば最新の予備ファイルをコピーするだけで済むが、更新が常時不規則に行なわれるシステムではファイル更新のトランザクション毎に入力メッセージ、あるいはファイルへのアクセス内容を磁気テープに記録しておかなければならない。そしてファイルの破壊時点までこの記録をさかのぼり、ファイルリカバリィのサポートプログラムによって正しい内容にファイルを作り直す必要がある。

(6) 周辺機器の切離しと再構成

機器の切離しは一般にフォールバックと言われている。オンライン・リアルタイムシステムでは計算機本体の二重系と同様に周辺機器の場合にも一部が故障しても全システムを切換えたり、動作停止にならないよう処置を考えておくことが大切である。リアルタイムシステムには多くの異なった形式のフォールバックがあり、システムを構成する多数の機器については、ユニットが故障したりあるいは保守のために切離した場合システム全体がいかなる動作を行なうかを前もって考えておく必要がある。フォールバックの形式としてまずカードリーダ、プリンタ、テープなど入出力機器の障害の時は障害機器を代替機で置きかえる。カードリーダ故障の時はキーボードからマニュアルで入力したり紙テープリーダより入力したりする。プリンタが故障の時は一時的に内容を磁気テープにとっておいて、故障修理後プリントアウトしたり、又は紙テープにダンプしておいてオフラインでプリントアウトする。リアルタイムシステムの場合、ディスク、ドラム等は系全体を二重化するのと同様にファイルも二重化しておいて故障した方を切離して保守する。

(7) チェックポイントとリスタート

チェックポイントは障害によってもう一度最初から作業をやり直す無駄を省くため適当な間隔で主記憶装置の必要なエリアを外部記憶装置にダンプしておく場所である。障害が発生すればその原因を改善してから最も近いチェックポイントの内容を主記憶装置に戻す。これでチェックポイント時点の正常な状態に回復されここから再び処理を開始すればよい。

(8) リラン

周辺機器に異常が生じた時、この障害が一時的障害であるか恒久的障害であるかの区別を行なうため、例えば磁気テープ、ディスクなどであれば同じ入出力命令を再度一定回数繰り返して発行し、それでも正常終了とならなければ恒久的障害と判定しユーザーの異常処理ルーチンへコントロールを渡したり、オペレーターに表示して指示を待つ。何回か繰り返している間に正常に戻ればそのまま続行されるが、この時の記録は磁気テープに保存され、後でのメンテナンスに利用される。

(9) プログラムエラーのリラン

一般にハードウェアの故障の時はリランという対策がとれるが、ソフトウェアのエラーの場合にはプログラム自体の欠陥のため同じエラーを起こす可能性がある。しかし、オンライン・リアルタイム・システムの場合はバッチ処理と異なり必ずしも同じエラーが生じるとは限らない。すなわち、多重化されたプログラムのタスク間のタイミングによってはテスト段階では考えられないエラーを起こすことがあり、このような場合にはプログラムエラーでもリランすれば、タイミングの変化により障害を起こすことなく処理を続けることも可能である。従って、システムが稼動状態に入ってある特別なデータでエラーが起きた場合には、この

データをはずして他のデータについてシステムが正常に戻るような工夫を設計時に考慮しておく必要があろう。

また、オンライン・リアルタイム・システムと言えども、機械的な故障によってはやむ得ず運転を中断することもあり得る。プロセス制御系ではこのような中断が重大な事故を誘発したり、座席予約や銀行システムのような在庫管理型では窓口業務が混乱する。そのためオンライン・リアルタイム・システムでは運転の中断が事故に結びつかないように、また在庫管理型では能率が落ちても人間側でそれをバックアップできるような処置を考えておく必要がある。

(10) 正常状態への回復

正常状態への回復、すなわちフォールバックの発生以前の状態にシステムを戻すには、このフォールバックの期間中に代替機器に書き込まれた出力すべてに対して回復を行なわなければならない。これにはオンラインとオフラインの両方式があり、計算機室のオペレータはどの位の期間システムがフォールバック状態にあったか、あるいは代替機器にどの程度の出力が書き込まれているかを判断してどちらかの方式をとる。すなわち、小量出力の場合にはリアルタイム動作の継続中にオンラインで行なうが、大量出力の場合には一たんシステムをとめてオフラインで行なうことになる。従って、前述のように周辺機器の切離し及び再構成を行なった場合には、これらの状態がオペレータに明らかにわかるような処置をオペレーティングシステムの中にとっておく必要がある。

4.5.3 オンラインプログラムで障害が発生した時の原因の追求と除去

(1) オンラインプログラムの問題点

まずオンラインシステムの問題点としては入出力機器が端末として回線

で結ばれて遠方にあるため、障害時の処置についてバッチシステムに比べると極端に難しくなる。また通常の計算機には使用されない特殊な装置、たとえばCCU (Communication control unit) とか、グラフィック・ディスプレイ、プロセスのためのI／O装置などがあり、これらの障害は単に計算機だけの問題ではなく計算機とその対象とするシステムのつなぎにおいて、または対象とするシステムと計算機との相互関連において問題が生じてくる。

またプログラム上の問題としてはプログラムが多重的に実行されるため各種のタイミング関係が問題となり、原因を追求したり除去したりすることが難しくなってくる。また、オンラインシステムでは業務開始後は発生したエラーを修正するために一度にテストする時間をとることが難しく、何回にもわけて少しずつ行なうためにかなりの時間が必要となることがある。

(2) 追求の手段

これらの誤りの追求の手段は各システム毎に異なっていて一般的な方法はない。しかし1つの手段としては次に述べる種々の診断プログラムを用意してエラー発生時にはこれらを有効に活用する。

(3) 診断プログラムの活用

一般的な診断プログラムとしてはコアダンプやファイルダンプなどがあり、これらについては故障の記録の所で若干述べたが、この他にプログラムの中にあらかじめトレースできる場所を決めておき、エラーが発生した時にはここをトレースする形にオペレーションシステムやアプリケーション・プログラムを組んでおく方式も考えられ、診断のために役立てることができる。このようなトレースを一般にスナップショット・ト

レースと言うが、ただこれを常時働かせるとシステムの効率を落したりトレース記録が膨大になるので、異常が発生した時にだけトレース箇所が生きるような工夫が望ましい。この他正常状態への回復だけでなく、原因を追求するためにもメッセージのロギング、すなわちジャナルを取って解析する診断プログラム、必要に応じてオペレーションシステムにおける割り込みの記録をとる診断プログラムも用意されていることが望ましい。

また、正常時でも計算機を動かしている間にCPUの動作が正しいかどうかチェックするプログラムをオンラインプログラムの一部に組み込んでおき、業務の切れ目には常にそのプログラムにコントロールを移しCPUの正常さをチェックするセルフチェックのプログラムもこの中に含められよう。

対象システムに応じてそのアプリケーション・プログラムなどの動きを考慮し、各システムで独自の診断プログラムを作っておき、機器の正常な動作や効率を防げない範囲でできるだけ多く組み込む必要があるが、これはまたシステムの開発能力との兼ね合いによっても大きく変わってくる。これらの診断プログラムがどのようにうまく作られているかと言うことは故障からの回復時間に大きく影響する。

(4) 機器故障の修理体制

これまではシステム設計の段階で考慮すべき機器故障や回復問題について述べたが、次に運営体制として、ハードウェア的あるいはソフトウェア的に機器に故障が生じた時どのような体制で修理あるいはプログラムの手直しをするかを考えておく必要がある。これによって発生した故障に対して修理時間（MTTR）を縮めることができる。

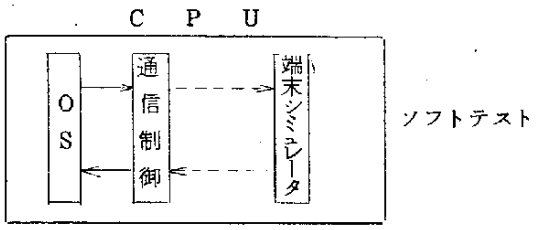
オンラインシステムではまず故障を起こさないようにしておくことが重要だが、一たん故障が発生した時にはそれをいかに早く修理するかと言うことも大切である。従って重要な機器ではハードウェアの修理できる人の配置、端末機器の場合では契約した修理グループの連絡体制、プログラム上のエラーに関してはそれの解析できる人の配置などを運営体制の中に組み込んだ修理に関する体制作りが必要である。

(5) プログラムエラーの修正

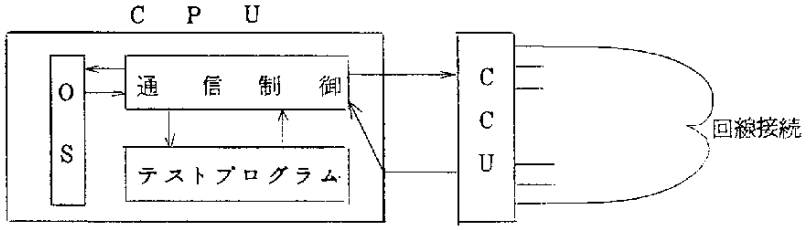
オンライン・リアルタイム・システムでは稼動状態に入るとプログラムエラーは許されないとは言っても、複雑な数十万ステップのプログラムがマルチタスクで動いているため、若干のプログラムエラーが主としてタイミングなどの問題から生じてくるが、この修正は大変難しくかなりの時間を必要とする。これに対しては前述のように誤りをおこしたデータを除去し、他のデータについて処理を続ける。そして誤りを起こしたデータだけを別の時間帯に投入してエラーの発生を調べる。オンライン稼動中このような時間帯をとるのが難しい時はデバッグ用計算機をオンラインシステムの他に設けるとか、同じ構成の計算機システムを借りるような処置をあらかじめとっておく必要がある。

またオンライン・システムではプログラムエラーを修正する時、1ケ所の修正が他の場所へ影響することもあり、エラーを修正した後のテストについては全くシステムの稼動の最初からと同程度のテストが必要であると言うことも十分に認識しておかなければならない。さもなくば現在生じたメッセージに対するエラーを除去しても、他のメッセージとの組合わせに対してエラーが続発すると言うようなことになる。従って、エラーの修正に関しては種々の手続きを決めておかなければシステムに

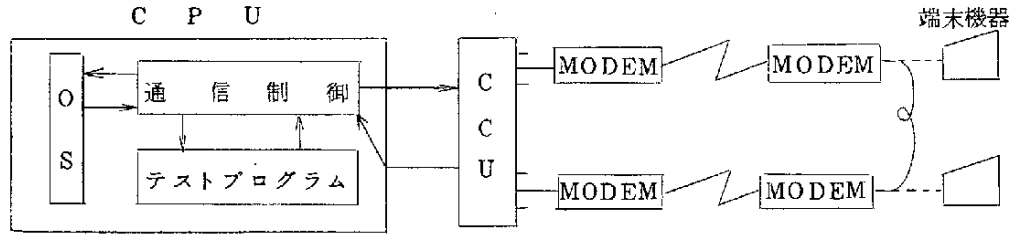
混乱を生ずることもあり得る。



（a） 第1段階



（b） 第2段階



（c） 第3段階

第4－1
図 端末テスト
第4－2

Ⅲ　システムズ・マネジメント

Ⅲ　システムズ・マネジメント

1. プロジェクト・マネジメント

1.1. プロジェクト・マネジメント

1.1.1. ＤＰマネジャーをとりまく問題

この５、６年前までの情報システムの開発は、その規模はさして大きくないものであり、そこに関連する要員の数、ユーザー部門、あるいはシステムの複雑さというものはさしてたいしたことはないというものがほとんどであった。それでこれらの情報システムの開発においては、コントロールシステムの確立などを前提をしないまでも、ＤＰマネジャーの個人的な手腕によって、その場限りの管理を行うだけでも一応の成果をあげる情報システムの開発が可能であったようである。

ところが近年になって様々な環境要因の変化が急速におこり、もはやＤＰマネジャーの直観だけでは情報システムの開発を効果的に行うことが不可能になりつつある。これらの要因のいくつかは次のようなものである。

- 急速な技術進歩
- システム開発要員の不足
- ユーザー部門とのコミュニケーションニードの増加
- プロジェクトスケールの増大
- 固定費の増大
- 新しいアプリケーションへの挑戦

１）急速な技術進歩

コンピューターに関連する技術の進歩にはおどろくべきものがある。その進展の速さは単に技術的な発明がおこなわれていくばかりでなく、それがすぐに応用技術としてコンピューターに採用されていく面でもす

ばらしいものをもっている。

コンピュータはＩＢＭシステム３７０、あるいはＵＮＩＶＡＣ１１１０の出現でもはや3.5世代をむかえている。ここではハードウェア面でみるとＬＳＩがふんだんに使われ、入出力装置の速度は3世代のコンピュータに比べてもはるかにまさっている。またハードウェアの保全性という面をみても、はるかに保全のやりやすい体制がとられている。またソフトウェア面でみると、今までの言語の拡大は当然のことながらテレコミュニケーションの技術、マルチプログラミング、ＴＳＳなどの技術が標準のものとして含まれているオペレーティングシステムの開発など、その技術進歩にはおどろくばかりである。

また最近はとくにミニコンピュータという分野が新しく出現してきており、端末機としてあるいは専用機としてその需要はうなぎのぼりである。このミニコンピュータも技術進歩のおかげで、小さいながらもはや中型機にまけない能力をもったものまで開発されてきている。

このような状況のもとにコンピュータのコストパフォーマンスは飛躍的に増大してきており、その応用分野はますます広がっていくことになり、ＤＰ部門での仕事もそれに応じてますます増加していく傾向にあるといえる。

2） システム開発要員の不足

現在でもプログラマーあるいはシステム・アナリストの不足は深刻であるのに、これからの分野の拡大、技術の進歩につれてこの問題はさらに深刻化する傾向をみせている。

プログラマーあるいはシステム・アナリストの不足は言うまでもないが、システム規模の拡大あるいは技術の進歩に応じて、プロジェクトマ

ネジャーの必要性、あるいは専門的なスキルをもった要員の必要性といったものはこれから急速に大きな問題になってくるものの一つであろう。

それと同時に、限られたマンパワーで増加していく仕事をやりとげていく為には個々の要員のスキルズインベントリーを通しての要員の配置をうまくやっていかないと、プロジェクト間でのアンバランスが生じたり、成功するはずのプロジェクトが失敗したり、プロジェクトのスケジュールが大幅にくるってきたりしてプロジェクトの達成がおぼつかなくなってくる状況をつくりかねない。

この問題を解決する為には、どのような教育、訓練計画をたてたらよいか、あるいは要員の採用計画はどのようにしていくかなど今のうちから様々な考慮をしていく必要があるはずである。

3) ユーザー部門とのコミュニケーションニーズの増大

今までの情報システムの開発はＤＰ部門内でのもの、いいかえるとユーザーがＤＰ部門自身であるというようなものが多かったし、ユーザー部門と関連をもつ情報システムをたとえとりあげたにしてもその割合は小さいものであった。

ところがコンピュータの能力が増大し、アプリケーション分野が広がっていくにつれて、ユーザーの欲している情報システムの開発が夢ではなく現実のものとなってきた。このような情報システムの開発においては、まさにユーザーとのコミュニケーションというものが重要な位置をしめるようになってきている。

ところが、ユーザーがコンピュータの専門家が使うような言葉で彼の要求を表現できるはずはないし、ユーザー自身がコンピュータの機能を理解していることはない。また遂にコンピュータの専門家がユーザーわ

かり易い言葉で説明してくれるとは限らないし、ユーザーの組織あるいは業務内容を明確に理解しているとは限らない。このような相互不理解という状況はユーザー部門のシステム開発をしようとするときにはきわめて大きな問題となってくることは明らかである。

システム開発はあくまでサービス機能であり、会社の実行部門であるユーザーの為の情報システムの開発を行ってこそ、その価値がでてくる。とかくシステム開発が、システム開発の立場に立って行なわれがちである。このようなことがおこらないように、システム開発へのユーザーの参加をもとめ、彼らとの間に有効なコミュニケーションを行うことがこれからのシステム開発部門の大きな課題の一つである。

4) プロジェクトスケールの増大

開発される情報システムの規模が大きくなり、複雑になるにつれて、プロジェクトの規模も年々増してきている。システムの開発にかかる期間が1、2年というものは普通のプロジェクトであると見なされるようになって、3年あるいは5年といったプロジェクトが珍らしいものではなくなってきている。

このようにプロジェクトの規模が増大してくると、それに投入されるリソースもたいへんな量になってくる。そこでこのような大規模なプロジェクトではリソースのムダがないように、あるいは長い開発の間にプロジェクトの目的が不明確にならないように、あるいはまたいつでもプロジェクトが中止できるような体制を作っておくことが必要になってくる。

プロジェクトでの基本的なリスクとしては、

- 開発はユーザーのニーズに一致する形で進められているか。

○　初期にユーザーニーズを満足していても、それが完成したときにはどうか。

○　そのシステムは実働できるのか。

といったものが考えられる。

5）固定費の増大

様々な情報システムが開発され、さらにこれからも新しい情報システムの開発が計画されているような現在、設備であるとか要員などの固定費は極めて大きなものとなりつつある。特にプロジェクトの規模が大きくなってくるにつれて、それが運営されるようになったときには、そこに多額の操業費がかかることは前もってわかっているのであるから、そのコストの妥当性をシステム開発に先だって検討しておくことが必要である。その検討でコスト的にもそのシステムが妥当であるとされて始めてシステム開発への第一歩が踏み出されるべきである。

6）新しいアプリケーションへの挑戦

新しいソフトウェアの開発、企業をとりまく環境の変化、あるいはコンピュータの能力の拡大、端末機の開発などによって、今までより広い範囲でコンピュータを使用することが可能になり、新しいアプリケーションを開発するニードがユーザーから次々と出されるようになってきている。

これらのアプリケーションの開発にあたっては、今までＤＰ部門が持っていたスキルでは足りないものがたくさんある。そこで、これらに必要なスキルを獲得したり、あるいはそこで必要となる新しい設備を獲得したりする為の計画をたてることが必要になってくるわけである。この為には、将来に対するはっきりとした予測を持つことが必要であり、そ

の計画を立案してこそ始めて、ユーザーに対してこれからのＤＰ部門のあり方を提示し、そのニードに答えることが可能になるわけである。

7） まとめ

このように様々な問題にＤＰマネジャーが直面している現在、これらの問題を解決し、情報システムの開発を効果的に行っていく為の何らかの方法論を開発する必要が生じてきている。

1. 1. 2. プロジェクト・マネジメントをとりまく環境と範囲

導入されるプロジェクト・マネジメントのタイプは組織によってまったく異っていることは明らかである。その導入のタイプを決める主要な要因には次のようなものがある。

- 会社の大きさとその内容
- ＤＰ組織の大きさ
- ＤＰ部門の組織形態
- プロジェクトの特性

ある組織に導入されたマネジメントは他の組織ではまったく働かないこともある。だから各々のＤＰ部門では、独立のプロジェクト・マネジメントを確立する必要があるわけである。

1） 会社の大きさとその内容

プロジェクト・マネジメントにもっとも大きな影響を与えるのは、その会社自身のポリシーや環境である。米国の政府関係あるいは海軍関係のＤＰ部門ではすでによく整備された標準をもっていて、それが実際に稼動している。このようなところでの標準は相互的にきびしくて、細部にわたっている。それとは反対に、小さな会社では手順として最小のもので、もっと大まかで自由度のあるシステムを求めているところが多い。

しかし会社の大きさが必ずしも導入されるべきシステムの内容をきめているとは思えない。例えば、その運営部門が分散しているような会社でも、プロジェクト・マネジメントに共通性をもたせなければならないという問題をかかえているところもある。

もう1つは、その会社が行っている商売のタイプというのがプロジェクト・マネジメントに影響を与える。例えば財政部門は、政府や企業の現状にあうような形で作業を進めなければいけないから、すでにある規律というものをもっているから、DP部門がここにサービスをしようとするときには、この規律というものをDP部門の中でうまく処理しなければいけないことになる。

2) DP部門の組織形態

DP部門の大きさと組織が、導入されるプロジェクト・マネジメントに最大の影響を与える。会社のすべてのDP機能が集められているような、小さなDP部門は、より簡単なシステムを望むことになる。というのは、そこはたいした責任をもっていないし、報告書をたくさん出すべき必要もないからチェックポイントもさして多くなくていいのである。DP部門でのコミュニケーションは容易であるから文書もさしていらないということになる。

DP部門を分割しているような大規模な会社、例えば、ORチーム、ソフトウェア開発チーム、アプリケーション開発チーム、運営チームなどに分割している会社ではより複雑なコミュニケーションを行う為に、より多くのチェックポイントが必要であり、それに応じてより多くの文書が必要になってくるわけである。

分散化されたDP部門は特殊な問題をもっている。各部門が行ってい

るプロジェクトのタイプとＤＰ部門のポリシーによって次の３つのアプローチの内の１つがとられる。

a）一カ所でプロジェクト・マネジメントが開発され、それを他のところにも適用していく。

b）各部門が独自のプロジェクト・マネジメントを開発する。ただしこのとき作業が重複することだけは避ける。

c）すべての部門が協力して、一つのプロジェクト・マネジメントのもとで働くようにする。

3）プロジェクトの特性

プロジェクト・マネジメントの範囲や複雑さは開発されるプロジェクトの特性による。もっとも重要な点は、開発に要する時間の長さとプロジェクトチームの大きさである。開発過程での文書は、プロジェクト・コントロールの一つの目的である、プロジェクトの進捗管理のガイドとして重要なものである。だから期間の長いプロジェクトをうまく管理してゆく為には、よりひんぱんに細かいレベルでの文書を作成させることが重要である。同様に、プロジェクトに割り当てられる人間が増すにつれて管理の必要性は増してくる。人間が増してくると、何かうまくないことのおこる機会が増してくるし、人間の間での話のずれが大きくなってくるから、管理の手順をよりきびしく、チェックポイントもきめ細かに設定する必要がある。

外部で開発されたソフトウェア・パッケージに信頼がおかれていて、最小のシステム分析やプログラミングですんでしまう場合とか、単に緊システムの設置だけで、新しい手順の開発はしなくていいような場合などのように、実行されるプロジェクトが何らかの制限を受けているよう

な組織では、必要とされるチェックポイントの数はおのずから少ないものになるであろう。

それゆえにプロジェクト・マネジメントが行なわれる環境が、そこで行なわれる管理手順の性質や範囲に大きな影響を及ぼすことはいなめない。

1.1.3. プロジェクト・マネジメントにおける基本的な原則

プロジェクト・マネジメントを行っていくときの原則としてはいろいろなものが考えられるが、ここでは特にプロジェクトという観点にたっての原則についてふれることにする。

1) 情報システム開発は一つの資本投資である

情報システムもやはり一つの資本投資であり、他のプロジェクトを同じようにそのコストが評価されるべきである。

コンピュータを取得しようとするとき、その理由として、これは技術への投資であるとか、会れは会社にあることが普通であるといわれることが多かった。しかし今日ではほとんどの会社にコンピュータがあり、コンピュータそのものに対する神秘性は徐々に薄らいできているようである。そこで本来のコスト／パフォーマンスという立場からコンピュータ取得についても考慮する可能性がでてきている。

コンピュータ技術が発展してくるにつれて、様々なアプリケーションが利用可能になってきており、DP部門の開発を待っている数が多くなってきている。この開発が遅れている唯一の理由は人的リソースの不足である。このような状況で、会社にもっとも貢献が期待されるアプリケーションの領域へ努力を集中させる為には、そのアプリケーションのコストを評価することが必要になる。このコストを妥当化するという規律

を設定すれば、あるアプリケーションがもっとも妥当であるとされるまでは、他のアプリケーションの調査も平行して行なわれることになるし、このコスト評価の資料をマネジメントに提出して監査をうけることもできる。

このコストでプロジェクトを選択するという標準を設定することによって、DP部門でのプロジェクト選択に妥当性を持たせることができるばかりでなくて、そのコストの資料を受けとったマネジメントは、DP部門以外のプロジェクトのコスト資料と比較してDP部門全体を通してのコストの妥当性を評価することが可能になるわけである。

単にコストで評価するといったが、情報システムの開発コストというものはなかなか定量化しにくいことは確かである。このことについては後に詳しく解説することにする。しかし、情報システムの開発を一つのプロジェクトとしてそのコストを評価することこそ、プロジェクト・マネジメントを確立する第1歩である。

2) システム開発の成功度合を測定する評価基準を作れ

プロジェクトが中止または終了した時点でそのプロジェクトが成功したかどうかを判定する成功尺度を作っておくことは重要である。これは成功した場合にはその成功度合をマネジメントが評価する為に用いるばかりでなく、特にプロジェクトの失敗を早期に発見し、その失敗の原因を調査するときに非常に役立つ。

情報システム開発における成功尺度の一つの特徴は、それが開発されるシステムの中に組み込まれ、その判定がシステムのアウトプットによって自動的に行なわれることが多いという点である。例えば受注処理システムの開発を考えてみると、システムに入力される一枚の送り状を処

理する為の平均コストあるいは年間・顧客当りの平均コストといった数字は、そのシステムの中に組み込まれなければならない統計の一種である。するとこの数字は一種の成功尺度であって、システムがいかに運営されているかあるいはどこを改善したらいいのかといった点を明示してくれる。システムのある部分を改善することによる結果は、自動的にこれらのアウトプットの数字として現われてくる。

3) チェックポイントを設定せよ

開発プロセスを管理する為には、それまでに何が終っていて、いまどこにいて、何がなされているかを知ることは必要である。ほとんどのシステム開発プロジェクトはその内容が相互的に異っているので、適切なチェックポイントを選択をするのはなかなかむずかしい。それらはわかり易いものでなければならないし、仕事の概念づけの助けとなる必要があるし、定義がはっきりしていなければならない。

開発フェーズにそったチェックポイントが設定されると、続いて各フェーズの中でのさらに細かいチェックポイントが設定されることになる。

例えば、プログラミングのフェーズでは個々のプログラムは一つの単位としてコーディングされ、テストされることになる。そこでこれらのプログラム一本一本についてチェックポイントが設定されることになる。

このチェックポイントの設定に関連する事項として平列努力がある。すなわち情報システム開発ではいくつかの仕事が並列して進められることがしばしばある。プログラミングフェーズなどはそのよい例である。そこでこれらの仕事が計画通りに進んでいるか、すなわち遅れている仕事はないか、あるとすればどのように調整したらよいかという問題に有効であるチェックポイントが重要なものとなってくる。これらのチェッ

クポイントをあまり設定しすぎると調整の為に余分な時間を使いすぎることになるし、余りに少なすぎるととりもどせないような遅れを生じてしまう結果になる。

このようにチェックポイントの設定はプロジェクトのかなめともなるものであって、十分に注意を払うべき必要がある。

4) プロジェクトの中止を恐れるな

システム開発プロジェクトをいくつかのフェーズに分割することにはいくつかの利点はあるけれども、その内の一つはそのフェーズの内のどこでもプロジェクトの中止、延期、あるいは続行を考慮できることである。

特にこのプロジェクトの中止についていえることは、プロジェクトの初期においてプロジェクトを中止するとしてもそれはさして痛手とはいえないけれども、プロジェクトがかなり進行してからプロジェクトを中止しようとすると、これはたいへん経済的にも大きな損失になるということである。

さらに初期のシステム研究あるいはシステムの定義というフェーズでは技術的な実行可能性あるいはシステムのリスクといったものが十分に検討されるべきで、これをおこたると重大な失敗の原因となりかねない。すなわち実行可能性といったものはこの初期のフェーズでのみ検討されるべきものであって、いったんこのフェーズでOKのサインがでてしまうと、もはやそのシステム開発は完成に向けて一直線に走り出すことになり、マネジメントは多大な開発コストならびに運営コストにコミットメントを与えたことになるわけである。

まとめてみれば、プロジェクトを中止するのは早ければ早い方がいい

のであって、特にそれが研究のフェーズにあるうちならば、さして気にするほどのことでもないということである。

5) システム開発はくり返しの過程である

システム開発の規模が大きくなり、その複雑さがましてくるにつれて、そのプロセスを理解し、管理していくことがむずかしくなってきているのは事実である。

これに応じる為には、まずプロジェクトの全体を見てその理解に努め、それを具体的に理解できるところまで細かくレベルダウンしていくことになる。すなわち情報システム開発プロジェクトは一種の階層構造をなしており、その階層構造に従ったレベルダウンが必要であるということと、そのレベルダウンは1種のくり返しになっているということである。例えばプログラミングのフェーズを考えてみると、まずプログラムがあり、それを構成するモジュールがあり、それからサブルーチンがあるといった具合である。

このようにプロジェクトを一つの単位としてみるのではなくてそれを何段階かについてレベルダウンしていくことによって具体的なタスクが設定されるべきであり、そのレベルになって始めて詳細な目的を設定し、手順を決め、その目標完了日を決めることができるわけである。またこのレベルになると、その仕事の内容はかなり類似したパターンになってくるので、お互いのメンバー同志で情報を交換したりすることが可能になってくる。

6) 文書を活用する

プロジェクト・マネジメントにおいて、一つの有効な管理の道具として文書がある。システムの規模が拡大するにつれて、そのプロジェクト

の中での情報の流れもそれに応じて増大してくることになる。そのようなときに、情報の流れを管理する為にその形式をきめ、必要となる時点を明確に定義した文書化の手続きをきめることはきわめて重要なことになる。いいかえると、プロジェクトを管理する為の唯一の資料として文書が存在しているのであるから、その形式の決定あるいはサイクルの決定には十分の注意を払うべきである。

文書化の利点をあげてみると、

(a) システム設計のレベルで、その内容を紙の上に整理することによって、その弱点を明らかにし、改善を行うことができる。

(b) アイデアを表現しようとすると、おのずから明確で、かつ、整理された言葉を使うことになり、そのアイデアをうまくまとめることができて、その結果として必然的に良いデザインが生まれる。

(c) 要員の割り当ては流動的であるし、そのプロジェクトが最後まで続けられていくかどうかもわからない。それゆえに、良いアイデアは書きとめておかないと、失なわれてまた作り直さなければならなくなる。

(d) 十分な文書化がチェックポイントであるフェーズがある。例えば設計フェーズで提出されるプログラム仕様書。

(e) 開発プロセスでの文書化は管理資料となる。

(f) プロジェクトのいろいろなフェーズで積みあげられていく文書を整理しておけば、保全の為のマニュアルあるいは資料を作るのが非常にスムーズに行うことができる。

7) まとめ

これらの原則を実際に適用していくことによって前に述べたＤＰマネ

ジャーの問題を解決していくことができるわけである。その為にはこの原則がただの羅列になることを避け、ある開発フェーズの中で順序だった形でとり入れられることが必要である。さらにいえば、そのようにされてこそ始めてこの原則が生きてきて、実際の効果を発揮してくるはずである。

1.2. プロジェクト・コントロール

ここでは、プロジェクト・コントロールを担当するＤＰマネジャーに役立つ様々な手続きや手法について述べる。

1.2.1. アプリケーションの定義とプロジェクト・セレクション

悪いシステム開発の種類には次の２つのタイプがある。

- ○　アプリケーションの定義が誤まっている場合
- ○　システムの定義が誤まっている場合

前者は、ユーザーもしくはＤＰ部門のＲ＆Ｄグループがプロジェクトの目的、範囲、利用可能なリソースなどを誤まって認識することが原因であり、後者はデータベースの仕様や例外処理に対する充分な理解をしていないことが原因となる。

これらの例としては、プロジェクトの初期において、ユーザー部門の特定業務をシステムの中に含ませるべきであるとして、その業務をシステムの中で実現する為に多大な無駄をしてしまったり、ユーザー部門とＤＰ部門の間でのコミュニケーション不足がシステムが運営され始めてからわかったりすることがある。

それゆえにすべての仕事を始めるまえにアプリケーションを適切に定義し、それらの間でのプロジェクト・セレクションを行うことが重要なこと

である。

1） ユーザー要求書

プロジェクト・セレクションにおける最初の文書であり、もっとも基本となるものがこのユーザー要求書である。ただこの呼び名やこれを書く人あるいは内容といったものは千差万別である。

この要求書にもとずいて、システムの範囲あるいはシステムの仕様を検討する為に次のような基本的な質問がなげかけられる。

- 現在のシステムをグレードアップすることですまないか？
- 現在の帳票やアウトプットは続けるのか？
- 手続きに変更は生じるのか？
- マネジメントのやり方や習慣に変化は生じるのか？

この要求書あるいは質問結果にもとずいて、委員会がどのプロジェクトにどれだけの予算を与え、ＤＰ部門の人間とユーザー部門の人間がチームを作って活動するかを選択し、それに権限を与える。

この時点である程度のコストの見積りを行なう必要はあるが、これには本質的な不確実性が含まれていることは忘れてはならないことであり、この見積りはステップが進むにつれて確実にしていくべきである。

プロジェクト・セレクションの概略は図１－１に示されるごとくである。ＥＤＰプロジェクトの新しいアイデアはいろいろな部門から発生する。ユーザー要求書でこのアイデアが文書化されるわけであるが、これは彼の要求しているサービスを普通の言葉で述べたものにすぎない。

このユーザー要求書が、全体的な実行可能性を検討する為にＤＰ部門のＲ＆Ｄ担当マネジャーあるいはデータ処理担当マネジャーのところに送られる。より詳しい情報が必要であれば、特定の質問をそえてそれを

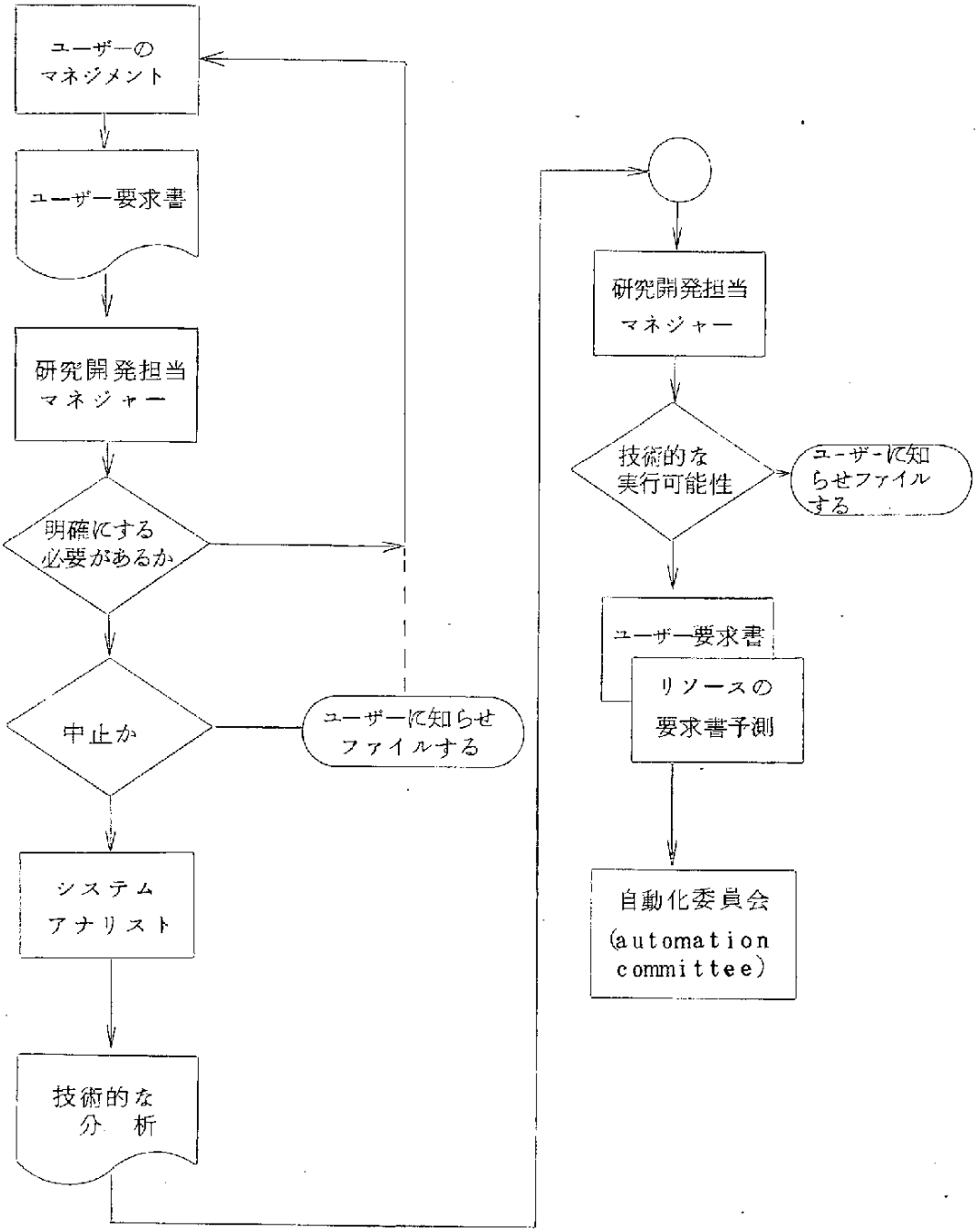


図１－１　アプリケーションの定義と
プロジェクト・セレクション

ーザーに返すことになる。多くの場合、ここで個人的に会うことになる。

マネジャーが何らかの利益を生み出すことができそうだと考えれば、これに彼の意見をそえて、ＤＰ部門のシステム・アナリストに送る。そしてシステム・アナリストがそのアイデアの技術的な実行可能性を検討し、システム、プログラミング、オペレーター、マシン、タイムなどのあらい予測をして、それをマネジャーのところにもどすことによって、さらにそのアイデアが評価されることになる。

2) リソースの要求予測

委員会がプロジェクトを承認したり、スケジュールをたてたりする資料として、データ処理担当マネジャーは、「リソースの要求予測」を作らなければならない。これは現在実行中のプロジェクトあるいはこれから始まるプロジェクトに必要になるマンパワーあるいは設備の必要量を推定したものである。これは次の３つのレポートを基礎にして算定される。

- 保留中のプロジェクト・ファイル

  すでに承認され、マンパワーと設備の推定必要量が付け加えられているユーザー要求書のコピー。

- 進捗報告書

  現行進行中のプロジェクトを終らせるまでに必要なマンパワーと設備の詳しい推定が書かれている報告書。

- オペレーターと設備の活動月報

  １カ月間でのオペレーターと設備の使用時間の報告書。

これらすべての要求量がこの６カ月間、次の６カ月間、次の１２カ月

間の3つの期間に分割されて集計される。そしてこの予測は普通月に1回更新されるが、リソースに特別の変更があった場合には、その時にも更新されることになる。

この予測にもとずいて現在のプロジェクトを完成させる能力あるいは新しいプロジェクトを開始する能力が評価される。1カ年の予測をもとに、現在のマンパワーあるいは設備で充分なのか、あるいはさらに追加する必要があるかが検討される。これにより、新しいプロジェクトの開始日あるいは終了日が決められ、保留中のプロジェクトの優先順位がつけかえられ、そして必要となるマンパワーと設備の獲得が行なわれる。

この予測の期間に近年とみに長くなってきている。というのはDP部門の規模が拡大し、保留中のプロジェクトが増してきているからであり、2カ年、3カ年あるいは5カ年計画が必要となってきている。

いずれにしても、将来どのようなリソースが必要となるかを文書化し、正式な手続きを踏んで開発の要求を処理することが重要なことである。

1.2.2. プライオリティ・システム

システムの導入が順序だって進んでくると、個々のプロジェクトの会社全体の業務に対するあるいはリソースを競合する他のすべてのDP部門のプロジェクトに対する相対的な重要性を決定する何らかの方法が必要になってくる。さらには、開発プロセスを通してそのプロジェクトされるリソースの量で個々のプロジェクトを客観的にリスティングすることにとって、プロジェクト相互間での関係を明確にすることも必要となってくることになり、そこで優先順位分類のシステムが役立つことになる。

委員会がユーザーの要求書あるいはリソースの必要予測に基ずいてプロジェクトを承認するときにはどの仕事よりも優先順位が先につけられるべ

きである。この仕事はなるべくユーザー部門あるいはＤＰ部門にまかせられるようにされるが、最終的には委員会が決定する。

ここで提案されている方式は、少々の変更を加える必要はあるけれども、どこにでも利用可能で有効な方式である。

個々のプロジェクトに対する優先順位は次のようにつけられる。まず、これらのうちの１つは、他のすべてのアプリケーション分野との比較をした上での特定のアプリケーション分野に対する相対的な重みを決める。次はそのほかのプロジェクトが特定のアプリケーション・プロジェクトに対する時間の必要量で相対ずけられる。そして委員会でこの２つの相対的優先順位のリストを個々のプロジェクトに割りあてて、この結果としてこの２つの組合せでプロジェクトの順位が決まることになる。

1） 会社中での給料、受け取り勘定等の機能に対して優先順位を表わす文字（A、B、CあるいはD）をつける。すべてのプロジェクトはこれらのいずれかに入ることになる。

A－最優先、必要となれば利用可能なリソースをすべて投入する。

B－重要、必要となれば現在進行中のプロジェクトのスケジュールをややりなおすこともある。

C－普通

D－非重要、その導入が遅れても、会社全体の業務には差しつかえないもの。

2） 優先順位を表わす数字（イ、ロ、ハあるいはニ）が個々のプロジェクトに割りあてられる。この数字の優先順位は次の点が基本となっている。

○　プロジェクトの緊急性

○　完成までの期待総時間

○　プロジェクト開発コスト

○　期待節減額

○　期待増加利益額

イ－最緊急、完成期待日が希望される完了日を越えているもの。

ロ－できるだけ早く必要、この優先度は期待開発コスト、期待節減額ならびに増加利益額による。これらのプロジェクトに導入されしだい利益性を増大するものと期待される。

ハ－普通

ニ－やればよい、しかしやったからといって開発コストに比例した節減額あるいは増加利益額があるわけではない。

この２つの優先順位が個々のプロジェクトに対してつけられると、他のすべてのプロジェクトに比べた全体的な優先順位をつけることができる。全体的な優先順位を高いものから低いものの順に並べるとすれば、$A_{イ}$、$B_{イ}$、$C_{イ}$、$D_{イ}$、$A_{ロ}$、……、$C_{ニ}$、$D_{ニ}$ということになる。

この優先順位システムあるいは優先順位の分類は時に応じて変わってくる。優先順位のつけ方やあるいはその分類のやり方が変わった場合には、それが混乱をきたす場合と容易に受け入れられる場合とがある。前者の場合には最初の順位がつけられた時の状況がたとえ委員会に理解されていようとも、優先順位に大きな違いを生じてくることもありうるわけである。保留中のシステムや開発中のシステムがすべて同一の緊急性をもっているならば、ＤＰ部門の長期計画においてさらに多くの能力を持ちうるように再検討されることになる。

システムを成功させる最も重要なポイントはユーザーの参加である。

スタッフは優先順位によって分類するシステムやその中での優先順位のつけ方にはなれ親しんでいるし、ユーザーのマネジメントには保留中あるいは現在進行中のシステム開発プロジェクトの優先順位を知らされている。そこでもしユーザーが自分のシステムの優先順位をあげようとすれば、委員会の審査にたえうるようなデータをそえて委員会に直接再検討を要求することができるようにすべきである。さらにはＤＰマネジャーがすべての責任を１人で背負っているわけではないということ、すなわち優先順位を最終的につけるのは委員会であるということが重要な点である。

優先順位をつけるこの方法は作りやすいしまた理解もしやすい。これはＤＰマネジャーが自分の限られたリソースをもっとも最高の見返りが期待できるようなカットポイントをプロジェクト・コントロールの中で実現するときのよい助けとなることはまちがいない。

1. 2. 3.　時間の推定とプロジェクト・プランニング

いったんプロジェクトが承認され、その優先順位がつけられると、その時間を推定し、プロジェクト進行の計画をたてるのはＤＰマネジャーの仕事である。すべての計画は各々のフェーズのチェックポイントにおいて見返えされるものであるということは最初から理解されるべきである。

プロジェクト・プランニングの内容は以下のようになるであろう。

- なされる仕事の詳細な説明。
- 個々のプロジェクトでのチェックポイントを明確にする。
- 個々の仕事を行うのに必要なスキルの割り当て。
- 個々の仕事の完成までの時間の推定。
- 個々の仕事に適切な人間の配置をすること。

○　仕事のスケジュールをたてる。

最後の２つは複雑であるので1. 2. 4.と1. 2. 5.で詳しく述べることにする。

1）詳細な仕事の概要

プロジェクトの初期においては、特定のプロジェクトの為のタスク・リストは、表１－２に示されている標準タスク・リストをガイドとしながら作り出される。タスクの修正やより細かなレベルへの分割は、特定のタスクに見合うように行なわれるべきである。例えば"インタビューの指揮"といったタスクはインタビュー項目の作成、インタビューの実施、インタビューをまとめサマリーを作るといったタスクに分割される。

表１－２　　標準タスク・リスト

一　　般

現在の問題の明確化
新システムの目的と期待利益
プロジェクトの範囲の定義
ユーザー作業の概略調査
準備的な概略設計
準備的なコスト推定
プロジェクトへのリソースの分配
プロジェクトスケジュールの開発
プロジェクト予算の開発

研究フェーズ

研究計画の開発
ユーザー・マネジメントのインタビュー
現在の文書の収集

運営統計とコストデータの収集
文書例の収集
現在のプロセスの観察
現在のシステムのまとめ
研究のマネジメントへの報告

分析フェーズ

帳票の分析
レポートの分析
ファイルの分析
必要とされる情報の分析
手続きの分析
代替案の開発
コストの分析
マネジメントへの報告

設計フェーズ

レポートの設計
ファイルの設計
インプット帳票の設計
緊急時手続きの設計
プログラム仕様書の作成
テストデータの収集
マネジメントへの報告

プログラミング・フェーズ

仕様の監査
ロジックの設計とブロック図の作成
デスクチェック
コーディング

テストデータの作成とテスト
プログラム文書化

導　　入
システム・テストの計画
システム・テストの指揮
ユーザーによるシステム・テストの監査
システムと運営の文書化
監視システムの稼動

このタスク・リストを作ることによって、時間の推定を行い、どのようなスキルがいるかを明らかにすることができる。また詳細なタスク・リストを作ることによってそれぞれのタスクの責任の所在を明確にすることもできる。図1－3にこれに用いるチャートをのせておくことにする。

2）プロジェクトのチェックポイント

プロジェクトで仕事が渡されるところには必ずチェックポイントを設ける。例えば、プログラミング・グループにシステム設計仕様が渡されたり、運営部門へ完成したプログラムが渡されたりする。文書類、フローチャート、ブロック・ダイアグラムなどの形のあるアウトプットがでるようなタスクの終了時点は他のチェックポイントである。これらのチェックポイントの目的はなされた仕事を導入の品質標準と比較したときに十分満足すべき結果であるかどうかを検査することである。

チェックポイントのもう1つのタイプは連続的なチェックポイントである。他のプロジェクトと比べて緊急であったり、かなりむずかしいプ

日付：

プロジェクトNo.　　プロジェクト名：

プロジェクト・マネジメント・レスポンシビリティ・チャート

タスク	実行			監査		
	中心作業者	参加者	コンサルタント	実働部門	マネジメント	重役

図1－3　プロジェクト・マネジメント責任図表

ロジェクトであることが解っているプロジェクトがある。これらのプロジェクトでは初期の段階でプロジェクトの一定期間ごとに見直しをすることが必要であり、おこりそうな問題を予測して計画の中に入れておくべきである。

3) 必要となるスキルの割り当て

個々のタスクの遂行に必要となるスキルとか経験は、要員配置のガイドとする為にも明確に定義されるべきである。個々のタスクで必要となるスキルは要員配置のところで述べることにする。

4) 終了時点の推定

終了時点を予測するということはあるタスクに必要となる総時間を予測するということである。その為にはそのタスクを構成している活動を充分に理解することが必要である。それゆえにいろいろの推定レベルが考えられる。準備的な推定はシステム研究フェーズが終ったあとでないと確実な推定にすることはできないし、プログラミングの確実な推定はプログラム仕様書ができあがるまではほとんどできないが、プログラミング作業の推定はシステム設計が終った段階で行うこともできる。推定手続きを手順が全体で同じように行なわれるように維持するのはプロジェクト・リーダーの責任であることが多い。推定の基本ルールには次のようなものがある。

- 推定を行う前にそのタスクについてできるだけ詳しく知ること。
- 前の作業が終るまで最終的な推定を行うのを遅らせること。
- 客観的に行うこと。

5) 推定へのアプローチ

推定には直観的推定、歴史的推定そして標準化された推定の3つがあ

ることは注意を払うべきである。

6） 直観的推定

この方法ではいままでの歴史、標準、そして彼の個人的な経験が組み合わせられて、非公式な形で推定が出される。この方法で推定をうまく行う人もいる。彼らの推定はかなり正確なものであるから、もしそのような人がいる場合にはこの方法によるとよい。

7） 歴史的推定

しかしながら、これらの人をスタッフに見いだすことはむずかしい。ところがうまいことに歴史的推定というのは直観的推定と同じ位の正確さを持っている。これは現在のプロジェクトと過去の類似のプロジェクトを比較するというものである。プロジェクトの違いが書きとめられ、前回のプロジェクトでの時間の延長部分が調整されて、今回のプロジェクトの推定値として採用される。しかし、この方法で行う為には比較すべきプロジェクトがあることと時間の正確な記録の２つが必要である。

8） 標準化された推定

この方法は定義が明確でしばしば行なわれるタイプのプロジェクトに適している。２つの標準が要求される。

- ○ タスクを定義し、それに必要な作業を定義する方法標準
- ○ 個々の作業の終了までの期待時間を決める実行標準

システム分析タスクは、プロジェクトが変わってもプログラミング・タスクほどにはその内容が変わることは少ない。また、プログラミング・タスクは容易に部分に分けて定義することができるということから、システム分析には歴史的推定が、またプログラミング・タスクには標準化された推定が採用されることが多い。

この標準的な推定方法が十分に理解されているとはいいがたいので、残りの部分でシステム分析、プログラミング、機械要件の時間推定をする標準的な方法にふれることにする。

(a) 推定標準

プログラミングといった機能に対する推定標準を設定するステップは次のようになる。

i) 使われる方法やタスクを設定あるいは定義する。

ii) タスクの実行に関する方法論を標準化する。

iii) タスクの実行に関連する変数を決定する。

iv) 様々なパラメータやタスクの間での時間の関係を設定する。

v) タスクそれぞれから標準を引き出す。

vi) これらの標準をスケジュールを作成するのに使う。

vii) 実際の実行データを評価する。

viii) 必要とあれば標準を修正する。

推定を行う為にプログラミング作業の基本的なタスクは次のように定義できる。

i) 方向づけと前準備としてのロジック設計

ii) 詳細なロジック設計

iii) コーディング

iv) テスト計画とデータの準備

v) コンパイルとテスト

vi) 文書化と設置の手助け

(b) プログラム作業の要因

それぞれのタスクによって異なってくるプログムミング時間を決める要因

を決定するのはなかなかむずかしい仕事である。プログラミングの時間とコストに影響をおよぼす要因の範囲とタイプは次の3つに分類することができる。

- 環境要因
- プログラムに直接関連する要因
- ロス要因

イ．環境要因

機能の実行に影響するこの環境要因に関する部分は、たとえプロジェクトの内容で異なってもそれが実行される環境が同じであれば

であるとみなすことができる。環境要因として明らかに含まれるものは、プログラムを開発する機械のタイプ、プログラムをそれに従って書く言語、プログラマーの働く物理的環境、プログラマーによって前もって決められた方法論などがある。これらはすべてプログラミングのコストに重大な影響を与えることは確かであるが、1つのプログラムと他のものとで、変わるものではない。それゆえに文書化の必要とされる範囲は主として文書化の時間に大きな影響を与えるわけであるが、個々のプログラムと同じようにしてすべてのプログラムというものも影響を与えることになる。

ロ　プログラム要因

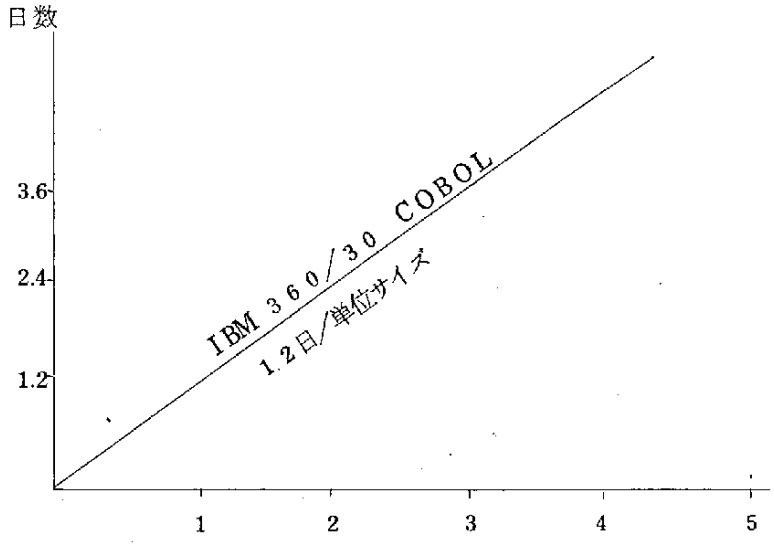
2つの要因は異なったプログラムの特定の要素に関連しているものである。一般的に言えば、これらはプログラム・サイズ、プログラムの複雑性とプログラムのデータベースという3つの特定の分類にまとめることができる。

ハ．プログラム・サイズ

プログラム・サイズがプログラミング時間に与える影響は基本的には線型である。これは命令の数、コーディングのライン数あるいはさらに簡単にはコーディングのページ数で定義することができる。確かに、プログラミングの時間とコストが推定できるような範囲であれば、プログラム・サイズを１０ページ１単位として推定することはさしてむずかしいものではない。それゆえに、コーディングのページ数に関連して開発された方法標準での基本サイズとして、コーディングシートで１０枚あるいは１８０ラインをとるのは普通である。

ニ．プログラムの複雑さ

これは定義するのが少々むずかしいし、より大きな影響力をもっている。このプログラムの複雑さを推定するもっとも単純な方法は“ＳＡＤ”リスティングとして知られている手法である。“ＳＡＤ”はSimple, Average, Difficult の頭文字をとってつけられた名前であって、すべてのタスクの内の３分の１が３つのカテゴリーの中の１つに入るようになっているものである。しかしながらプログラムの複雑さを検討するのには単純すぎる。というのは単純なプログラムと平均的なプログラムの間の差は、われわれが必要としている半日の精度以上になってしまうからである。それゆえに、このカテゴリーの数を拡張するという方法で複雑さを定義するという目的にかなったものを作り出すことにする。６つのカテゴリーを作ることによって妥当性のある推定と精度の両者を満足させることができる。



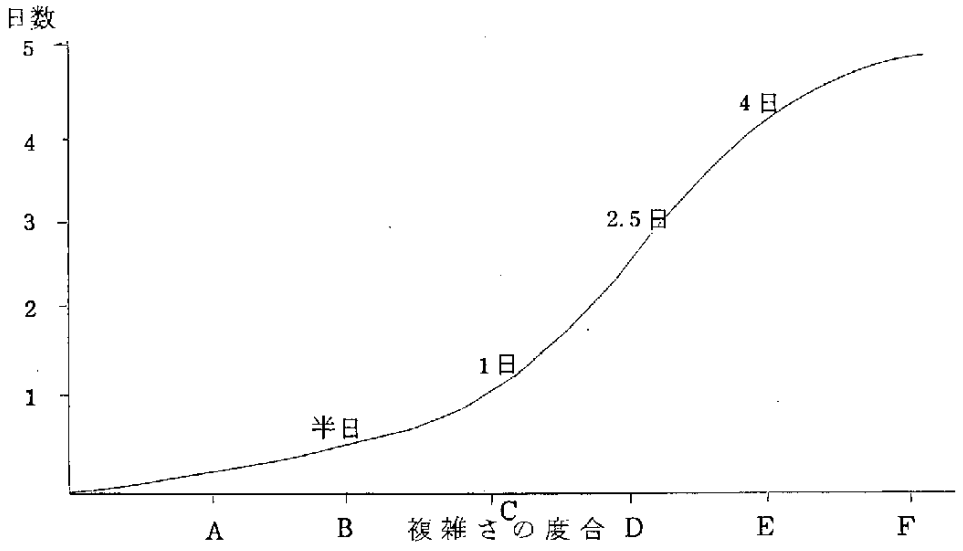


図1－4　コーディングに関する時間見積り

A　単　純

B　やや単純

C　普　通

D　複　雑

E　特に複雑

F　不可能

最後のカテゴリーFはプログラムを書くことが不可能という意味ではなくて、時間を線型に推定することが不可能という意味である。レベルFのプログラムはレベルEのプログラムに比べて極めてむずかしいものであるから、半日の精度で時間を推定することはできない。図－1－4に時間の関係と"不可能"なプログラムがいかにむずかしいかが示されている。

ホ．プログラムのデータベース

この3番目の要因はそのプログラムで取り扱われるファイルもしくはデータベースの数に関連する。これはデータベースでのフォーマットの数もしくはもっと簡単にシステムが操作するハードウェアのファイルもしくは要素の数で数えることもできる。つけ加えるとすれば、ハードウェアの要素ごとに重みをつけること位である。例えばディスクには3、テープには2、プリンターとカードリーダーには1といった具合である。これらの得点の合計がデータベース操作の複雑さを表わす1つのメジャーとなる。大規模でデータベースを指向しているシステムでは、データベースの形やフォーマットの数でさらに修飾することが可能である。

このようにして、プログラムの要素や要因が決まると図1－4に

示されているような時間の関係を表わすことが可能になってくるわけである。

ヘ．ロス要因

要因の3番目の分類はロス要因で、初期の推定の段階では考えられないような何らかの外部要因の変化によってもたらされるかもしれない推定値と実際値の差を考慮するものである。この外部要因には不適切なシステム設計　プログラミング中のユーザーによる変更、適切な時期にテスト時間がとれないことなどが考えられる。これらの要因は予測をよい精度に保つためにできるだけ分離しておく必要がある。もし特定のロスがよく起きるようであれば、その原因を調べて除くように努力する必要もある。

ト．時間の関係を設定する

いったん要因が完全に定義されると、個々のタスクの個々のパラメーターに対して時間の関係を設定する必要が生じる。明らかにこの時間の関係が業界全体で標準化されていることはない。というのは、設置状況ごとにまた機械ごとに環境要因が様々だからである。これらの要因についてに前述した通りであるが、もっとも大きな違いといえば、設置される機械ごとに必然的にプログラミングの方法論が異なることであろう。その結果として、機械を設置するレベルにおいてのみ、方法論を適切に定義し、特定の時間関係を開発することになる。

しかしながら、これらの時間関係が定義できるようなところまでの何らかのガイドラインを設定することは可能である。例えば、第1ステップではタスクを定義されたものとしてとらえ、次にそれを

動作時間研究で使われる手法を効果的にシミュレートするような細かな要素に分割する。このようにすれば、方向ずけの最初のロジック設計というタスクは次のようなサブステップにブレークダウンすることができる。

- ○ 仕様書を読む。
- ○ ファイルのレイアウトを評価する。
- ○ プログラムを主となるロジック要素に分割する。
- ○ 個々の要素についての全体的なブロック図を作成する。

いったんタスクが定義されれば、そこに含まれる要因がコストに影響を与えるようなサブタスクを定義することができる。このようにすれば、ファイル・レイアウトの評価は、プログラムのデータベースとして定義された変数の直接的な関数となる。

同様にして、プログラムを分割することによってプログラムの複雑さとシステム仕様の複雑さが直接的に関連ずけられることになり、プログラムの複雑さとして述べた要因と直接関連することになる。

この時点で、個々のサブタスクを実行するのに必要な時間の範囲を推定することが可能になる。例えば、仕様書を読むといっても、最も簡単なものであれば半日ですむけれども、レベルEのプログラム仕様書を読むのにはやはり4日間位かかることになろう。このようにすれば、タスクを実行するのに必要となる時間を示しているようなテーブルあるいは簡単なリストを作ることができる。例えば、個々の複雑さのレベルに対して半日かかり、ファイル1本につき+1日づつの日数が加えられるといった具合である。同じように、プログラミング作業の個々のタスクやサブタスクについての比較をした

タ　　ス　　ク	大きさ	複雑さ	データベース	範　囲　の　例
タスク1－マイクロ・ロジック				
仕様書を読む		∨		½時間～30時間
ファイルレイアウトの見直し			∨	½時間／ファイル
プログラムに分割する		∨		1時間～20時間
ブロック・ダイアグラム		∨		½時間～10時間
アナリストによる見直し		∨		1時間～8時間
タスク2－マイクロ・ロジック				
個々の要素に分割する	∨	∨		½時間～4時間／要素
ダイアグラム－レベル2	∨	∨		1時間～8時間／要素
見直し		∨		½時間～8時間
タスク3－コーディング				
コード化	∨	∨	∨	8時間／150ライン
見直し	∨			½時間～2時間／150ライン
リンケージ部を加える		∨		½時間～8時間

このようにしてタスク1をまとめると、

A － ½日
B － 1½日
C － 4日
D － 6½日
E － 8日

} ½日／ファイルを加える

図1－5　時間関係の構成

時間関係も開発されるべきである。その例が図１－５にある。

(c) システム分析

同じような手法をシステム分析プロセスにも適用することはできるけれども、この分野で推定の精度はなかなかあげにくいものである。さらにはプログラミング・プロセスでは、半日単位で時間の推定を行なえるような時間関係を示すものまで作ることはできたのに、このシステム分析フェーズではそれほど容易には作れない。そこでこのシステム分析のようなプロセスでの推定の単位というのは２日、３日あるいは５日といったものになってしまう。

またシステム分析に関連する要因も比較できる。機械とその構成要素、システム分析の方法論に関する標準、物理的な環境といった環境要因はプログラミング・プロセスとはほとんど同じである。しかしながら、システム設計を記述するような要因については大きな差異がみられる。それらは、

- システムの複雑さ
- そのシステムでのプログラム本数
- ファイルの数
- 検査される文書の数
- インタビューされる機能の数
- そのシステムによって影響をうける組織要素の数

システム分析の複雑さが必ずしもプログラミング作業の複雑さに関連することはない点に注意すべきである。例えばマルチ・プログラミングモードで何本かのプログラムが同時に走る場合を考えてみると、組み合わされたプログラムが効率的に稼動するように個々のプログラ

ムの特性を決めるのはシステムアナリストの仕事であって、それがプログラマーに渡されたときには、すでにプログラムの組合せとかプログラムの特性とかいったものは考えないでもいいようにすべきである。

イ．システムの複雑さを推定する

ＤＰシステムの運営を特徴づけるようなある標準的なシステム機能には次のようなものがある。

- データ収集
- データ編集
- エラー収集
- データに順序をつける
- 照　合
- 計　算
- データ伝送
- ファイルアクセス
- ファイルの更新
- 問合せ応答
- データ集計
- レポートの準備
- プリントアウト
- レポートの配布

これらの機能の個々について、その範囲を推定することによって、新しいシステム全体の複雑さの概要を知ることができる。

(D)　システム分析の複雑さを推定する

システムアナリストによって行なわれる普通の作業は以下のよう

に分類される。

1. 研究フェーズ

   a 運営統計と文書を集める。

   b インタビューを指揮する。

   c プロセスを観察する。

   d ファイルとレポートを収集する。

   e 現システムを詳細に書き出す。

2. 分析フェーズ

   a 個々の文書やファイルを分析する。

   b 情報の有効活用を計画する。

   c 機能面での要求を記述する。

   d システムの代替案を開発する。

3. 設計フェーズ

   a 新しいファイルを設計する。

   b 新しいアウトプットを設計する。

   c 新しいインプットを設計する。

   d 新しい手続きを設計する。

   e 新しいプログラム仕様書を設計する。

システム分析の複雑さというものは最初のシステムの記述によるところが大きい。というのは、もし現在のシステムで５０カ所からデータを集めているとすれば、その５０カ所でインタビューをやらざるを得ないことになるからである。

しかし、システム開発プロセスが進むにつれて、初期の予測をくつがえすような新しいシステム特性が現われてくることは充分ある

から、その為のコントロールポイントを明らかにしておくことが必要になってくる。

9） まとめ

今までの内容をまとめてみると、ＤＰマネジャーはプロジェクトの最初で、詳細なタスクの概要をまとめ、チェックポイントを設定し、推定値を作り出すことが必要である。そして個々のチェックポイントにたどりつくごとにプロジェクトの見直しを行って次のタスクの推定値をより正確なものに書きかえていく。そして、システム設計の終了時点で、プログラミング時間の推定を、それに関係する要因を再び評価することによって、調整することになる。

10） その他の要因

最終的な推定値を出す前に考慮しなければならない要因があと２つある。

- ○ システムアナリストあるいはプログラマーの経験
- ○ スラックタイム

個人の経験レベルによって推定値を調整する方法にはいろいろある。もし推定を行う時点で参加する要員がわかっている場合には、推定値をそれらの方法のいずれかを使って調整すればよい。またもし要員が未定である場合には、平均的な要員を想定していることを明記しておき、ある程度のバラツキを考慮に入れることにする。このことに関しては要員の配置のところで詳しくふれることにする。

過去のデータをもとにして推定値を出す場合には、すでにそこにある程度のスラックタイムが入っていることになる。確かにシステム開発プロセスの中では予期しえないことが起ってくるから、このスラックタイ

ムが入っていることはさして気にとめることはないし、推定が少なめになるよりは多めになる方がより良い結果を生むことが多い。

プログラミング時間の推定を進めると同時に、どの位のコンピュータ時間が必要であるかという推定をする必要がある。この時間の推定にあたってはコンパイルあるいはアセンブリに必要な時間とプログラム・テストに必要な時間に分けて考えるべきである。

プログラムがその大きさと複雑さによってリストされれば、そのコンパイルにかかる期待時間を設定することはさしてむずかしくない。また必要となるコンパイルの数もやはりプログラムの複雑さと大きさから直接導き出すことができる。そこである言語、機械あるいはオペレーティング・システムの組合せが導入されたところで、このコンパイル時間とテスト時間を算定するための標準を設定すべきである。この導入が行なわれる前であっても、経験を積んだ人間であれば時間を見積ることはできるし、メーカーの本を参考にすれば見積りは可能である。

何回テストを行なうかも、またプログラムの複雑さ、大きさあるいはそのプログラムで取り扱うファイルの数から算定することができる。それゆえにこのテスト時間算定の為の標準も作り出されるべきである。個々のテストの時間は、普通設置する場合のデータを分析することによって決められることが多い。

これら2つの時間をたすことによってコンピュータ時間の推定値とするわけである。

いかによい推定がなされたとはいえ、開発プロセスを進めていく上では、ロスがあったり、期待しなかったような開発がなされてしまうことがある。これは推定をやるのも開発を進めるのも“人間”だからであっ

て、ここに各々のタスクの終了時点でチェックポイントを設定する価値がでてくるわけである。

1.2.4. 要員の配置

タスクを設定し、チェックポイントを設定し、時間の推定を行なった後で、ＤＰマネジャーが行なわなければならないのがプロジェクトチームへの要員の割り当てである。この仕事はプロジェクトの個々のフェーズで必要とされるスキルを決定することから始まることになる。

システム分析に必要となるスキルは次のようなものである。

- 調査をするスキル－データを集めたり、システムのテストをしたりするのに必要となる。
- ものを書くスキル－進捗報告書を書いたり、文書を準備したり、システムの概念をユーザーに宣伝したりするのに必要となる。
- 分析をするスキル－システム要件をまとめるのに必要となる。
- 創造するスキル－詳細なシステム設計に必要となる。

プログラミングに必要となるスキルは次のようなものである。

- 分析をするスキル－システム仕様を理解するのに必要となる。
- 創造するスキル－ロジックを作り出すのに必要となる。
- 調査をする能力－デバックを行なうのに必要となる。

これ以外に特定のプロジェクトにおいてはプログラマーにはある経験が要求されることがある。それはあるハードウェア、言語、オペレーティングシステム、ソフトウェアパッケージあるいはそれらのアプリケーションなどについての経験である。

個々のタスクが特定の要員を必要とするというよりもある特定のスキルの組合わされたグループを必要としているという形で要員を活用するとい

スキルマトリックス：プログラマー	機械と言語					アプリケーションの経験			特性			
名前	IBM1400	CDC7600	COBOL	AUTOCODER	BAL	給与	受取り勘定	在庫管理	コーディングが早い	デバックが早い	文書化がうまい	

図1－6　スキルマトリックスの例

う自由度をもたせることができる。例えばユーザー部門の要員はデータを集めたり、システムテストの結果を見たりすることができるし、あるプログラマーは特定のプログラムを組むよりもデバックをする方が適しているといったことがある。

これらの調整をする為には、R＆D担当のマネジメントにユーザー部門とDP部門の要員のスキルのサマリーレポートを利用可能にしてくれることを要求することになる。大規模なシステム開発の場合には、スキルズインベントリーが正式に設定されるが、普通は要員の見直しを行なうプロセスでその内容を更新するということによって非公式にスキルズインベントリーが行なわれることが多い。

スキルマトリックスの例が図1－6に示されている。これは例として示しただけであって、マトリックスの内容は各社の機械、ソフトウェア、アプリケーションあるいはスタッフの特性に応じて書きかえることが必要である。これらのマトリックスはDP部門の個々の仕事の機能ごとに開発されるべきである。

このマトリックスの中に個々のメンバーのスキルをリストする場合には単に“ある”、“なし”という形で書きこむことが望ましい。さらに簡単にして、対応するスキルをもっているかあるいは能力をもっている場合にだけ“ある”を入れて、その他のところはブランクにしておくこともできる。このようにすれば、特定のプロジェクトに対するスタッフを選ぶのが非常に簡単になるからである。

図1－7に示してある書式はプロジェクトチームのメンバーを選び出すときに使うものである。1行目に必要とされるスキルや特性が書き込まれる。ここに書き込まれる項目は仕事の種類が異なってくれば、その内容は

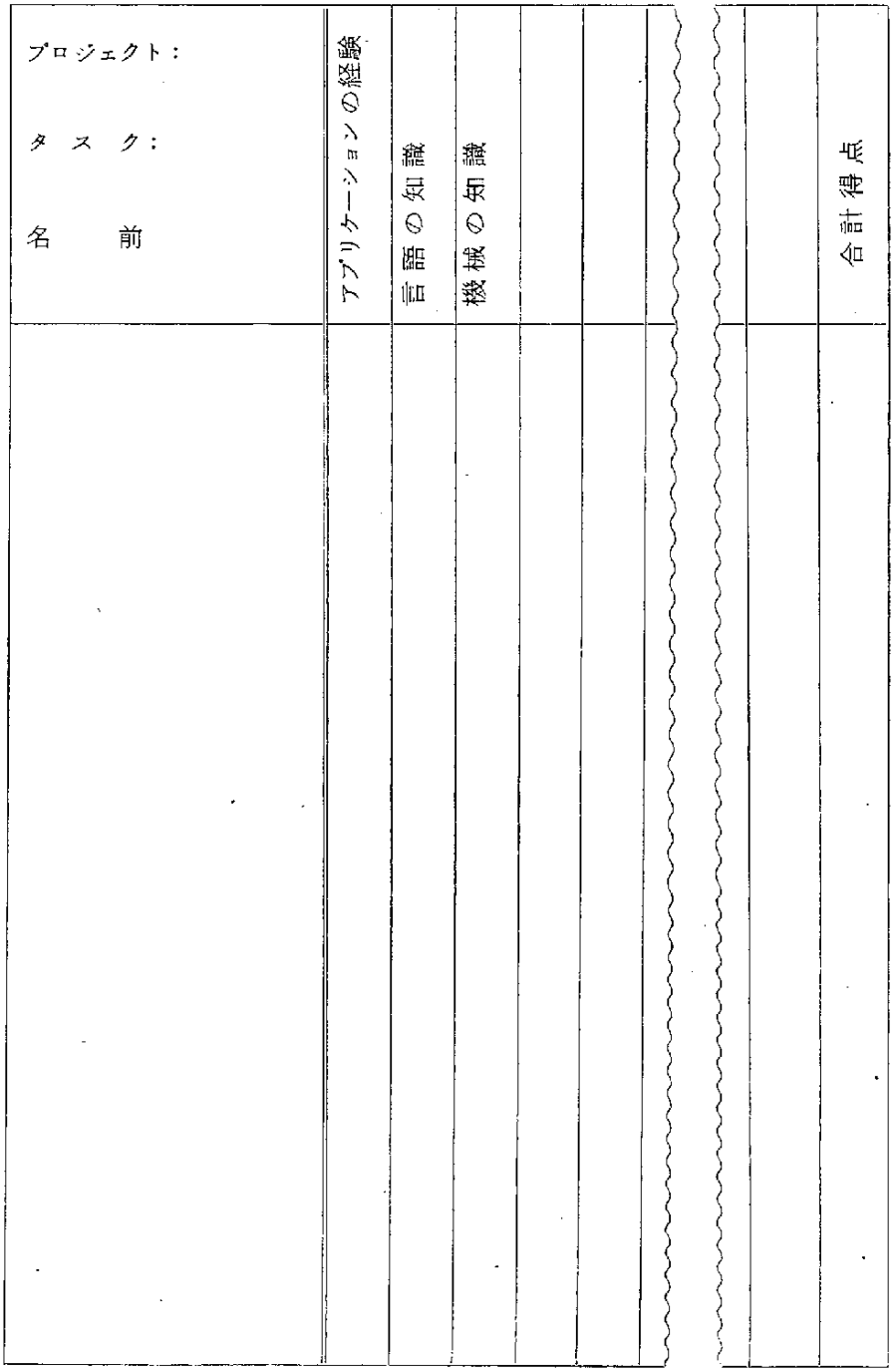


図1－7　スタッフ選択マトリックスの例

変わってくる。例えば、システムアナリストにとっては、同じようなアプリケーションを前に経験していることが重要である、しかしプログラミング言語についての知識はプログラマーに必要であるけれども、システムアナリストには必要でない。

名前の欄には現在フリーであって選択することのできる要員を並べるわけであるけれども、もしそのプロジェクトの優先順位が高ければ、その他のプロジェクトで働いている要員も並べることになる。並んだ要因について、スキルマトリックスの“ある”、“ない”を得点の1、0に変換して選択マトリックスの対応する列に入れる。あるいはそのプロジェクトに特に重要であるものがあれば重みづけをして0から10までの数字を入れることになる。例えばプログラマーがその言語を知っていることが大変重要であれば、それを知っている要員には10点、知らない要員には0点を与えることになる。そしてこの得点をたしてもっとも高い得点をもっている要員を選ぶことになる。またもし2人以上の要員が必要であれば、2番目、3番目と点数の高い方から要員を選んでいく。

もっとも重要なスキルはプロジェクト・マネジメントであり、その3つの主要な特性は次のものである。

- ○ 監督の経験と能力
- ○ そのプロジェクトで使われる設備についてのR&Dの経験
- ○ アプリケーションについての知識

これらのスキルはとくに定量化しにくいものであり、それゆえにプロジェクト・マネジャーもしくはチームリーダーを選ぶのはとかく主観的になり易い。そこで選択の候補者について個々のスキルが弱い場合には“W”、強い場合には“S”をつけて分類すると、次のような選択テーブルをつく

ることができる。

	候補者							
要因	1	2	3	4	5	6	7	8
監督経験	W	W	W	W	S	S	S	S
設備経験	W	W	S	S	W	W	S	S
アプリケーションの知識	W	S	W	S	W	S	W	S

このテーブルでいえることは、1番目の候補者はまったく適しておらず、8番目の候補者がもっとも良いということ位である。この間の候補者については、どのスキルがもっとも必要であるか、あるいはどのスキルであれば短期間に訓練することができるかなどを考慮して選ぶことになる。

バックアップ要員の選択は注意を要する。最初はプロジェクトチームのメンバーを選ぶ時点で、DPマネジャーは少なくとも、緊急の場合にチームのメンバーといっしょになって働くことのできる要員を指名しておく必要がある。

もし部門の中にバックアップとして働けるような要員がいない場合には、必要に応じて使えるような余分の要員を訓練することを考慮しなければならない。もし要求されるスキルをもった要員がいなければ、チームメンバーの中のだれかにそのスキルの教育をすることになるし、優秀なマネジャーであればその教育グループの中に余分の要員を入れておくであろう。

特に重要なプロジェクトにおいては、教育中の要員をチームのメンバーとすることもある。彼らは緊急な事態にならない限り特別な仕事をするわけではないが、必要とあれば、すぐに仕事にかかれるように、チームの進行状況や文書の保持などにいつでも気をくばっている。

このバックアップ要員を含ませることのコストの妥当性は容易に解決する。すなわち、そのプロジェクトの遅れによるコストと教育中の要員のコストに加えることの緊急時に、それらの要員を使えることによるコストとがバランスすればよいわけである。

以上をまとめると次の様になる。

要員の配置では次の3つのステップを踏むことになる。

- 個々の仕事のレベルで要求されるスキルを明確にする。
- スキルマトリックスを作成し、保全する。
- 個々のプロジェクトについて、スキルマトリックスによって要員を選択する。

上2つのタスクは特定のプロジェクトによるという種類のものではなくて、ＤＰマネジャーが人事部門と協力して行なうべきものである。最後の要員の選択と割り当てではスキルマトリックスとそのプロジェクトで特に必要となるスキルのリストとを活用する。

1.2.5. スケジューリング

仕事を達成するためにどれだけのマンパワーあるいはどれだけの延べ人数がいるかということを決定する推定はスケジューリングに先だって行なわれる。スケジューリングの目標は仕事を達成するまでにどれだけの時間がかかるかを決定することである。そこでスケジューリングを行なう為には、テストの為に利用できるコンピュータ時間、ユーザー部門の要員を使える時間、管理やその他の仕事に費やす時間などについて明確に決める必要がある。次に個々のプロジェクトタスクについての目標完了日を示す時間表すなわちスケジュールが作られる。

よく見られるプロジェクトの完了日の遅れの多くは適切なスケジューリ

ング手法を使うことによってなくすことができる。正確なスケジューリングというのは正確な推定を意味しているのではなくて、利用可能な要員や設備や見落されがちな要因に十分な注意をはらったり、行なわれるべき事象の順序を示したりすることを意味している。例として、プログラミング担当のマネジャーがもっとも経験の浅いプログラマーにそのシステムの中でもっともやさしいプログラムを割り当てたとしよう。これは当然のことであるように思えるが、プロジェクトにとっては重大なミスがある。というのはそのマネジャーが2つの要因を見落しているからである。最初の要因は、この"やさしい"プログラムが完了日に間に合わせる為のキーポイントであることである。というのはもしこのプログラムが間に合わなければ、全体のプロジェクトがそれを待つことになるからである。もう1つの要因はそのプログラマーに余裕のないことである。そのマネジャーがそのプログラムに1カ月かかるものであると推定したとすれば、彼はそのプログラマーに完了日はちょうど1カ月前にそのプログラムを渡すことになる。結果は、そのプログラムが毎日遅れていって、プロジェクトのすべてのメンバーが何もしないで待っているということになる。これの解決策はこれらの要素を明確にし、それらにスケジュール上で余裕を与えることであろう。これを達成する為の手法にはいろいろなものがある。

よく知られているスケジュールの種類に次の4つがある。

- マイルストーンチャート（milestone chart）
- ウォールチャート（wall chart）
- ガントチャート（Gantt chart）
- ネットワーク（networks）

これらの手法の利点と欠点は以下に述べる。ＤＰマネジャーは、環境や

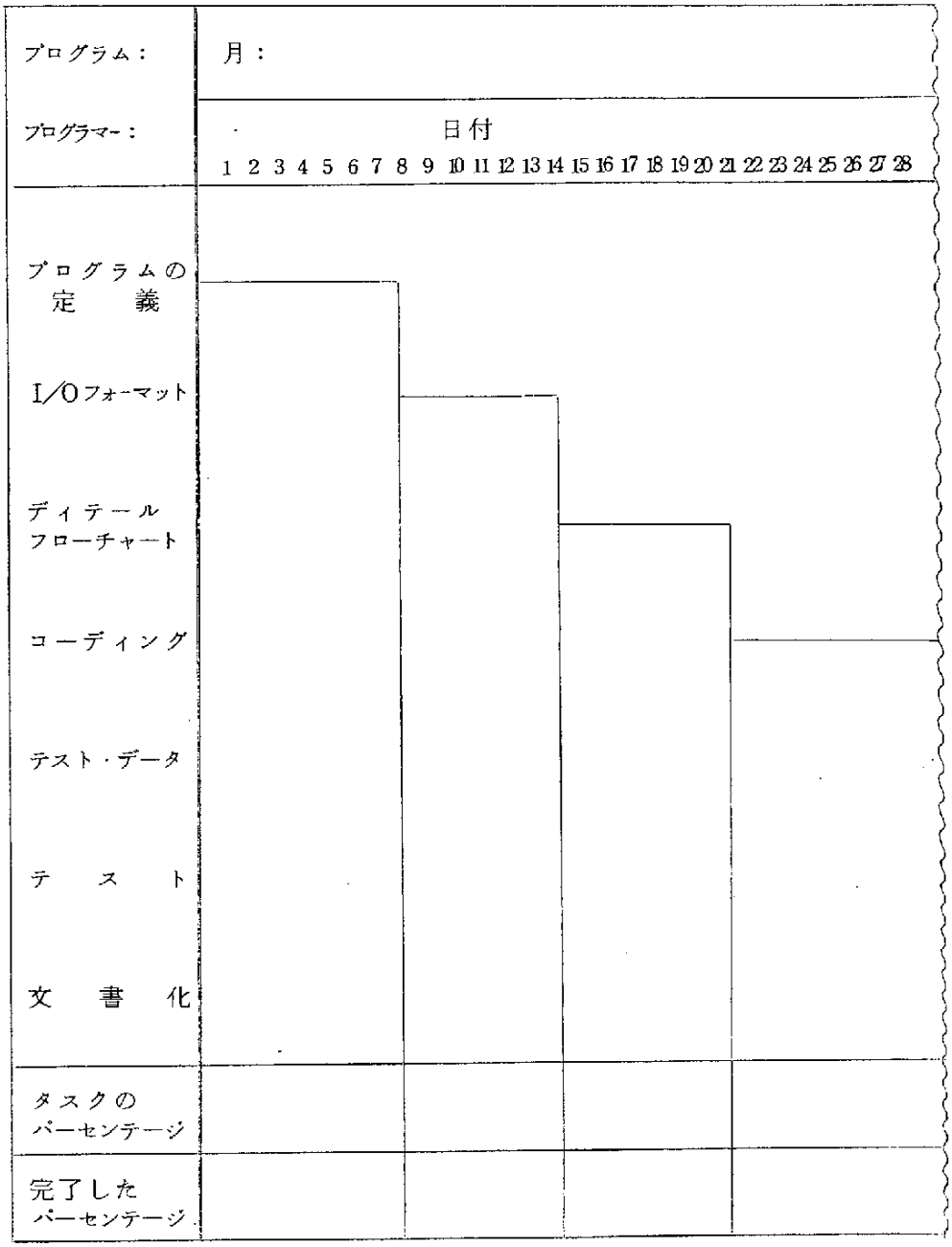


図1－8　ガントチャートの例

ワークロードに見合うようなものを選びそれらをさらに修正して使うべきである。また短期間もしくはさして重要でないプロジェクトではよりやさしいものを好むものである。

A　マイルストーンチャート

もっとも単純でもっとも使いやすいものがこのマイルストーンチャートである。チームのメンバー一人一人に1ページが与えられ、そこに彼が達成しなければならないタスクとその推定時間と目標完了日が書きこまれる。そして実際の完了日とコメントを書きこむ欄を用意しておくこと。この種のスケジュールの欠点は他の仕事と相互に関連することを表現できないことである。

B　ウォールチャート

非常に多くの種類のウォールチャートが売り出されている。ここでは詳しくふれないことにする。というのはその書式は一つ一つがまったく異なっているからである。ウォールチャートの利点はわかりやすく、読み易く、簡単に変更ができることである。しかしながらこれは他の手法に比べて高いものであり、その価値はまだ証明されているとはいえない。

C　ガントチャート

ガントチャートはマイルストーンチャートを図式化したもので例を図1－8に示す。一行目は日付が記入され、1列目には達成されなければならないタスクが記入される。ある日までに完成された個々のタスクのパーセンテージが最後の行に書きこまれる。タスクの進行状況が日付と対応づけられてこのチャートに書きこまれる。このガントチャートは読み易いし更新は容易であるが、標準的な手法とするのには少々複雑すぎるかも知れない。さらにこのチャートでも仕事の相互関係が十分に表現

されているとはいえない。

D　ネットワーク

ネットワーク手法を使えば、今までの手法で欠点としてあげてきた仕事の相互関係をうまく表現することが可能である。ＰＥＲＴはこのネットワーク手法の中でも特に有名なものであって、プロジェクトでのイベントの相互関係を表わすのに最も適した手法である。ＰＥＲＴについてはすでに多くの文献等で紹介されているのでここではその内容にはふれないことにする。

ＰＥＲＴは単にスケジュールに役立つというだけでなくて、リソースアロケーションにも活用することができる。

ネットワーク手法を他の手法と比べてみた場合の利点は、

- 細かいところまで注意をむけさせる。
- 論理的な計画を立てることができる。
- 見落しの危険をへらす。
- それ自体がよいコミュニケーションの道具となる。

などであり、それと反対に欠点は

- 時間がかかる。
- 訂正するのがむずかしい。
- 期待される完了日と実際の完了日を比較するという問題を生じる。

などがある。

1.2.6.　テストのスケジューリングとその管理

スケジューリングを行なうときにもっとも重要であり、しばしば忘られがちなところがテストのスケジュールとその管理である。テストのスケジュールはシステム開発の総括的な部分であり、システム開発全般に責任を持

っている人によって準備されるべきものである。テストのスケジュールには次の３つの目的がある。

- ○　これによってプログラマーとシステムアナリストの両者に、テストを行なう時間表を提供する。
- ○　これは、マネジメントがそのプロジェクトの進捗状況を判定するガイドとなる。
- ○　これをもとにして、必要な設備とともに運営要員を配置する。

テストスケジュールにはどれ位のコンピュータ時間が必要となるかという推定値は含まれているべきであるが、この値は実際と比較しながら修正していくべきである。

1）チェックポイント

テスト計画が立てられたならば、テストの進行状況を検査する為のチェックポイントが選ばれなければならない。チェックポイントは特定の時間間隔もしくはプロジェクト開発のマイルストーンごとに設定される。図１－９に示されているテスト計画の為のチェックポイントは有効であることが実証されている。プログラマー以外の要員によるプログラムテストは必要でないけれども、テストが客観的に行なわれ、文書のテストを行なえるようにしてやることが必要である。

ププログラムテスト

A　あるプログラムのテストの責任をもっているのはだれか？
B　テストにはどのデータが必要か？
C　インプットデータは現実と合っているのか？
D　どんなリソースが必要か？
E　他のプログラムと同時にテストされるプログラムか？

F　どんなアウトプットが期待されるか？

G　テスト結果の見通しの責任はだれにあるのか？

統括プログラムテスト

A　何本のプログラムが含まれるか？

B　プログラムパッケージのテストの責任はだれにあるのか？

C　テストに必要なデータは何か？

D　テスト結果の見通しの責任はだれにあるのか？

システムテスト

A　システムのどこでデータが入力されるのか？

B　テストに必要なデータは何か？

C　どの部分ではシミュレーションテストを行い、どの部分では現実のテストを行なうのか？

D　どこにチェックポイントを設定するか？

E　移行の方法はどのようにするか？

F　テスト結果の見通しの責任はだれにあるのか？

図１－９　テスト計画のチェックリスト

もっとも注意を払わなければならないのは品質管理である。品質管理は次の３つの点をもとにして行なう。

- テストの適合性
- スルーネス（thoroughness）
- 必要となるリソース

次のような点についての標準を設定すべきである。

- 使用されるデータの種類

○　インプットとアウトプットの制限

○　テストの規定（すなわちテストに使われる材料、テストを順序づける手順など）

○　テストの分析（すなわちデバック技術、特別の診断ルーチンなど）

プログラムとシステムテストにおけるもっとも重要なチェックポイントはテスト計画の完了日である。システムテスト計画の内容が図1－10に示してある。責任の範囲とテストの管理がこのテストプランに詳しく述べられていることに注意してほしい。

1.0　一般的な情報
　1.1　表　紙
　1.2　目　次
　1.3
　1.4　参照項目
2.0　一般的なテストの構造
　2.1　テストの目的
　2.2　責任の範囲
3.0　一般的なテストの管理
　3.1　インプットの管理
　3.2　処理の管理
　3.3　アウトプットの管理
4.0　一般的なテストの手順
　4.1　テストの約束
　4.2　テスト結果の処置
　4.3　テスト結果の文書化
　4.4　テスト結果の分析

5.0　テストのスケジュール

5.1　テストの順序

5.2　テストの数と頻度

6.0　テストーケース1

6.1　必要なリソース

6.2　トランザクションとファイルの例

6.3　期待されるアウトプットとファイルの例

6.4　テストの手順

図1－10　システムテスト計画

2）プログラムテストの計画

プログラムテスト計画が図1－11に示してある。これを前もって準備しておけば、R＆D担当のマネジメントがその内容を調べ、進捗状況を報告することができる。

エラーが見つかったら、その内容を訂正するとともにテストは最初から行なわれるべきである。というのはその修正がそれまでのモジュールに影響を与えないことを確かめる必要があるからである。テスト計画の修正は必要に応じてたびたび行なうべきである。

進行状況の報告はスケジュールと密接に関連している。スケジュールが作成された時点で、現実との比較をし、その評価をすることができるような形で進行状況の記録がとれるようにしておくことは重要なことである。この報告と監査については次のところでふれることにする。

1.0　一般的な情報

1.1　プロジェクト番号

1.2　プロジェクト名

1.3　プログラム番号

1.4　プログラム名

1.5　プログラマー

1.6　テストの総回数の期待値

2.0　テスト－ケース1

2.1　ロジックテスト番号

2.2　目　的

2.3　テストの回数の期待値

2.4　個々のテストの結果

図1－11　プログラムテスト計画

1.2.7.　進捗報告

プロジェクト計画に対するプロジェクトの進行状況を評価する能力こそプロジェクト・コントロールのかなめである。しかし進行状況の評価は正確なかつ時を得た進捗データの収集と報告によるところが大きい。

進捗報告の主要な要因は次の3つであろう。

- チェックポイントの指定
- 進行状況の記録
- トラブル報告書

1）進捗報告のチェックポイント

進行状況のチェックポイントの設定はプロジェクト計画の段階で、DPマネジャーとプロジェクト・マネジャーが協力してこれを行なう。進

行状況のチェックポイントの中に最低これだけは含まれていなければいけないという項目のリストを図１－１２に示す。チェックポイントの数というのはプロジェクトの大きさに範囲によって多くしていくことになる。

1. システム研究の終了
2. システム分析の終了
3. システム設計の終了
4. コーディングとコンパイル
5. 初期のプログラムテスト
6. 最終のプログラムテスト
7. システムテスト計画の完了
8. システムテスト－中間監査
9. システムテストの終了
10. 移行に先立つ準備の終了

図１－１２　進捗報告のチェックポイント

システムの設置ごとにまたプロジェクトごとにそこでの状況というものは変わってくるから、チェックポイントの設定にあたっては、効果的なプロジェクト・コントロールをするのに十分な細かさでチェックポイントの間隔を設定することが重要なことになってくる。この間隔についてのルールの一つは、少なくとも一カ月に一度のチェックポイントは必要である、というのがある。重要なプロジェクトでは２週間ごとあるいは３週間ごとというのが望ましいであろう。

もう一つのルールとしては、約１５人　週に１回のチェックポイント

を設定する、というのがある。これによれば、５人のプロジェクトでは３週間ごとにチェックポイントを設けることになり、５人以上のプロジェクトであれば、少なくとも一週間に一度はチェックポイントを設けなければならないことになる。チェックポイントの分割は、各フェーズに割り当てられた要員の数に従ってプロジェクトのいろいろなフェーズで調整されることになる。プロジェクトが最初からつまずいたりしたら、さらに細かいチェックポイントがスケジュールに組み込まれるべきである。

スケジュールが変更されたら、進捗報告の中でその日を明確にすべきである。チームのメンバー一人一人が個々のチェックポイントの存在を認識し、そこで彼に何が期待されているかをはっきり理解する必要がある。

2） 進行状況の記録

報告書は個々のチームメンバー（すなわちプログラマーとシステムアナリスト）からプロジェクト・マネジャーのところに行き、それからＤＰマネジャーのところに行くというのが普通である。プロジェクトの記録をとるという仕事は、もし個々のメンバーの報告の責任が明確にされていなかったり、強要されていなかったりすると、あいまいなものになりがちである。

“正確な”報告の必要性という点に重きがおかれるべきである。プログラムが“９０％完了した”という報告が出されるような状況は極力避ける必要がある。この達成する方法としては、個々のメンバーに割りあてられたタスクを完了させる為にはあとどれだけの時間が必要であるか推定させる方法がある。この推定法は完了したパーセンテージの推定よ

りも信頼できるということがわかっている。もし完了した割合が必要であれば次のように算定すればよい。

$$\frac{\text{今日までの消費時間}}{\text{今日までの消費時間＋完了までの推定時間}}$$

進行状況の記録としては次の３つの文書がよいであろう。

- チェックポイントのスケジュール
- メンバーごとの時間要約
- 公式な進捗報告書

3) チェックポイントのスケジュール

前に述べたチェックポイントのスケジュールは進捗報告書にもとづいて、それぞれのチェックポイントで更新されるべきである。もちろん、全体的なスケジュールも同じようにしてチェックポイントのところで調査されるべきである。またタスクの期待完了日が修正されるにつれて対応するチェックポイントの日付も更新されることになる。この仕事はＤＰマネジャーもしくはプロジェクト・マネジャーによってなされることになるが、もし後者である場合にはＤＰマネジャーの承認を受けなければならない。

4) メンバーごとの時間要約

この時間要約はタスクごと、チームメンバーごとの実際の消費時間を示しているものである。この文書は現在使っているタイムシートに必要に応じて修正を加えたもので十分である。この報告書は週毎、２週間毎、１カ月毎位に出される。まわりの状況に特別なものがなければ、これは少なくとも１カ月に一度は提出されるべきである。

チームのメンバーごとに提出される時間要約をもとにして、プロジェ

クトマネジャーは彼自身の進捗報告書を準備する。この進捗報告書は個個のチェックポイントで準備されるべきである。この報告書には作業の内容、所定の人・日とお金の推定値、累積値、完了日の予測などが含まれる。さらにこの報告書の中にはこの報告期間に起こった問題が物語風に書き込まれると同時に、前の報告書で述べられた問題の結果が報告される。またプロジェクトマネジャーは報告した問題点に対してどのようなアクションをとっているかを明確にしなければならない。

まとめてみると、まず一定期間ごとにチームメンバーによって時間要約が進捗記録として準備される。これをもとにしてプロジェクトマネジャーが進捗報告書を準備する。そしてプロジェクト・スケジュールと対応するチェックポイントのスケジュールが進捗状況に見合うように調整される。

5) トラブル報告書

プロジェクト・コントロールシステムの一つの目標は、潜在的にあるトラブルがひどい状態にならない前にそれに光をあてることである。チェックポイントを細かに設定することと、進捗報告書を細かに分析することがこれに対するキーポイントとなるが、それに加えて、これらの問題を報告する何らかの非公式な手続きを設けることは有効である。プロジェクトチームの各メンバーは自分に割り当てられたタスクの完了には気を配ってはいるが、それ以上にすべての要員がプロジェクトの全体的な成功に責任があることを忘れがちである。もしこの態度が適切に心にいだかれていれば、いずれのメンバーも彼が起こりそうであると考える問題をプロジェクトマネジャーに報告することになる。例えば、もしプログラマーが最初推定された時間内でブロックダイアグラムを書くこと

が不可能であると考えたときには、プロジェクトマネジャーに対してコーディングが計画通りには終わらないことを報告すべきである。逆にロジックの設計を十分時間をかけておこなえば、コーディングの時間を減らすことができるであろう。プロジェクトマネジャーはスケジュールの調整にあたって、これらのコメントを考慮に入れるべきである。

しかしながら良い報告書が唯一の重要な要因ではない。マネジャーはこの報告書を分析しコントロールサイクルを完了させる為に適切な手を打てるようにすべきである。

1.2.8. 監査と進捗分析

プロジェクト計画の一端として、プロジェクト・マネジャーとDPマネジャーの行なわなければならない仕事は、どんなチェックポイントが必要であるかということだけではなくて、どんな点をだれがそれぞれのチェックポイントで監査すべきかを決めることである。目標は問題が大きくならない内にそこに光をあてることであるといつでも心に描いているべきである。

1) 進捗の監査

図1－13は、チェックポイントでの監査、責任、分析についての提案等を含んだ監査と管理のガイドである。

このチェックポイント・リストの使用例として"研究の終了"というチェックポイントを考えてみよう。このリストによれば、DPマネジメントと同様にユーザー部門のマネジメントが監査に参加すべきである。監査の方法は注意深く考慮されるべきである。例えば、研究の終了を監査する一つの方法は、このリストに述べられているように、注意深く研究レポートを読むことである。これは明らかに十分ではあるが、しばし

ば失敗することがある。もしこの時点での監査が何らかの意味をもっているならば、監査に責任をもっている部門によってその報告書が読まれ、理解され、質問されるべきである。

この監査のガイドには特に注意を払うべき項目がリストされている。例えば研究の終了というチェックポイントにおけるこれらの項目の一つに“現在のシステムのまとめへのユーザーの参加”がある。監査が適切に行なわれないと、これらの項目は重大な問題を引き起こし、多大な支出を強いられることになろう。

起こりがちな問題とリストされているものは、ほんとうによく起こる。詳細を不十分なままにする、あるいは不正確にしてしまうというのはシステム開発のこの段階における共通問題であり、その問題を解決するまでプロジェクトが前に進まないようにするのはユーザーとＤＰマネジメントの責任である。それを訂正する方法もまたこのガイドに示されている。

チェックポイント：プロジェクトプラン
監査をする人：ＤＰマネジメント、ユーザーマネジメント
方法：報告書の読書とプロジェクトリーダーのインタビュー
監査項目：完了したタスクのリスト、マネジメント責任図、
　　ユーザーもしくはＤＰ部門の承認、監査と管理作業の達成
起こりがちな問題：タスクを見落す
訂正方法：レドープラン（Ledo plan）

チェックポイント：推定の完了
監査をする人：ＤＰマネジメント、ユーザーマネジメント
方法：報告書の読書とインタビュー、コンサルタント標準、
　　ベンチマークテスト
監査項目：詳細にいたる検討、計画の確認
起こりがちな問題：楽観的な推定
訂正方法：推定の修正、その推定値での書きかえ

チェックポイント：スケジュールと予算の完了
監査をする人：ＤＰマネジメント、ユーザーマネジメント、
重役
方法：報告書の読書とインタビュー
監査項目：リソースの仮定、緊急なタスク、使われているコスト要因
起こりがちな問題：ロストタイムの未考慮
訂正方法：さらに細かく監査ポイントを設定する

チェックポイント：初期のシステム研究
監査をする人：プロジェクトマネジャー
方法：インタビュー
監査項目：範囲の拡張、ユーザーは満足か
起こりがちな問題：ユーザーのインタビューが不可能である
訂正方法：ユーザーと会う

チェックポイント：研究の終了
監査をする人：ユーザーマネジメント、ＤＰマネジメント
方法：詳細に報告書を読む、討論、研究チームのインタビュー、ユーザーのインタビュー
監査項目：発見された例外、ユーザーの観察、現在のシステムのまとめに対するユーザーの承認、範囲の拡張
起こりがちな問題：詳細部分があいまいになる、例外事項の不完全な詳細
訂正方法：研究チームの続行の要請

チェックポイント：システム分析の完了
監査をする人：ＤＰマネジメント、ユーザーマネジメント
方法：討論
監査項目：代替案、真実の要件、設計時間の推定
起こりがちな問題：機械化しない解決案を拒否する
訂正方法：代替案の中にコストの推定値を入れる

チェックポイント：データベース設計の完了

監査をする人：ＤＰマネジメント、ユーザーマネジメント

方法：報告書の読書、ユーザーとの討論、プロジェクトチームへの質問

監査項目：ユーザーの監査ファイル、古いファイルへの新しいデータベースの適合性

起こりがちな問題：不完全性

訂正方法：必要とされる追加項目の評価

チェックポイント：システム設計

監査をする人：ＤＰマネジメント、ユーザーマネジメント、重役

方法：報告書の読書、プロジェクトチームへの集中的なインタビュー

監査項目：互換性、例外処理、管理、エラー訂正手順、プログラミング時間の推定

起こりがちな問題：要件が不完全

訂正方法：無視した項目がこのプロジェクトに含まれるべきか決める

チェックポイント：コーディングの完了

監査をする人：プロジェクトマネジャー

方法：プログラマーのインタビュー、プログラムの検査

監査項目：標準、言語の活用、ロジック設計の正確さ

起こりがちな問題：ロジックエラー

訂正方法：書き直し

チェックポイント：テスト

監査をする人：プログラマー、プロジェクトマネジャー

方法：テスト結果を見る

監査項目：結果－仕様、進行状況－テスト計画

起こりがちな問題：すべてのテストが行なわれない

訂正方法：より多くのテスト

チェックポイント：システムテストの完了
監査をする人：ユーザーマネジメント、ＤＰマネジメント
方法：テスト結果を見る
監査項目：テスト結果－システムテスト計画、テスト結果－
仕様
起こりがちな問題：無視する
訂正方法：再びテストするか訂正する

チェックポイント：移行
監査をする人：ユーザーマネジメント、ＤＰマネジメント
方法：討論
監査項目：移行計画、移行のスタッフ
起こりがちな問題：不適切なスタッフ
訂正方法：移行努力を単純にする

図１－１３　チェックポイントにおける監査と管理のガイド

2) プロジェクト進捗の分析

各々のチェックポイントで作成される進捗報告書はプロジェクト進捗の分析における重要な文書である。この報告書には開発中のシステムの各々のフェーズについての完了率の推定値を含んでいる。これらの数字がどのフェーズが、あるいはどのフェーズの部分が計画通りであるか、進んでいるか、あるいは遅れているかを決定する為に、計画上の数字と比較される。スケジュールより遅れているものに対しては、それが今回限りのものであるかあるいはこれからも起こりそうなものであるかを解明するために分析が行なわれるべきである。これらの分析をもとにして、どのような処置をとるべきかをマネジメントが決定する。

3) 修　正

個々の進捗報告書が分析され、問題に対する訂正処置がとられた後で、今日までの進捗に見合わせてすべてのスケジュールと予算が修正される。スケジュールを修正する前に、残っているフェーズもしくはプログラムについては新しい推定をしなければならない。残った仕事に対する新しい推定値は終わっていない仕事の数字にのみ考慮され、もとの数字はすてられる。

たとえば、ある特定のプログラムがロジック設計、コーディング、テストは8人・日かかるものと推定されていたとしよう。10人・日の努力が費やされたのちに、そのプログラマーが10%だけ完了したと報告したとする。残りの90%がこれまでの10%と同じペースで仕事が進められるとすれば、残った仕事に対する推定は90人・日ということになる。

1. 2. 9. コストの分配

コストを分配し、予算額と比較するのは、ＤＰマネジャーと委員会の機能である。プロジェクト・コントロールのこの領域は、前もって決められた会社のポリシーや手続に従って行なわれることになる。例えば、コスト報告書の書式についてはＤＰマネジャーがほとんど選択できないように、この種の文書のほとんどの大部分は前もって決められていることが多い。こういうわけであるから、詳細なコストの分配についての文書についてはここでは特に触れないことにする。

しかしながら、ＤＰマネジャーが責任をもっているシステム開発コストの分配については様々な問題がある。というのはコストの分配はプロジェクトコントロールの一つの側面である。

システムを開発するコストは使われているリソースの種類に応じて記録したり分析したりすべきである。システム開発では３つの種類のリソースが使われる。

- 人・月というような単位で測定されるマンパワー
- ＣＰＵ使用時間などの単位で測定されるコンピュータ使用時間
- ドルで表わされる備品の支出などの間接コスト

さらには新しい設備の購入または設置のコストが一つのプロジェクトに割り当てられることが時々ある。その他に考慮しなければならない間接コストとしては、一時的な要員教育のコスト、コンピュータ以外のＤＰ設備の購入並びに設置コストあるいはプロジェクト・コントロールシステムなどの手続きのコストなどがある。

マンパワーコストは、被雇用者のトータルコストを基本にして算定されるべきである。被雇用者に直接割り当てられるコストとしては彼のサラリ

一だけであるが、雇用主が払うべきものとして保険、社会保障費、休暇などがある。一般管理費などのその他のコストは、マンパワーの直接コストに含ませるべきではなくて、備品などと同様に間接コストとして取り扱うべきである。雇用者ごとの時間に関するコストは、彼が特定のある時間にどんな仕事をやっているのかに関係なく、すべての時間を費やしているものとみなすべきである。

コンピュータ時間を割り当てる方法には2つある。一つは実際のコストを算定する方法である。まずDP運営部門の総コストが累積され、機械時間で割られて平均時間当りのコストが算定される。そしてプロジェクトごとの時間がかけられて、総コンピュータ時間コストが計算される。この方法の欠点は特定のプロジェクトのコンピュータ使用時間がたとえ一定であっても、そのプロジェクトへのコストの割り当てが月毎に変わることである。

二番目の方法、すなわち標準コストはこの欠点を取り除く方法として開発されたものである。この方法では、実際に生産的であった時間数だけが考慮に入れられる。コンピュータ時間の時間当りの標準コストは、DP部門の推定総コストを生産的な標準時間で割ることによって得られる。この標準コストに個々のプロジェクトで使用された時間をかけることによって得られたコストが、個々のプロジェクトに請求されることになる。もし実際の時間とコストが標準と同じものであれば、DP部門のすべてのコストはすべて分配されることになる。もし差がいつでもあったり、大きかったりしたら標準が修正されるべきであるということである。

この標準コストの利点は個々のプロジェクトの請求はコンピュータ時間の使用に比例していることである。これによって予算を立てることができ

るし、実際のコストに対して予算コストのより意味のある分析をすることができる。

進捗報告書を評価するにあたっては、時間の比較をするのと同じ要領で、予算面でプロジェクトがどれ位終わっているかを決定する為に支出と予算の間での比較がなされる。

そのシステム開発がどうしてもっと経済的に行なうことができなかったかという質問に答えるのに、このコスト分配の値が様々な形で使われる。これらの質問のいくつかを次にあげる。

a　経済的な要員をやとって彼らを訓練するのがよいか、あるいはこの訓練コストを省く為に高いサラリーを払うのがよいか。

b　あるプロジェクト、あるいはフェーズではコンサルティング会社を雇った方がより経済的であろうか。

c　現在の設備のグレードアップを考えるよりも外部のコンピュータを借りた方がお金の節約になるのではないか。

d　デバックあるいはプログラミング時間をへらす為に新しいソフトウェアに投資すべきであろうか。

e　現在以上にマンパワーが必要になったときには、新しいスタッフを雇うのと現在のスタッフに残業をさせるのとではどちらが経済的か。

2. システム開発部門の組織

2.1 システム・ライフ・サイクルにおけるＤＰ組織の考え方

一般にいかなる仕事を行なう場合でも、それを遂行すべき組織のあり方が仕事の成果を大きく左右する。特にＤＰ組織では、販売・経理その他様々な組織体の中でも比較的歴史が浅く、組織づくりの伝統が必ずしも確立されていないこと、また以下に述べる複雑な要因・条件に制約されること等から、その編成については十分な考察が必要である。いわば、ＤＰ組織の編成およびその運用はデータ・プロセシング全般の成否を握る鍵ともいうべきものに他ならない。この章では、このような観点から、ＤＰ組織に関する諸問題を論じることとする。なお、いうまでもなく、抽象的・一般的にどこまでも通用する組織というものは存在しない。組織がつくられ、機能するのは、あくまで具体的・個別的な“場”においてである。従って、以下この章では、ＤＰに関する諸機能を全て包摂する典型的なＤＰ組織―“理念型”としてのＤＰ組織―を基調として記述するが、読者が自らのＤＰ組織を検討されるに当たっては、要求される諸条件にもとづき適宜取捨選択あるいは変形・加工していただきたい。

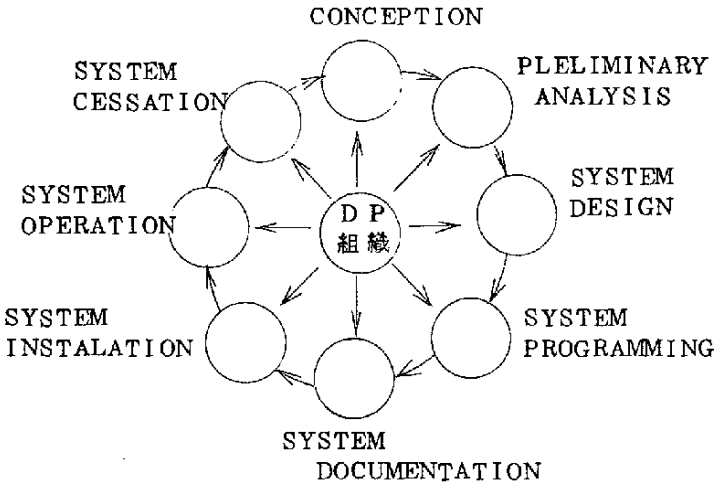
2.1.1 システム・ライフ・サイクルとＤＰ組織

ＤＰ組織は、システム・ライフ・サイクルとの関連で捉えると、システムの調査・開発から実施・運用、そして次期へのレベル・アップに至るシステム・ライフ・サイクルの全過程を管理する軸として考えることができる。（図参照）このことから、ＤＰ組織はおよそ次のような特徴を持つこととなろう。

第一は、期待される機能が多様であるということである。ＤＰ組織は、

単にシステムの開発のみを行なうのではなく、またシステムの維持・運用のみを行なうのでもない。いわば、それら全てを管理するのである。しか

図　システム・ライフ・サイクルとＤＰ組織



も、ＤＰ組織が管理すべきシステムは、通常一つにとどまらず、幾つかのシステムが併行的に推進されており、さらにそのそれぞれのシステム・ライフ・サイクルのステージは必ずしも同一過程にある訳ではない。こうして、ＤＰ組織が果たすべき機能はきわめて多様で変化に富んだものとなる。

第二の特徴は、ＤＰ組織のあり方が対象となるシステム―幾つかのシステムを対象とする通常の場合は、そのうち主要なシステム―のシステム・ライフ・サイクル上のステージによって規定されるということである。即ち、ＤＰ組織が管理すべきシステムが今どのような段階にあるのか、予備調査の段階なのか、設計の段階なのか、それとも維持・運用の段階なのか、という事情によって、ＤＰ組織のあり方も多かれ少なかれ変わらざるを得ないことになる。

第三の特徴は、ＤＰ組織が、通常、きわめてスタッフ性の強い仕事と、逆にきわめてライン的性格の強い仕事とを併せ持つことになるということである。システム・ライフ・サイクルの中でも、特にシステムの調査・分析あるいは基本設計というステージは、非常にスタッフ業務的性格が濃い。これに対して、プログラムのコーディングやコンピュータのオペレーションという仕事は、どちらかというと定常的・繰り返し的なライン作業である。ＤＰ組織は、いわば工場とそのスタッフを併わせ持つ組織であるということができる。ＤＰ組織がしばしば部品製造業と比較されるのも、こうした事情によるものに他ならない。

2,1,2　ＤＰ組織編成の基本要件

さて、具体的なＤＰ組織を編成するに際しては、幾つかの要件を踏まえなければならない。

その一つは、およそ組織と名のつくものを検討するときには必ず踏まえるべき一般的要件である。そのような要件としては、責任・権限の明確化であるとか、管理スパンの適正化であるとかが挙げられる。これについては、読者は様々な組織論によって学ぶことができよう。

二つ目に留意すべきは、ＤＰ組織に固有の要件である。前節に述べたようにＤＰ組織は、他の組織に比して特徴的な点を有している。従って、ＤＰ組織の編成に当たっては、期待される多様な諸機能を適切かつ有機的に配置すること、システムのライフ・サイクルに応じ変化する要請に対応しうる弾力性・機動性を確保すること、あるいはスタッフ的な業務とライン的な業務との間に円滑かつ効果的な関係を樹立すること等、ＤＰ組織に固有な特徴を十分に織り込むことが必要である。

なお、その際、ＤＰ組織は、その母体である全体組織から見た場合、

コンピュータを核として集結した非常に専門家的な集団となることも念頭においておく必要がある。

2.1.3　ＤＰ組織編成に当たって考慮すべき具体的条件

前節で述べた基本要件を踏まえつつＤＰ組織を編成するに当たっては、次のような具体的諸条件を考慮に入れなければならない。

第一は、ＤＰ組織にとって、いわば外的な条件であって、そのようなものとしては、母体となる全体組織（例えば企業）の目的・性格・規模・要員事情等等が挙げられる。例えば、公共へのサービスを仕事とする自治体と利潤の極大を最終目的とする企業とでは、ＤＰ組織に対する要請は異なったものとなろう。また、ある企業が、全体としてライン組織を主としている場合と、ライン・スタッフ組織を主としている場合とでは、ＤＰ組織のあり方も変わってこよう。さらに、その企業全体の要員が極めて逼迫しているのか、それとも比較的潤沢であるのかという事は、ＤＰ組織とその要員にとって、非常に現実的な規制条件となるであろう。

考慮すべき第二の条件は、いわばＤＰ組織の内的条件である。内的条件としては、ＤＰ組織自体の歴史・規模・要員事情、あるいは対象とするシステムの範囲・規模等々がある。例えば、ＤＰ組織が比較的大きい場合は、その内部の編成として後述する機能別編成が採用されうるが、小規模な組織では、そうした編成は非効率的なものとなるであろう。また、ＤＰ組織の要員がどの程度養成されており、またどのようなルートで補充・育成されるのかということは、ＤＰ組織自体のあり方を大きく規制することとなろう。あるいは、対象とするシステムの規模や範囲が大がかりな場合と比較的小型のシステムの場合とではこれを管理すべき組織のあり方も変わってしかるべきである。

さて、これまで、システム・ライフ・サイクルとの関連でＤＰ組織の編成についての基本的考え方を論じてきた。冒頭でも述べたように、この章では、いわば理念型としてのＤＰ組織を基調として論を進めている。従って読者は、以下のＤＰ組織に関する具体的記述を読まれる際は、上記のような視点から自らの具体的条件を整理・把握した上で、必要な選択・変更を行なっていただきたくあらためてお断り申し上げる。

2.2　ＤＰ組織の機能

まず、組織を編成するに当たっては、その組織にどのような機能・役割が期待されるかが定められなければならない。そこで、この節では、ＤＰ組織に通常期待される機能について考察を進める。

2.2.1　ＤＰ組織の主な役割

一般に、ＤＰ組織は、大きく分けて次の３つの役割を担っていると考えられる。

1）ＤＰシステムの総合企画・調整

ＤＰシステムに関する総合的な企画・調整の役割、換言すれば、所謂戦略・戦術を策定する任務である。

2）ＤＰシステムの設計・開発

具体的なＤＰシステムを設計・開発する役割、いわば策定された戦略・戦術に従って実際の「システム作り」を遂行する任務である。

3）ＤＰシステムの維持・運用

そのようにして設計・開発されたシステムを維持・運用する役割、別な面から、即ち道具立ての面から見るとコンピュータを運用・管理する任務である。

これら三つの機能は、システム・ライフ・サイクルにおける三つの主要局面にほかならない。以下、それぞれの機能について、やや詳細に述べることとする。

2.2.2　ＤＰの総合企画・調整

この機能において、何より重要なことは、ＤＰシステム推進に関する長期プランが策定されなければならないということである。ＤＰシステムは、その性格上、一度開発あるいは導入されると、容易にもとの状態に復しがたい。それだけにＤＰシステムを開発・導入するにあたっては、見通し得る範囲で、長期のビジョンを明らかにしておくことが望ましいわけである。

この種の長期プランは、できる限りその組織全体の長期計画（企業においては経営計画）の一環としてそれとの斉合性を確保しつつ、検討されることが望ましい。従ってまた、そのプランは、コンピュータの分野のみにを奪われず、全体の管理システムの方向の中に位置づけつつ策定されることが望ましい。しかも、システム・ライフ・サイクルが比較的長期にわたることから、その間の環境・条件の変化を適宜織り込み、これを常にアップ・トゥ・デイトなものとしてゆく努力が大切である。このようにして作成された長期プランは、今後のＤＰシステム開発・推進に関する戦略的方向を指し示すものとなる。すると、次にこの戦略を具体的に実現するための手段・方法が問題となる。このようにして、総合企画・調整機能の次の主な仕事は、システム開発プロジェクトの調整およびこれに伴うコンピュータ関連リソースの投入調整である。このためには、長期プランに従って、個々のシステム開発プロジェクトに優先順位が設定され、高価でかつ限られた人・物・金という諸資源―即ちコンピュータ等ハードウェア、コンピュータ要員など―をどのように効率的に配分し使ってゆくかが、検討され、

決定される。システム要員の採用・教育・配置計画も、この過程で検討される。このようにして、ＤＰシステムの実施計画が確定される。

上に述べた総合計画・調整の機能は、きわめてスタッフ的な仕事である。

そして、システムの規模が大きければ大きい程、システム・ライフ・サイクルが長ければ長い程、その任務は重要になるということができる。

2.2.3　システムの設計・開発

これは、確定されたシステム実施計画に基づいて、個々のシステムをつくる機能である。この機能は、更に二つに大別することができる。

その一つは、システムの仕様を確定する仕事である。これによって、何を目的とし、どの様な機能を包摂したシステムをいつまでに開発するかが定められる。勿論、その場合に新しいコンピュータ・リソースが必要であれば、その仕様・納期等を定めることも、これに含まれる。

その二つは、確定された仕様にもとづき、プログラムを作成しテストする仕事である。これによって、構想されたシステムは、現実にコンピュータによって処理されうる形態をとることとなる。

これらシステムの設計・開発機能には、所謂システム・メンテナンスも含めて考えられよう。また、設計・開発業務そのものを効率的に進めるための技術的活動も、この機能の一環として捉えることができよう。

2.2.4　システムの運用・オペレーション

一度設計・開発されたシステムは、その運用過程で様々な人間の関与・介入があるにしても、コンピュータによって処理・運用されることとなる。

そして作り上げられたシステムがいかに良いシステムであっても、コンピュータを道具とする日々の処理・運用が適切に行なわれなければ、その所期の狙いを達成することができない。この任務によく応えることが、コ

ンピュータのオペレーションを軸とするシステム運用機能の役割である。

その機能には、その他にも入力データの作成（カード穿孔等）、用紙等の購入・在庫管理などが含まれる。この運用・オペレーション機能は、いわば工場に比較しうるきわめてライン的な仕事である。それだけに地味ではあるが、基本的な仕事であって、その良し悪しは、日々の情報処理の効率を大きく左右することになる。

2.3 DP組織の位置づけ

さて、それでは、DP組織は、企業・官庁等の全体組織の中でどのような位置に置かれるべきであろうか。実は、この問題は、当該DP組織が具体的に上記諸機能をどの程度まで具備しているかという事によっても大きく変わってくる。それというのも、実際に問題となるDP組織は、上記諸機能を全て併わせもつ典型的な形態をとる以外にも、システム設計・開発機能とその運用・オペレーション機能のみを併せ持ったり、場合によっては運用・オペレーション機能のみを有したり、多様な形態をとることがあり、それによってDP組織の置かれるべき位置も当然変わってくるからである。以下の記述は、ここでも典型的なDP組織を基調として進めるが、読者はその事を念頭において読んでいただきたい。

2.3.1 ユーザー部門への編入

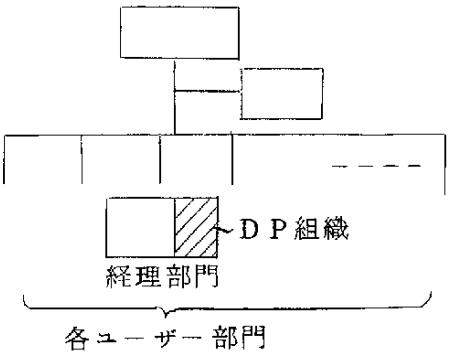
DP組織を全体組織の中へ位置づける一つの形態は、所謂システムのユーザー部門にこれを編入する形態である。例えば、主として経理業務のEDP化を担当するDP組織であれば経理部門にこれを置き、生産管理システムであれば生産管理部門に設置するという形がこれである。

この形態は、ユーザーとのコミニュケーションが十分に確保され、使う

立場に立ったシステム設計、オペレーションが行なわれるという良さを持っている。しかし、その反面、運用如何によっては、ＤＰ部門がその自主性を喪失し、ユーザーの請負部門に堕する危険を孕んでいる。

一般に、この形態は、ＤＰ組織が取扱うシステムが比較的特定の分野に限

表２－１　ユーザー部門への編入



定されているような場合に有効であろう。従ってＥＤＰ化の初期の段階で見られる形態でもある。

2.3.2　管理部門への編入

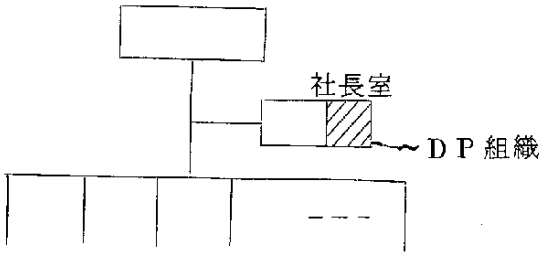
第二の形態は、ＤＰ組織を、社長室や総務部といった所謂管理部門に設置する方法である。

この形態では、他のユーザー部門から等距離に立つこととなるため、各ユーザー部門のニーズを公正に捉えて総合的なシステム化を図ってゆく上で効果が挙がろう。また、その場合に、当該管理部門から種々のサポートを比較的容易に受けられるのも、この形態の良さであろう。しかし、ユーザー部門との間にカベが出き易い点は否めない。

こうしたことから、この形態は、システム開発が比較的進んだ段階で、

各システム間のつながりが問題とされ、総合的な視野が求められる場合に有効であろう。換言すれば、先に述べたＤＰ組織の機能のうち、第一の総合企画・調整機能が特に重要視されるような場合に、この形態を採用することが望ましい。

表２－２　管理部門への編入



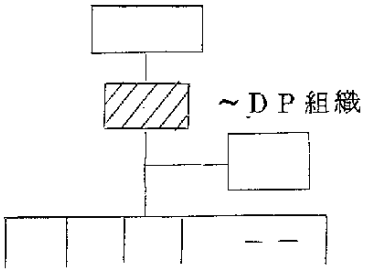
2.3.3　トップ・マネジメント直轄

第三の形態は、ＤＰ組織を社長その他トップ・マネジメントの直接の指揮・命令下に置く形態である。

この場合は、トップの意向がＤＰシステムに迅速・的確に反映される。

またＤＰ組織は他部門ユーザーに対し極めて強い立場に立つことになるため、強力なリーダー・シップを発揮し易いという長所を持つ。しかし、管理部門へ編入する場合と同様、組織運用に当たっては、ユーザー部門とのカベをつくらぬよう注意することが必要となる。

表２－３　トップマネジメント直轄



これらのことから、この形態は、全体組織全般に及ぶ大規模なシステムを開発したり、画期的な考え方を導入したりするような、特に強力なリーダ・シップが求められる場合に極めて有効なものとなる。

2.3.4　多事業所体制とＤＰ組織

これまで述べてきたのは、いわば一つの全体組織がある場合に、ＤＰ組織をそのどこへ位置づけるべきかという問題であった。しかし、中には全体組織が一箇所に集中しておらず、幾つかの、あるいは多数の事業所を有している場合がある。そのような場合に、ＤＰ組織はどのように組織されるべきであろうか—この点について、最後に触れておきたい。

考えられるのは、一つは、各事業所ごとにそれぞれ独立したＤＰ組織を設ける形態である。その場合、それぞれのＤＰ組織は、自らが属する事業所のトップによって管理されることとなる。もう一つの形態は、本社もしくはそれに対応する機構に、全事業所にわたるＤＰ組織を設ける形態である。その場合、ＤＰ組織のメンバーは実際には各事業所に必要に応じ配置されることになろうがそれらを管理するのは、本社（あるいはそれに属するＤＰ組織）である。このように、多事業所体制におけるＤＰ組織の形態には、分散型と集中型の二つのあり方が考えられる。

表２－４　分散型

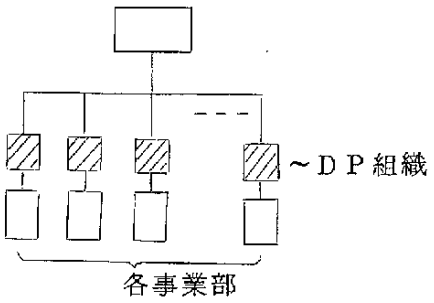
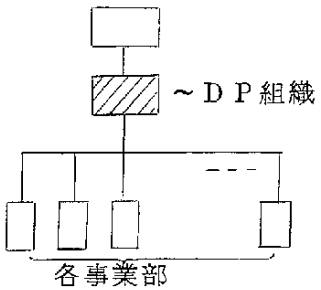


表２－５　集中型



分散型は、各事業所の現実に即応し、その特殊性を活かしたシステム化が図られる良さを持っている。しかし、限られたリソースを組織全体で最適に配分・調整したり、システム開発努力の重複を回避したりするためには、集中型の方が効果が挙げられよう。

2.4 ＤＰ組織の内部編成

前節では、幾つかの機能を有するＤＰ組織の全体組織の中での位置づけを見てきた。ひとたびこの点が決まると、次の問題は、ＤＰ組織自体をシステムに関する仕事に対してどう編成するかという事になる。この節では、この点について説明を進める。

2.4.1 内部編成上の諸要因

冒頭で述べた通り、ＤＰ組織を編成するに当たっては、その内・外にわたる諸条件を十分頭に入れておく必要がある。ここでは、ＤＰ組織の内部編成を行なうに際して、特に留意すべき事項について、１、２補足しておきたい。

第一は、内部編成に際しては、ＤＰ組織が扱おうとしているシステムがどのような種類のものであるかを考慮しなければならない。即ち、システムの規模がどうか、組織全体にまたがるシステムか、特定部門のためのシステムか、また、デシジョン・メーキング的レベルのシステムか、オペレーショナルなレベルのシステムか、更に、現在、システム・ライフ・サイクル上のどの局面にあるのか、等々の事が検討されねばならない。

第二に、組織編成に当たっては、必ず要員（ヒト）の問題と併せて検討がなされる必要がある。組織と人間とは相互に依存し合う関係にある。従って、組織を現実に支える人間をどのように調達し、育成し、ローテーシ

ョンさせるかという事に十分心を配った組織が望ましい訳である。

2.4.2　機能別編成

ＤＰ組織の内部編成の一つの方法に、機能に着目して、同種機能別に仕事をつまり組織を編成するやり方がある。

具体的には、「2.2」節で述べたＤＰ組織の機能に従って、総合企画・調整部門、システムの設計・開発部門、システムの運用・オペレーション部門という形に、必要に応じては、それぞれをさらに細分化して、編成する形態である。つまり、ここでは、経理システムの開発と生産管理システムの開発とは一つのグループに纏められ、同様に、経理システムのオペレーションと生産管理システムのオペレーションとは、オペレーション・グループが併せて処理するという形になる。

この形態は、それぞれの機能の専門性が存分に発揮され、その意味で仕事の効率も高められよう。従って比較的大規模なＤＰ組織においては、その効果が活かされることとなろう。しかし、小規模な組織では、このような機能別専門分化はむしろ要員上のロスを招くのが通常であろう。なお、この形態をとる場合には、適宜要員のローテーションを行ない、企画から開発、さらにはオペレーションに至る一連のＤＰ関連機能を幅広く習得しうるよう、換言すれば、所謂「専門バカ」を防ぐよう留意しなくてはならない。

2.4.3　対象システム別編成

これに対して、別の方法として、そのＤＰ組織が扱おうとしているシステム毎に組織を編成するやり方がある。その場合は、例えば、経理システムに関する企画・開発からオペレーションに至る業務を経理グループとして編成し、同様に、生産管理グループその他を編成することになる。

この形態には、それぞれのグループが、対象とするユーザー部門と緊密な連携をとり、その実態に即したアプリケーション・システムが開発・運用されるという長所がある。しかし、各部門にまたがるような大規模なシステムの開発には不適当であり、また仮りにコンピュータ・オペレーションについても各対象システム別グループが行なう場合には、コンピュータ使用効率上の問題も生じよう。一般的には、この形態はＥＤＰ化の初期の段階に、あるいは比較的小規模なＤＰ組織に対応するものと考えられる。

機能別編成といい、対象システム別編成といっても、これらはあくまで物の考え方であって、実際にＤＰ組織を編成する場合には、両者が適宜組合わせられることも多いであろう。

2.4.4　システム設計、プログラミングおよびコーディング

ＤＰ組織を編成する際問題となる点として、システムの設計、プログラミング、そしてコーディングという各機能をどう編成するかという問題がある。これについて、若干述べておくこととする。

この問題に対する一つの解答は、これら三つの機能はそれぞれ独自な性格をもつ仕事であり、従って一人の人間にこれを併せ持たせることは望ましくないという考え方である。確かに、ユーザーからニーズを受けとめてシステムの設計をする仕事と、与えられた仕様に従ってコーディングする仕事との間には、無視しえぬ違いがある。しかし、仮りに非常に小さなシステムにおいて、設計、プログラミング、コーディングをそれぞれ別の人間が処理する場合を考えれば直ちに解る通り、具体的な条件如何では、この方法は要員的にも時間的にも極めてロスの多いやり方となる。

従って、この問題に対して一般的に妥当する解答はないと考えるべきであって、システムの規模・性格、要員事情その他を考えながら、総合的に

判断する他ない。

2.4.5　専門的諸機能の配置

ＤＰ組織の骨格は、ほぼ上述した通りであるが、ここで幾つかの忘れてはならない専門的あるいは周辺的諸機能がある。

これらを列挙すると、第一は調査・研究機能であって、コンピューターのハードウェア、ソフトウェアの調査・研究、諸技術・技法の向上・標準化といった仕事がこれに含まれよう。第２に、プログラムあるいはそのファイル等を管理するライブラリー機能がある。第三に、データ・ベースの企画・調整を行なうデータ・ベース・アドミニストレート機能も、次第に重要となりつつある。第４にコンピュータに関するマン・パワーの開発を担当する教育機能も重要な役割を負っている。

これらの各機能は、ＤＰシステムの開発が進み、ＤＰ組織が大きくなるにつれて、その重要性を増し、次第に独立した機能として姿をあらわしてくる。これらについても、ＤＰ組織を編成するに当たっては、独立した内部組織として編成するか否かに拘らず、十分に配慮することが望ましい。

2.5　ＤＰ組織とユーザー部門

2.5.1　システム・ライフ・サイクルにおけるＤＰ組織とユーザー

いかなるシステムも、何らかのユーザーのニーズがあり、これに応えるために開発され運用される。そして、システムの企画から実施・運用に至る各局面で、ＤＰ組織はユーザー部門との関わりを持つ。その関係は、システム・ライフ・サイクルの各局面に応じて様々に変化する。しかし、その中心的な関係は、システムの設計・開発における関係である。この節では、この点からＤＰ組織とユーザー部門との関係について述べることとする。

2.5.2　受託（請負）タイプ

ユーザー部門に対するＤＰ組織のあり方の中で、いわば最も素朴な形態として、受託ないし請負的なタイプがある。そこでは、システムを検討しその仕様を定める主体はユーザー側にあり、ＤＰ組織はそのユーザーからの委託ないし注文に応じる形で、プログラミング・コーディング、そして完成したシステムのオペレーションを処理するという形態をとる。

これは、ＤＰ組織の機能として先に挙げたもののうち、第一の総合企画・調整機能および第二のシステム設計・開発機能の主要な部分をユーザー自らが行なう形態であると考えられる。この様な場合は、ＤＰ組織はユーザー部門内に位置づけた方が適当であろう。ＥＤＰ化の初期の段階では、通常この形態が多いと考えられる。

2.5.3　共同タイプ

第二の形態は、ＤＰ組織が、システム設計・開発機能、更には総合計画・調整機能についても相当程度受け持ち、ユーザー部門と一体となってシステムの開発に当たる形態である。その典型的な例は、ＤＰ組織とユーザー部門とのそれぞれの関係者を選んでプロジェクト・チームを結成するケースである。

歴史的には、ＤＰ組織が様々な経験を経て、次第にシステムに対するアプローチの蓄積を深めてくるに伴い、先の受託タイプからこの共同タイプに移行するという場合が通常であろう。この形態では、一般にユーザー部門も幾つかにまたがることが多く、従ってＤＰ組織の位置づけの面からは、先述の管理部門へ編入する形態がこれに対応することとなろう。

2.5.4　コンサルタント・タイプ

更に高度になると、ＤＰ組織は、先に述べた諸機能を全て包摂した。そ

の意味で典型的なＤＰ組織となり、ユーザー部門のニーズ等を把握しつつ、自ら総合的なプランを樹て、現状の問題点を解析し、ユーザーに積極的に改善提案等を行なうまでに成熟しよう。このような形態を、コンサルタント・タイプと称することができる。勿論その場合でも、ユーザーはあくまでユーザーであり、従ってシステム開発の諸問題については、必ずそのユーザーの合意が獲得されねばならない。

またこの形態は、ＤＰ組織の位置づけの面からは、先述した管理部門編入もしくはトップ・マネジメント直轄の形態に、ほゞ対応するものである。

2.5.5　オープン・ショップとクローズド・ショップ

さて、ここでＤＰ組織とユーザー部門との関係について、やゝ視点を変え、所謂オープン・ショップとクローズド・ショップの問題について触れておこう。その場合、オープンといい、クローズドという事の意味内容を明らかにする必要があるが、ここでは、システム設計・開発機能あるいはコンピュータ・オペレーションの機能をＤＰ組織が自ら処理するか、それともユーザー部門の処理に委ねるかの意義であると理解しておくこととする。

そこで先ず、システム設計・開発機能についてであるが、これについてユーザー部門に分担せしめる所謂オープン・プログラマー制は、歴史的には、独りＤＰ組織にとどまらず、周辺のユーザー部門の中にもＤＰに関する知識と理解が深まった段階で出現するのが通常であろう。この方法は、アプリケーション分野に精通したユーザー自身がシステムの設計・開発に当たるため実務に密着したシステムがつくられること、またシステムあるいはプログラムのメンテナンスがユーザー部門で処理されることからＤＰ組織が開発的な業務に専念しやすいこと等の良さをもっている。しかし、

反面プログラミングの効率や標準化の面からは、むしろクローズド・ショップの方が優れているであろう。

他方、オペレーション機能についてみると、これをユーザー部門に委ねる所謂オープン制は、ユーザーが自らの必要に応じ必要な形でコンピュータを運用しうる良さを有してはいるが、コンピュータが大型であり、その効率的なスケジューリングが求められている場合には、むしろＤＰ組織が一元的にこれを操作するクローズド制が有効であろう。

これらについても、オープン・クローズドそれぞれの長所・短所をにらみ合わせながら決定することが必要である。

2.6　委員会組織およびプロジェクト・チーム

この章では、これまで、ＤＰに関する定常的組織について、様々の問題を論じてきた。そこで最後に、このようなＤＰ組織を前提としながら、多かれ少なかれ臨時的な性格を持つ組織体として、委員会組織およびプロジェクト・チームについて述べておこう。

2.6.1　委員会組織

ここで、プロジェクト・チームと区別して委員会組織というのは、トップ・マネジメントあるいはこれに近いレベルにおける管理・運営のための組織である。

このような委員会の一つの典型は、英語でSteering Committeeと称されるものであろう。これは、トップ・マネジメントの下にＤＰ組織やユーザー部門の代表が委員として集められ、長期的なシステム開発の基本方針や組織体全般にまたがるシステムの開発について、審議・運営してゆくという形態である。こうした形態は、各ユーザー間の利害を調整し、全体

を一つの方向へと導いてゆく方法として非常に効果的であろう。

また、そうした総合的恒常的な委員会ではなく、一定のシステムを開発するために、ＤＰ組織および関連ユーザーを委員とし、そのシステムのライフ・サイクルの間これを管理する臨時委員会組織も考えてよい。ユーザー部門に主体的な参加意識をもたせ、また効果的にユーザーの合意を調達するためにも、このような方法は有効である。

2.6.2　プロジェクト・チーム

ここでいうプロジェクト・チームは、前項の委員会組織に較べ一段下のレベル、いわばシステムに開発の第一線のレベルにおける共同参加組織である。即ち、システム開発のプロジェクトが定まると、これに関係するＤＰ組織およびユーザー部門のメンバーが選定されてチームが結成され、このチームが主体となってシステム・ライフ・サイクルの全過程を管理する形態である。このような方法も、ユーザー部門に責任意識をもたせ、またその同意をとりつけながら開発を進める等多くの長所を有している。

いずれにせよ、プロジェクト・チームは主に一時的・臨時的なテンポラリー組織であり、その結成や運営、さらに解散等に関しては定常的なＤＰ組織とはまた異なった配慮が求められるのは、いうまでもない。

3. ＥＤＰの教育・訓練

3.1 ＥＤＰ教育の目的

3.1.1 専門能力の向上

企業や組織内で開発される情報システムの規模が巨大化すれば、必然的にそれを開発するプロジェクトの規模も巨大化する。そうした大型プロジェクトは、多様な、異なる能力を持つ専門家たちの協力によって開発されなければならない。プロジェクトの成否は、全体をとりまとめていくプロジェクト・マネジメントの巧拙によることはもちろんであるが、プロジェクトの各フェイズ、各タスクを担当するメンバーが、そこで要求される専門能力をフルに発揮することにも大いに影響される。

ＥＤＰ教育の目的の１つは、分業化されたプロジェクト開発作業の専門職種ごとに、高い能力を与えることである。アメリカの企業においては、いちどある職種（たとえばプログラマー）に就いたら、本人が申し出て他の職種を担当できる能力のあることを示し、企業にそれを認めさせるという機会までは、いつまででもその同一職種についているという場合には、とくにこうした職種別の専門訓練が重要となる。ローテーションによって、１人の従業員がさまざまな職種を担当して経験を積み、次第に組織の高い地位へと昇進していく日本の企業においても、その職種に任命された担当者ができるだけ早くその職種の専門家にふさわしい腕をふるえるようにするためには、やはり専門訓練は重要であろう。

こうした、専門職種ごとの能力訓練を行なうためには、プロジェクトにおいて各担当者が果たすべき機能とそれを構成するタスクが明確に規定されていることが重要である。

システム開発の各フェイズ、各タスクの規定については、第Ⅱ部の全体、および第Ⅲ部第1章で触れられているが、それによって、そのタスクを担当するのにどんなスキルが、どんなレベルで必要かということも明確になってくる。教育・訓練はそうした規定に即して行なわれるものでなくてはならない。

3.1.2　能力開発と計画的配転

前項ではタスクの側からみた訓練のイメージであるが、独立した人格をもつ人間がわが国の企業に（多くの場合終身雇用制の下で）仕事を求めている場合、形式的な分業の理論に基づいて同一人を同一の仕事に定年までしばりつけておくわけにはいかない。ここに各個人の能力を伸ばし、豊富な経験を与え、昇進させていく計画的なローテーションが必要となる。教育訓練は、そうしたローテーションと表裏一体のものとして、個人の能力開発や態度形成を目的とする側面をもつ。

とうぜん、担当者の頻繁な交代はシステム開発作業の円滑な流れを阻害することもあるだろうし、担当者相互の意思疎通に問題が起こる可能性もあるかも知れない。システム・ライフ・サイクルにおけるスタンダードの役割りの一つには、そうしたコミュニケーションの確保があり、作られるシステムをなるべく属人化させない効果がある。

人事異動がある程度頻繁に行なわれても、転任者がなるべく早急に新しい職種において能力を発揮できるためには、個々の教育訓練の内容が効率的に編成されていなければいけないし、特定の個人のキャリアが実りあるものになるためには、教育訓練全体のカリキュラム体系がシスティマティックに構成されていなければならない。多くのわが国企業においては、ＥＤＰ部門と他の部門との人的交流もよく行なわれる。企業によっては現業

部門の経験者でなければＥＤＰ部門に配属しないとか、同一人がＥＤＰ部門に留まる期間をせいぜいＸ年に限るとかいう規定をもっているところもある。そうした形での配置転換が行なわれる場合には、ＥＤＰ教育のカリキュラムは、教育担当部門が持っている全社的視野での教育・訓練体系との関連性をとくに意識して、総合カリキュラム体系の一環として構成されなければならない。

3.2　ＥＤＰ教育・訓練の対象

3.2.1　対象の分類

ＥＤＰの教育・訓練はその対象によって次のように分類することができる。

1）一般部門

- ○トップ・マネジメント
- ○管理者層
- ○一般社員、従業員

2）特定ユーザー部門

- ○ユーザー・マネジメント
- ○ユーザー部門担当者
- ○ユーザー部門スタッフ
- ○ユーザー部門オペレータ

3）ＥＤＰ部門内

- ○ＥＤＰマネジメント
- ○プロジェクト・マネジャー
- ○プロジェクト・リーダー

○システムズ・アナリスト

○システムズ・エンジニア

○システムズ・デザイナー

○シニア・プログラマ

○プログラマ

○オペレータ

○ライブラリアン等

3.2.2　企業内他部門へのＥＤＰ教育

コンピュータを用いた情報処理システムが企業内で本当に効果を上げるためには、ポテンシャルなユーザーである社内の他部門のマネジャーたちにコンピュータの特徴や能力について適切な理解を持ってもらい、コンピュータを活用するにふさわしい場を提供してもらうことが重要である。

コンピュータ以前に我国に導入されているＱＣ（Quality Control）、ＩＥ（Industrial Engineering）、ＯＲ（Operations Research）等の管理技術の我が国企業における成否や定着性をみると、やはりそうした新しいものの紹介を組織の高いレベルから除々に下向きに進めていった企業のほうが、良い結果が得られているようである。

しかし、従来のマネジャー向けＥＤＰ教育のテキストを見ると、いささかハードウェアやロジック、アプリケーション技法などテクニカルな面への重点が置かれすぎているような気もする。比較的凝り性で理屈っぽいといわれる我が国の国民性からみて、たしかにそうしたテクニカルな細部に興味を覚えるマネジャーもいるが、かえって現業業務とシステムとの関連といった大局を見失っていたずらに枝葉末節に注文をつけ、システム開発作業をミスリードするという結果にもなりかねない。

いっぽう細部に走った教育内容は、そうしたマニア的興味を感じられないその他大部分のマネジャーたちに、"コンピュータとはわけのわからんものだ"といった印象を植付けるだけという結果も生じやすい。カリキュラムを作成する側の一方的な押しつけにならず、もっと深いマネジメントに対する理解の上に立った材料選択が望まれる。

一般の社員、とくにスタッフに対するＥＤＰ教育は、理論的・システマティックなアプローチへの手引きとして有効であることが多い。たとえばフローチャートによる因果関係のロジックの表現や、仕事に使われる情報の流れの表現は、日常業務の分析や改善にもよく使われているものであるが、ＥＤＰ教育がそのよいイントロダクションになることが多い。ハイアー・レベルのプログラミング言語（たとえばFORTRAN）の訓練も、たんにオープン・プログラマの層の拡大という見地だけから行なうのでなく、論理的な厳密なものの見方とその表現の訓練という見地で行なえば、効果はさらに高まるにちがいない。

3.2.3　システムズ・アナリストの教育

システム開発の主役ともいうべきシステムズ・アナリストに要求される知識・能力・資質はかなり程度が高く、広範囲にわたるものである。

Brandon Applied Systems Inc. のディック・ブランドン社長の講演によれば、"システムズ・アナリストに要求される知識を現在の大学教育でつめ込もうとすれば、専門課程に６～７年在学しなければならない、ということになる。同氏はまた、"現在米国におけるシステムズ・アナリストの７０％がプログラマ出身者である。私はこの比率は逆に３０％ぐらいになるべきで、他の７０％は他部門での実務経験の持ち主がよいと思う"とも言っている。

我国ではアナリストは１００％近くがプログラマ出身者で、システム開発の初期に行なわれる業務分析や予備設計の段階において、マネジメントとの適切な意思の交流を欠き、かれらのニーズや真の問題を理解せずにコンピュータ本位のシステムをつくり上げている例が多いようである。もちろん中には企業の要求に適切に応えることのできるすぐれたシステムも作られているが、それは体系的な教育・訓練の成果というよりはむしろアナリスト個人の持ち前の能力による場合が多く、そのアナリストが昇進等によって担当をはなれたときに、もはやシステムのグレートアップはおろかメインテナンスすら怪しくなるというケースすらある。また、そうした天分に恵まれたアナリストの中には、プロジェクト全体にとって重要な各種スタンダードの遵守を無視するという規律の面から望ましくない事態も起こることがある。

また、技術革新のはげしいコンピュータ・テクノロジーの第一線に身を置いて、つねに最初の知識に対処していくための、アナリストに対する再教育の問題も考えておかねばならない。

以上のような諸点を考えると、短年月でのアナリストの養成は絶望的であって、多少は規律を無視し、システムを属人的なものとする傾向はあっても、才能ある社員の自然発生という偶然を待望するほかないように見えるかも知れない。しかし米国においては、我が国でいう文科系―たとえばビジネス・スクール―出身者が、プログラマ出身のアナリストと対等に最新テクノロジーを駆使して成果を上げているのを見ると、体系的・効率的な教育・訓練は不可能ではないと思う。その際に、従来のわが国の専門技術訓練に欠けていたマネジメントの視点とビヘイビオラル（行動科学的）な角度とを考慮した、広い意味のマネジメント・サイエンスを中核としたカ

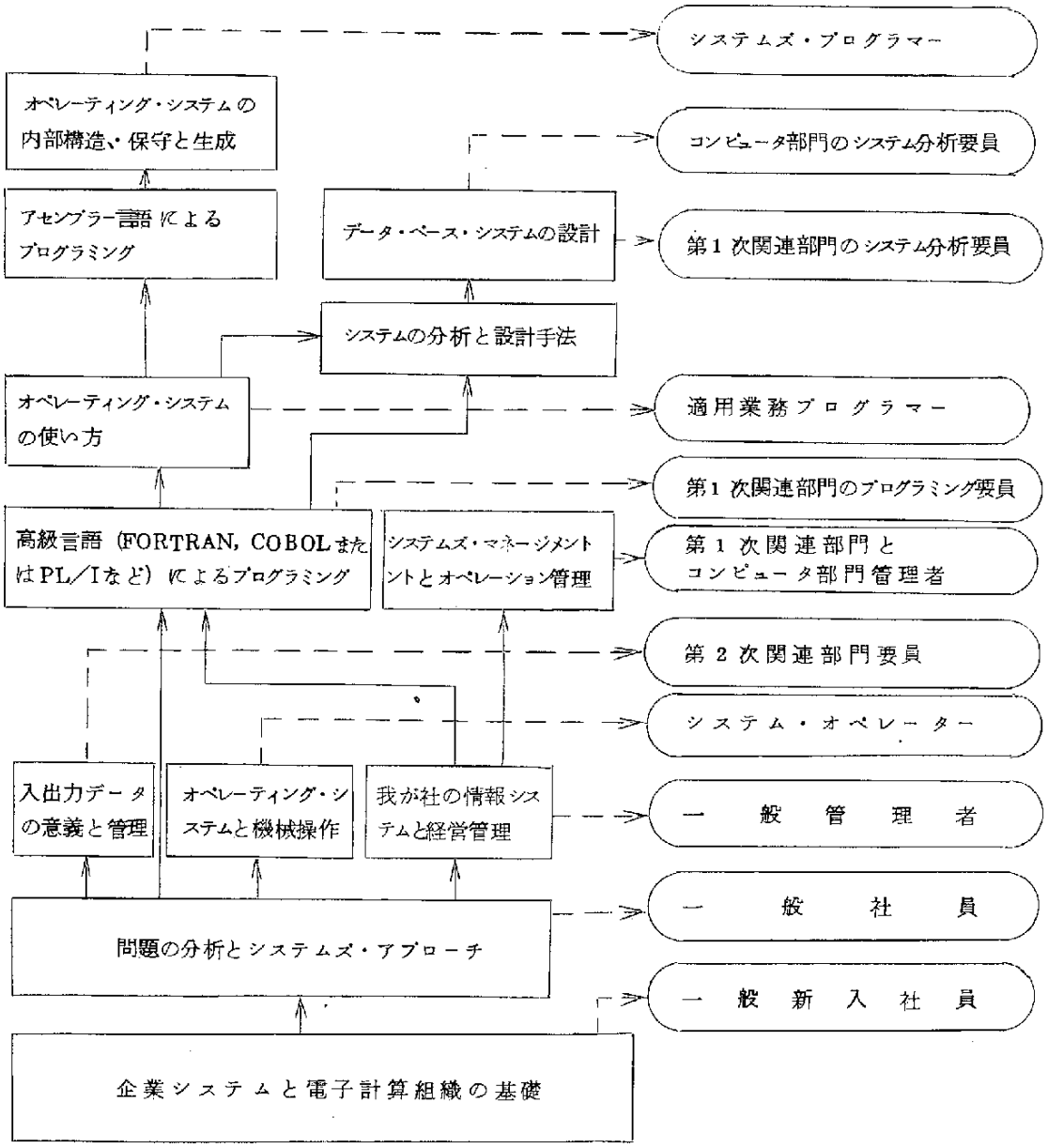
リキュラムを編成することが有効であろうと思われる。

3.2.4　ローテーションを前提としたカリキュラム体系

さきにも述べたように、我が国にはアメリカと違う事情があって、従業員を特定の職種に永久にしばりつけてはおけない。彼の勤続年数に応じた処遇として昇進を考えなければならないからである。しかし、価値感の多様化という情報化社会の一つの傾向からみて、逆に自分にあった仕事を永久に続け、昇進を望まない社員も出てくるかも知れない。管理能力がないのに、勤続年数（あるいは担当者としての貢献）だけを理由に管理者のイスを与えられることが本人にとって不幸であることも、起り得るかも知れない。

ＥＤＰ部門内で必要とされる幾つかの職種に要求される高度の専門能力を提供できるもので、しかも各個人のもつ昇進・配転経路（キャリア・パス）の多様性に対応できるものであるためには、ＥＤＰ教育体系のモジュール化、ビルディング・ブロック化が必要である。

次に示すのは日本ＩＢＭでカスタマー教育を担当されている江村潤郎氏の資料「企業内におけるコンピュータ要員の教育」に示されているものである。



図一　モジュラー教育体系の例

3.3　ＥＤＰ教育の方法

3.3.1　教育機関

(1)　社内教育

ＥＤＰ教育が役に立つものであるためには、マネジメント層やシステムズ・アナリストに対する教育・訓練が問題本位的（problem oriented）に行なわれることが肝要である。この意味からは、企業秘密の保持といった心配なく、生きた問題を通じて訓練のできる企業内教育は大きな意義がある。また参加者のレベルや興味の対象をある程度そろえられるので、教育訓練の効率も高まる。いわばオーダー・メイドのカリキュラムを持てるわけである。

しかし、そうしたカリキュラムの開発、コースの運営を社内で行なうことは多大の費用と労力を要し、資源の面からはかえって効率が悪くなる可能性もある。とくに講師も社内の専門家を充てている場合、カリキュラムの内容に偏りが出来たり、講師の社内での立場が影響したり、理論面での最新情報とのずれができたり、教授テクニックに問題があることも考えられる。

社内におけるオーダー・メイドの教育カリキュラムならびにコースの開発・整備は、教材・設備・専任者などを含めて巨額の投資になるので、十分な継続性と対象者の層の厚さとを考慮した上で決断しなければならない。安易な流行やムードで、中途半端な費用をかけるのは、いちばん損なやり方である。また、教育訓練の功を焦るあまり、コース開発に十分な時間と費用をかけず、即席で即効をねらうやり方も、つつしむべきであろう。

(2) 学校教育

プログラマやオペレータを養成する各種学校から、高度の専門家を養成する大学・大学院に至るまで、情報処理に関係ある学校教育機関は多い。

さきに教育対象の1つとして若干の考察をしたシステムズ・アナリストの供給源として大学教育を考えると、新規採用のルートは他にないとは言え、いくらかの問題点がある。

その1つは、システムズ・アナリストに要求される幅広い知識に対して、情報処理関係の学科の内容が数理的側面およびハードウェアの原理的側面に片寄っており、情報処理研究者の養成にはなっても実務担当者の供給源としては物足りないと思われる点である。また、そうした大学で訓練を受けた新入社員のほうでも、現実の企業の“どろくさい”面になじめないといった点がある。

また、コンピューターの設備という物的な面では、一部を除いてほとんどの大学は貧弱であり、企業の大型設備とのギャップが見られる。

入社後の社内教育との調整をはかって、両方でその長所を発揮するように考えるのが1つの解決策であるし、国内留学や研究生制度などによって企業と大学との人的交流をはかることも大切であろう。

(3) 専門団体

我国には情報処理研修センター、日本能率協会、日本経営協会、日本生産性本部、などセミナーを開催してＥＤＰ関係の教育・訓練を行なっている専門団体が数多く存在する。

上記(1)で触れた社内教育の欠点をカバーする意味では、こうした団体のセミナーは、ちょうどレディ・メードの多様なカリキュラムの中から

自社に適したものを選んで出せる場合にメリットが大きい。とくに他企業の事例・動向をも含んで新しい情報が得られるという点に期待が多く寄せられ、システムズ・アナリストの教育に向いていると考えられている。

こうした団体では、各社の特殊事情に応じて、イージー・オーダーに当たる社内教育コースの開発・運営の委託をも受けている。経済性と効果という両面から考えると、団体の側でもモジュラー的な形で用意したカリキュラムの構成要素を、その企業の特質・要求に応じてアレンジするという形で、ある程度企業側もリーダーシップを持って動くことが望ましい。

(4) メーカー

オペレータやプログラマなど、ハードウェア、ソフトウェアとの密着性の高い職種に対しては、その機種のメーカーが開催する講習会を利用する度合が多い。従来、ほとんどがメーカーのサービスとして無料で行なわれているから、経済性は抜群である。また、こうした講習会の中には、体系的な教育内容と、実物主体の完備した教育方法にもとづいて、マネジャーやアナリストに対する教育を行なうものも少なくはない。ただし、当然のことながら、ハードウェアやソフトウェアに対する展望には、重点の偏りが出るのはやむを得ないであろう。

3.3.2 教育方法

教育工学といった名称で、教育のために色々な機材を活用する方法が研究・試行されているが、そうした機材の中心として考えられているのはコンピュータである。将来コンピュータを駆使すべき立場にいるＥＤＰ要員の教育訓練にこそ、大いにコンピュータが活用されてしかるべきではなか

ろうか。

教育・訓練というのは、オペレータ・レベルでの能率向上が認められるといった場合を除いて、その成果を経済的に表現することが困難であり、教育内容や教育用器械の整備のための投資が押えられる傾向が強かった。

しかし、情報処理システムの開発と運用に当たっては今後ますます人間の重要性は増すものであり、その点から言っても、もっと費用を投下してもよいと思われる。

ずっと時間を経過してから表われ、しかも金銭的計測が困難な教育・訓練の効果を、ずっと早い時期に定量的に補捉できる代替指標の確立も、今後の課題であろう。

4. システム監査

4.1 システム監査の意義

4.1.1 システム監査の必要性

企業目的達成のための諸活動が、システムという概念をもった、より有機的総合的な機能としてとらえられつつある。

これに対応して、マネジメントも、伝統的な個別機能を持った組織別のマネジメントから、システム中心のマネジメントへと、その重点を移す必要がある。内部統制機能としての監査も同様である。

伝統的な会計中心の監査から脱皮して、システムを対象とする監査へと移行を計らねばならない。

コンピュータを含むシステムの監査は、その対象を大きく二つに分けることができる。

第一は、主要なシステムの、ライフ・サイクルの進展に対応して実施されるシステム監査である。

第二は、全システムの総合的な監査であり、システムの中心的な役割を果たすコンピュータの運営管理が、監査の中心となる。

システム・ライフ・サイクルの管理運営、コンピュータ・インストレーションの管理運営のいずれも、当然、担当のライン管理者により、長期的短期的なマネジメントが行なわれており、担当ライン監理者に責任と権限があることは当然である。

システム監査は、これらライン監理者の責任と権限を奪うものではなく、システムの成否が企業目的の遂行に及ぼす影響の大きさに鑑み、システムの現状把握と今後の方向に関して、直接トップ・マネジメントに対して、

その評価分析結果と改善案の勧告を行なうものである。

これによって、日常のマネジメント・プロセスから独立して、システムが真に企業全体の目的と方向に合致しているか、現在および将来におけるリスクを最小限にくい止めるにはどうすればよいかといった判断を行ない、解決策の実行に結びつけることが可能となる。

4.1.2　システム監査の背景

システム監査を計画し実施するに当たって、一般に現在のシステムをとりまく諸環境条件を考慮し十分な認識を持つことが必要である。

(1)　企業におけるシステムの中枢的役割

企業におけるシステムは、給与計算から出発して、だんだんと拡がりと深さを増し、今や企業の中枢的役割を果たす力を備えつつある。

日常的運営監理に止まらず、人員計画、組織、生産販売計画、投資計画、財務計画、市場計画など、経営戦略の策定に直接影響を及ぼす。

(2)　企業内外への影響の増大

システムの拡大は、企業内外に新しい問題を発生する。

企業内においては、システムと人間の調和をいかにして計り、社員のモラルの維持向上を達成するかという問題。

企業外への影響をいかにしてプラスの方向に保つか、いわゆる情報公害やプライバシーの侵害を防止する必要性も、ゆるがせにできない。

(3)　システム技術の高度化

システムの開発運営は、総合技術であり、コンピュータ関連技術を中心に、数学、計測、通信、機械、人間工学、会計学、経営学、心理学、など、関連する分野も多く、それを総合するシステム工学ともいうべき新しい諸手法が生れている。

(4) システム環境の急速な変化

システムをとりまく諸要因の変化が激しい。企業自体の目標や組織の変化はもちろん、企業環境、例えば市場構造、消費者の意識、経済情勢などの変化、価値観の多様化など、いずれも、システムの機能、効果に影響を与える。

すなわち、システムの評価は、時間と共に変わる。

システムは、変化に対応できる機能を持たなければならない。むしろ、それが、システムの本質といえるかも知れない。

(5) コンピュータおよびその周辺技術の発展

システムの中心となるコンピュータは、ハードウェア、ソフトウェア共に、急速な発展を遂げている。

システム・ライフ・サイクルの過程にコンピュータの進歩を極力とり入れることが必要である。

常に最新の装置、技術を利用することは、実際上不可能で、どこかで妥協することになろうが、将来の装置・技術の変化に対応できる融通性を備えていなければならない。

4.1.3 システム監査の目的と効果

システム監査の目的および効果を列挙する。

(1) システムが、企業の目標、経営の方向に合致するかどうかを評価し、確認する。

(2) 総合的、大局的見地から、トップ・マネジメントおよびＤＰ担当マネジメントに対して適確な助言と援助を行なう。

(3) システムの効率性をレビューする。

(4) 開発・運営のコスト、システムの導入により得られる利益を対比する。

（経済性）

(5) リスク・アセスメント

(6) プロジェクト管理体制の妥当性

(7) 技術的内容を分析し、改善案、代替案と対比する。

(8) システム導入による組織、人員計画への影響を評価する。

(9) 企業内各種規則・業務処理手順等への影響

(10) システムの障害発生の防止と、発生時の全社的な影響を最小限に止める方策を確立する。

(11) 機密保護に関する方策を確立し、企業および社内外の関係者の権利・プライバシーを護る。

(12) 以上の各項に関し、担当マネジメントから独立した監査チームによるレビューを行なうことにより、各関係組織の短期的な利害関係に基づくシステムの欠陥あるいは劣化をできるだけ防止して、システム・ライフをひき延ばす。

(13) また、社内外の各種分野の知識・経験をシステムの機能向上のために活用することができる。

4.1.4 システム監査の種類と、システム・ライフ・サイクルにおける位置づけ

システム・ライフ・サイクルは、その発生から消滅まで連続的な過程であり、しかも、一つの企業内において多数のシステムが共存する場合が多い。

このような状況のもとで、システム監査は、何を対象として、どの時点に行なうべきかを考える。

企業の規模、組織の構成システムの性質、その他いろいろな環境によっ

て条件は異なる。ここでは比較的一般的な考え方を述べたい。

システム監査の範疇を二つに大きく分ける。

一つは、個々のシステム（必ずしもすべてのシステムという訳ではなく、主要なシステムと考えてよい）をとり上げて、そのシステム・ライフ・サイクルの中で、いくつかのマイル・ストーンを設定し、そのシステムに関して監査を行なう。いわば縦割り監査である。

担当部門（ＤＰ部門および主な直接関連部門）におけるマネジメントが、適切に行なわれておれば、システム・ライフ・サイクルの各ステップ毎に監査する必要はない。むしろ有害である。

次の二つの時点が適当であろう。

まず、システム設計が完了し、システム詳細設計に入る時点である。

すなわち、システムの目的、全体仕様、開発計画が決定され、実際の開発に入る直前であり、開発投資のリスクを未然に防ぐ意味でのキーポイントである。これを「システム開発監査」と呼ぶことにする。

次いで、システムが完成し実稼動に入って初期の問題が解決され安定状態に入った時点である。システムの実稼動後、３カ月から６カ月位が大体の目安となろう。

この場合の監査は、システムの完成された時点における評価を行ない、企業の全業務体系、既存システムとの適合性を検証し、更には将来このシステムが他に及ぼす可能性のある悪影響を未然に防ぎ、システム・ライフの長期化を計るのが主目的である。これを「システム運営監査」と呼ぶことにする。

第二の範疇に属するシステム監査は、コンピュータ・インストレーション、すなわち、ＤＰ担当部門における業務全般の監査である。

いうまでもなく、コンピュータ（およびそれに付属する機器）は、あくまでシステムの一つの構成要素に過ぎない。

しかし同時に、コンピュータは、システムの最も重要な要素であり、中心をなす存在である。コストも高い。

コンピュータ、ならびにそれを直接管理するＤＰ担当部門が、いかに運営されているかは、その企業の全システムの効率に大きな関わりがある。

コンピュータのハードウェア、ソフトウェア、および関連する情報処理技術の発展に歩調を合わせ、システムの陳腐化を防ぐ意味からも、重要である。実施するサイクルとしては、１年から２年に１回位がよい。必ずしも定期的でなくてよい。これを「ＤＰ運営監査」と呼ぶ。

4.1.5　システム監査のための組織と担当者

既に述べたシステム監査の目的を達成するために、システム監査の実施は、ＤＰ担当部門から独立し、できるだけ高いレベルのマネジメント（できればトップ・マネジメント）に直属する組織あるいはチームによって行なわれるべきである。権限が不可欠であるからである。

同時に、システム監査チームには、権威が必要である。チーム全体として多くの経験、新しい知識、広い見識を持ち、システム分析能力と説得力を備えていることが望まれる。

チームの構成には、システムの性格や規模が十分考慮されなければならない。一般的に、コンピュータ・ハードウェア、ソフトウェア、会計管理、プロジェクト・コントロールの専門家が含まれている必要がある。もちろん、ＤＰ担当その他関係部門の代表が参画することも必要である。

企業内に、必要なだけの専門家がいない場合には、経験のあるコンサルタント会社、あるいはコンピュータ・メーカーに要員派遣を依頼するのも

有力な方法である。

システム監査のための専門組織を恒久的に置くことは、一般的にはまだ困難であろうが、コンピュータ導入の知識と経験を持った層が年々厚くなってきている現在、他の機能を兼務する組織、少なくとも管理者を置くことは、容易であろう。

4.2 システム監査の実施

システム監査の実施にあたっては、ＤＰ担当部門その他システム開発実施部門の業務を阻害することのないよう、十分な計画と準備に基づき、最小限の時間で最大の効果を挙げなければならない。

以下にシステム監査実施における手順を述べる。

4.2.1 監査計画の設定

監査担当管理者は、主要なシステムの進行状況について、適時情報を収集し、必要に応じて監査の計画をたてる。監査の種類としては、主として前に挙げた3つ、すなわち、システム開発の監査、システム運営の監査、およびコンピュータ運営の監査であるが、これ以外にも、例えば、システム設計の大幅な変更、企業経営環境の変化、コンピュータ機種の変更などに際しても、システム監査の必要度について判断し、実施する。

監査計画には次のような項目が含まれる。

1) 時期および期間

監査結果をシステムに反映させるタイミングを失しないよう、終了する目標日を必ず決める。

2) 対　　象

どのシステムのどの範囲を監査の対象とするか。

3）目　　的

諸般の情況をもとに、具体的に監査の目的を設定する。実施する規則になっているから実施するという漠然としたことでは成功しない。

4）レ　ベ　ル

対象および目的に応じて、どの程度詳細な監査を行なうかを決める。

マネジメントのレベルか、技術的内容まで立ち入るのか。

5）担　当　者

何人で行なうか。リーダーを誰にするか。

その他、必要な事項を含めて監査計画書を作り、必ずトップまたはそれに近いレベルのマネジメントの承認を得る必要がある。

4.2.2　要員の確保

マネジメントの承認が得られると、次に、監査担当要員を確保する。

対象、目的、レベルによって、確保すべき委員の数、専門分野の種類はさまざまであろう。

しかし、社内外から人を求める際に共通して考慮すべき素質として下記の点が挙げられる。

◎困難な課題にも積極的に取り組むファイト

◎一定の分野における専門的知識・経験を有するスペシャリスト

◎物事を客観的かつ公正に判断できること

社内の他部門から人を求める場合には、参加者が、所属部門の利害を離れて一定期間全社的見地から監査業務に没入できるよう、十分な配慮が必要である。社外、たとえば、コンサルタント会社に依頼する場合には、知識経験の範囲、程度、過去における実績、機密保護に関する条件、費用などについての配慮が必要である。

4.2.6　結果の確認

監査報告書がトップ・マネジメントによって承認されることによって、担当部門は、監査チームの勧告に従う義務を負う。

従って、改善結果について監査チームにも報告を行ない、監査チームはそれを確認する。

4.3　システム監査項目

システム監査を効率よく行なうために、監査の種類に応じて、検討項目をあらかじめ用意しておき、監査を行なう都度、その対象や目的に応じて項目を追加および取捨選択するのが便利である。可能ならば更に具体的なチェック・リストの形にするのがよい。

4.3.1　システム開発監査

システム開発監査は、システム全体設計が完了し、詳細設計に入る前に行なうもので、その主要目的は次にあげるとおりである。

◎システム設計が企業目的環境に沿って有効なものであるか。

◎システム開発・導入に伴うリスクが評価され、対策を講じられているか。

◎プロジェクトの態勢に問題はないか。

◎コンピュータの能力および準備状況は十分か。

＜監査項目の例＞

(1)　システムの目的が明確にされているか。

(2)　システムの機能が明確にされているか。

(3)　システムの性能（速度や最大処理能力など）に関する目標が決められているか。

(4) (2)、(3)について、関連部門の合意を得ているか。

(5) システム設計は、(2)、(3)を充たしているか。

(6) 代替案を考慮したか。

(7) 技術的に可能か。最善か。

(8) システム導入以降の環境の変化に耐え得る融通性を持たせているか。

(9) 現行システム、組織、業務手順に対する影響が明確にされ、関連部門と合意に達しているか。

(10) システムのメリットとディメリットが明確にされているか。

(11) システムの障害発生に対し、システム設計面から対策がとられているか。

(12) データの破壊、漏洩についての安全対策はたてられているか。

(13) 入出力データの正確性の確保についてどのような手段を講じる予定か。

(14) プロジェクトの組織は妥当か。

(15) 人員、能力は十分か。

(16) プロジェクト管理の態勢、手法は妥当か。

(17) 関係部門との情報交換、協力態勢は妥当か。

(18) プランに対して進捗度はどうか。

(19) 社内の諸標準規程に合致しているか。

(20) 予算の見積りと実績

(21)

(22) 使用を予定しているコンピュータ・システムの容量、能力は十分か。
（主記憶装置、外部記憶装置、通信回線、端末機器などを含む）

(23) 納期を確認したか。

㉔ 使用するプログラミング・システム、プログラム言語、プログラム・パッケージなどの選択は最適か。

㉕ プログラム開発計画、テスト計画は妥当か。

㉖ 移行計画を考慮しているか。

㉗ システムの各種障害についての予防措置、応急対策を、ハードウェア・ソフトウェアの両面から考慮しているか。

㉘ プログラムの文書化と保守に関する計画

㉙ （結論として）計画どおり開発に着手すべきか。

4. 3. 2　システム運営監査

システム運営監査は、システム開発が完了し、本稼動に入ったシステムについて行なうもので、その主要目的は次のとおり。

○システムが設計どおり完成したかどうかを確認する。

○今後の環境の変化に対する適応性、危険性を評価し、システム・ライフの長期化を計る。

○企業全体への影響を分析し、全社的に組織、人員、業務手順、などの適正化を計る。

＜監査項目の例＞

(1) システムの目的が計画どおり達成されたか。

(2) システムの機能が計画どおり実現されたか。

(3) システムの性能は計画どおりか。

(4) 関連部門の要望が十分組込まれているか。

(5) 年間を通しての最大負荷、および将来予想される負荷の増大にどこまで耐えられるか。

(6) その他考えられる環境の変化に対する融通性はどうか。

(7) 他システムとの重複、矛盾はないか。

(8) 各種システム障害に対する対策およびその運用手続が明確で、関係者に徹底されているか。

(9) データの破壊、漏洩についての対策は具体化されているか。

(10) 入出力データの正確性は確保されているか。

(11) プログラムの文書化と保守手順は確立されているか。

(12) 特に、例外処理手順が明確になっているか。

(13) ユーザーの満足度はどうか。システム設計変更の希望は。

4.3.3 ＤＰ運営監査

ＤＰ運営監査は、一企業におけるコンピュータ部門の運営管理状況を全体としてレビューし、その適正化を計るもので特定の時期は決められないが、一年または、二年に一度位が適当であろう。

主要目的は次のとおり。

○コンピュータ投資の効率化

○各システムが全体として経営の合理化に有機的に機能しているかどうかを評価し、必要ならば改善策を講じる。

○情報およびその処理能力の集中化の結果、ＤＰ部門における問題が全社的な損害をもたらす危険を防ぐ。

＜監査項目の例＞

(1) 組織とその職務は妥当か。

(2) 関連他部門との関係はどうか。

(3) 職種別の要員数、能力は十分か。不十分な場合、増強の計画は妥当か。モラルの問題も含む。

(4) 部内の教育計画、実績は妥当か。

(5) 導入済および導入予定のコンピュータ・システムの構成はどうか。

コンポーネント毎の稼動状況／稼動計画、ＣＰＵとＩ／Ｏのバランス

システムとしての稼動実績

(6) システムの能力は十分か。今後の業務拡張計画をも考慮する。

(7) マシン・スケジューリングの手順は、明確にたてられているか。またその管理・責任はどうか。

(8) 稼動実績の記録、およびその分析はどのように行なっているか。

(9) システム・オペレーションの手順は明確になっているか。

(10) 各種障害時の処理手順は明確か。過去の障害と結果の実績は？

(11) 入出力データの管理、チェックは十分に行なわれているか。

(12) 外部記憶媒体（テープ、ディスク等）の管理、保管状況はどうか。機密保護は？

(13) 使用しているプログラミング・システムの運用状況と問題点。

(14) プログラム開発手順、保守手順は正しく運用されているか。

(15) 新しいアプリケーション導入はどのように検討、計画、実施されているか。

(16) 現行各アプリケーションの状況はどうか。無駄はないか。重複はないか。改善計画をどうたてているか。

(17) システム分析やプログラミングの生産性をどう分析、把握しているか。

(18) 費用の記録、分析、配賦の方法と実績、予算との対比。

(19) アプリケーションの効果をどう分析しているか。初期の目標設定との差異はどうか。

Ⅳ　わが国における
システム・ライフ・サイクルの実態

Ⅳ　わが国におけるシステム・ライフ・サイクルの実態

1. アンケート調査

わが国主力企業におけるコンピュータ・システム開発作業の標準化に関する実態についてアンケート調査を実施してみた。われわれがもった問題意識はシステム開発の標準化における我が国特有の進行状況を把らえアメリカで発表されている文献で議論されている内容と差があるか否かを知るところにあったが、正確な実態をアンケート調査によってひきだすことについては疑問を多くもっていた。その理由として、アンケート調査に一般的につきまとう問題点とともに、システム開発作業の標準、特にS・L・C・の概念が広く共通なものとしていきわたっていないところからくるものがあったわけである。しかしなんらかの形で実態にさぐりを入れる必要を感じていたので、インタビュー調査と共に別途示すような形式によるアンケート調査を実施することにした。

調査対象は当協会会員企業257社のコンピュータ・システム開発担当部所を含め約500社をとりあげた。正確な実態をさぐるには調査対象母集団の吟味とコール・アップを必要とするが本調査では実行していない。したがってここで示す調査結果は単なる参考資料としての意味しかもっていないことを付記しておこう。

（1）　アンケート調査用紙

別　紙：　①、②、③　その3a

（2）　実施時期

昭和47年11月13日発送

同２５日〆切

（３）　発送数　５００通

返送数　１３６通

内　　訳

サービス関係

	社
金融（銀行、保険）	33
コンピュータ及び一般情報サービス	28
商社	5
運輸	9
その他	11
小　　計	86

製造関係

鉄鋼、造船	12
電力、ガス	7
機械	8
化学	13
その他	10
小　　計	50
合　　計	136

（４）　結　　果

①　標準化進行の有無について

	進められているか計画中	すすめていない
サービス	78	8
製　　造	48	2

標準化をすすめていない組織では理由として「必要性がない」をあげている。

② 標準化を現在進めているかあるいは計画中かについて

		進められている	計画中
サービス	金融	23	8
	コンピュータ・情報サービス	20	6
	商社	3	2
	運輸	5	3
	その他	4	3
	小計	56	22
製造	鉄鋼・造船	9	3
	電力・ガス	6	1
	機械	7	1
	化学	10	2
	その他	8	1
	小計	40	8
	合計	96	30

③ 標準化のレベルについて

ここではレベルをつぎの3つに区分している。

a. プログラミング段階以降のドキュメンテーションの標準化

b. Preliminary Analysis(Feasibility Study)からすべての段階を区分する概念としての標準。

c. ドキュメンテーションだけでなく、作業方式(Methodology)

自体の標準化

	進められている			計　画　中		
	a	b	c	a	b	c
金　融	10	7	6	2	3	3
コンピュータ・情報サービス	9	4	7	3	0	3
商　社	1	1	1	1	1	0
運　輸	0	5	0	2	1	0
その他	2	3	0	2	0	1
小　計	22	20	14	10	5	7
鉄鋼・造船	3	2	4	0	3	0
電力・ガス	3	2	1	0	1	0
機　械	2	5	0	0	1	0
化　学	5	4	1	0	0	2
その他	5	3	0	0	0	1
小　計	18	16	6	0	5	3
合　計	40	36	20	10	10	10

④　標準化の動機と背景について

（動機の項目）

a．自然発生的

b．社内（部門内）有志による研究

c．外部コンサルタント

d．海外企業視察

e．その他

（背景の項目）

a．企業規模の拡大

b．競争の激化

c．社外からの要請

d．スタッフの不足

e．その他

	動機					背景				
	a	b	c	d	e	a	b	c	d	e
金融	18	1	0	9	14	11	16	3	1	7
コンピューター情報・サービス	17	6	3	4	7	7	12	0	4	9
商社	3	0	0	1	1	3	1	0	0	1
運輸	6	0	0	3	2	3	4	0	1	1
その他	5	2	0	0	1	1	4	1	0	2
小計	49	9	3	17	25	25	34	4	6	20
鉄鋼	5	1	0	6	9	3	5	1	1	5
電力・ガス	1	0	0	1	6	2	2	0	0	3
機械	4	0	0	2	5	3	5	0	0	2
化学	7	2	0	2	8	6	3	0	1	4
その他	4	1	0	1	4	4	4	1	0	2
小計	21	4	0	12	32	18	19	2	2	16
合計	70	13	3	29	57	43	53	6	8	36

なお背景、動機について「その他」の項目に記入されているうちの主なものをつぎにあげておこう（同じ内容のものでも列挙してある）。

○　コミュニケーション確保のため

○　社内の利益管理、スケジュール、人事管理などを実施しやすくするため

○　未経験スタッフの早期戦力化

○　メインテナンスの容易性のため

○　技術部門からの要請

○　プログラム・メインテナンス

○　プログラムの生産性向上

○　電算機要員のローテーション

○　メインテナンス負担

○　メインテナンスの容易性

○　人事上の要請→要員のローテーション

○　システム開発部門の拡大

○　共同センター利用のため

○　メインテナンス段階とコミュニケーションの確保

○　適用業務の大規模化、複雑化（システムが相互に関連するようになる）

○　ソフトウェア企業の義務一会社の方針

○　要員動員力の拡大、ロスの減少

○　多人数によるシステム開発の能率化正確化をはかるため（管理者の指示による）

○　オペレーションの効率化

○　組織的な仕事の流し方と逆行するため

○　システムの品質管理・工程管理

○　運用、メイテナンスの合理化―要員の効率化

○　システム開発作業の大規模化、効率化

○　運用管理面におけるシステムプログラムメインテナンスの効率化をはかる

○　システムの維持・管理のため

○　組織的活動の必要性

○　組織分化の必要性

○　指導、監督の容易さ

○　システムの平準化（メインテナンス含む）

⑤　マニュアルの有無

	有	無	無回答
金　融	18	11	2
コンピュータ情報・サービス	18	6	2
商　社	4	1	0
運　輸	7	1	0
その他	3	5	0
小　計	50	24	4
鉄鋼・造船	9	2	1
電力・ガス	6	1	0
機　械	7	1	0
化　学	8	4	0
その他	6	3	0
小　計	36	11	1
合　計	86	35	5

⑥ 組織にあたえた影響と組織的承認のレベル

（組織にあたえた影響の項目）

a．影響なし

b．システム開発部門内体制変化

c．社内全体にわたる組織変更

d．その他

（承認のレベル）

a．社長レベル

b．副社長レベル

c．専務・常務レベル

d．役員レベル

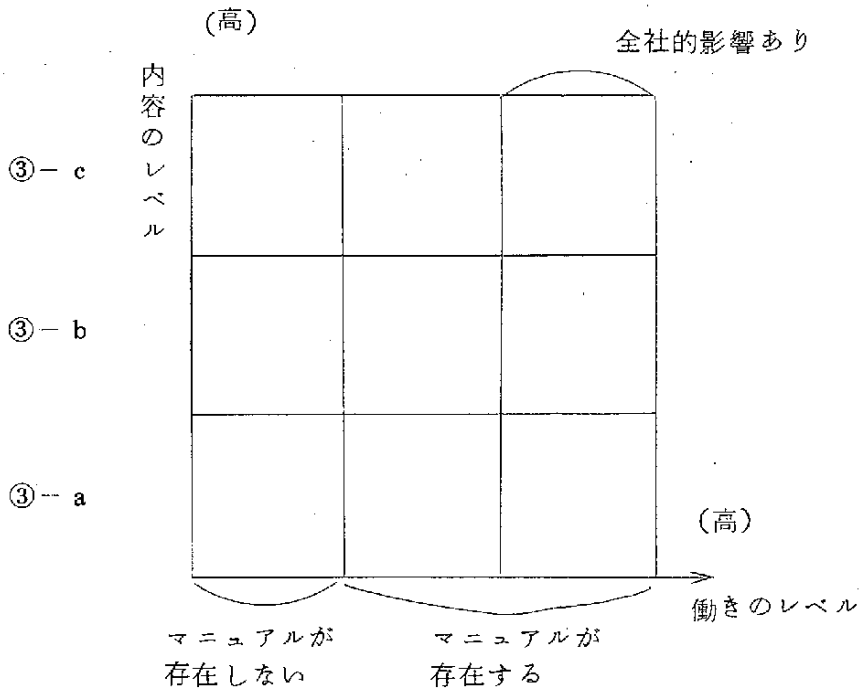
e．部長レベル

f．その他

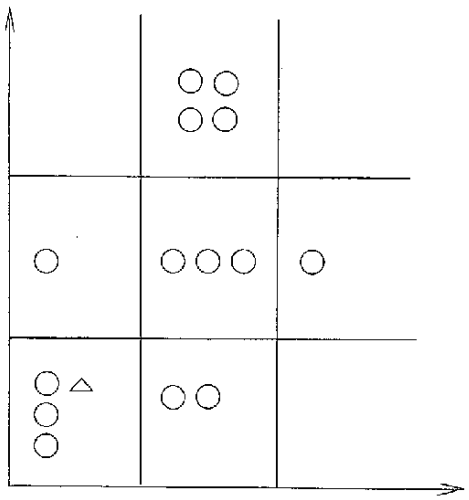
	組織にあたえた影響					承認レベル						
	a	b	c	d	無回答	a	b	c	d	e	f	無回答
金融	5	15	1	10	0	1	0	2	1	19	8	
コンピュータ情報・サービス	4	5	9	5	3	12	0	3	0	9	1	1
商社	1	4	0	0	0	0	0	0	0	5	0	0
運輸	3	5	0	0	0	0	0	1	1	5	1	0
その他	1	4	1	2	0	0	0	0	2	3	3	0
小計	14	33	11	17	3	13	0	6	4	41	13	1
鉄鋼	3	4	1	3	1	1	0	1	0	6	2	2
造船	2	4	0	1	0	0	1	0	0	2	4	0
電力・ガス	0	5	1	2	0	0	1	2	1	4	0	0
機械	3	8	0	0	1	0	0	0	2	8	2	0
化学	5	4	0	0	0	0	0	2	0	4	3	0
その他												
小計	13	25	2	6	2	1	2	5	3	24	11	2
合計	27	58	13	23	5	14	2	11	7	65	24	3

（5）　標準の存在の仕方について

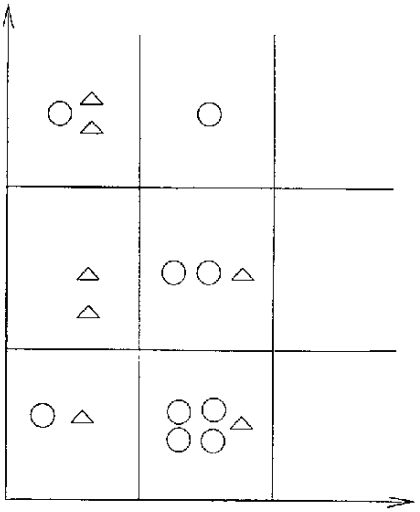
組織のなかでの標準を、内容のレベルと働きのレベルの２つの面からみてみよう。一般に内容のレベルが高くても、客観化されて存在しなければ働きのレベルは高くならないことが予想される。ただし組織の規模が小さいうちは１人のマネジャーの頭の中に、主観的に存在している標準概念は大きな働きをすることが考えられる。ここでは、アンケート調査のデータをもとにして、業種別に企業をこの２つのディメンションから位置づけてみることにする。下図でたて軸は前記③のレベルをとり、横軸は前記⑤、⑥を用いてランクづけしたものである。



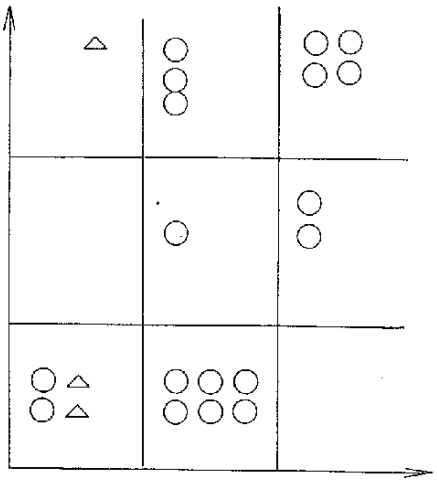
銀　　行



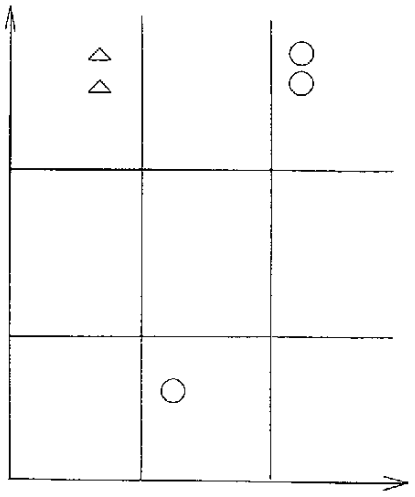
保　険、信　託



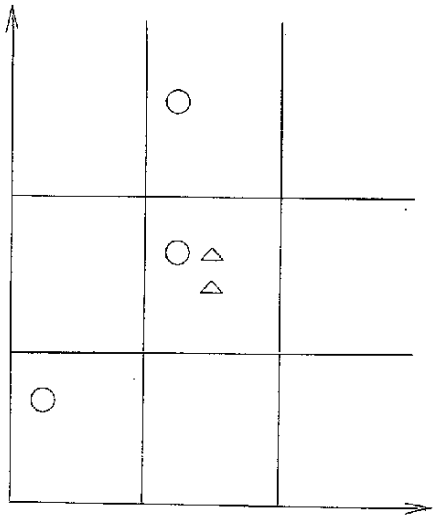
コンピュータ・サービス



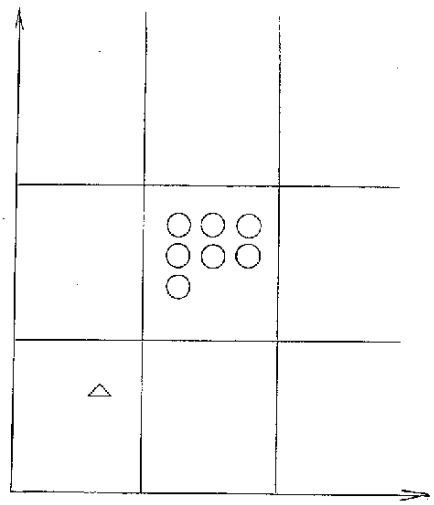
一般情報サービス



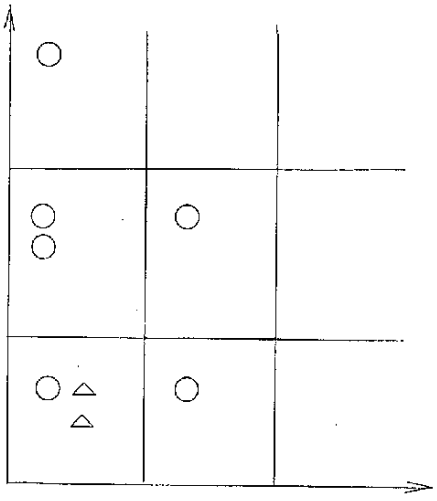
商　　社



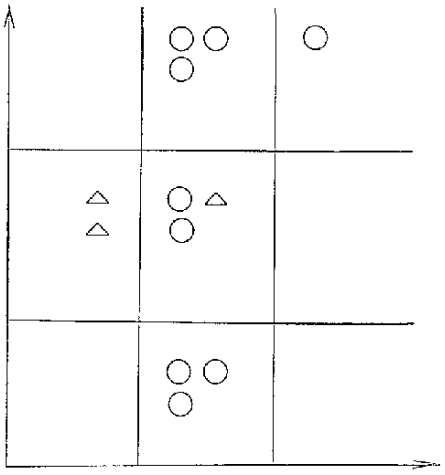
運　　輸



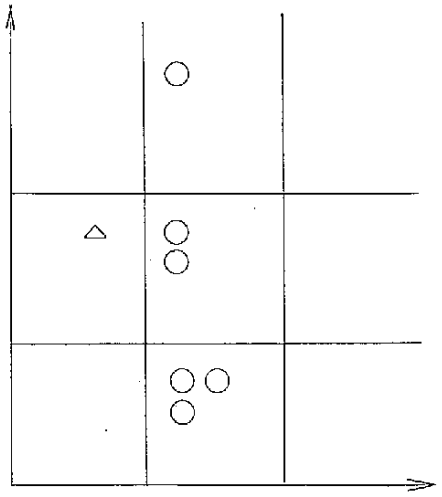
その他（サービス業）



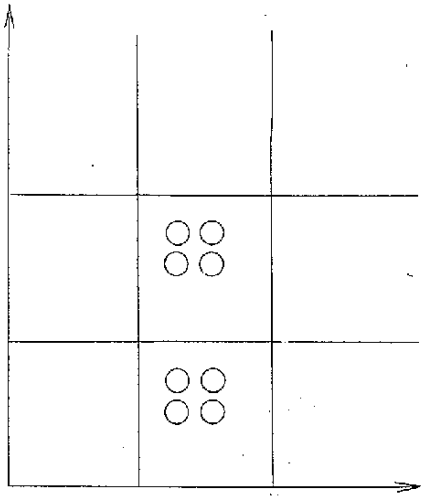
製　鉄　、　造　船



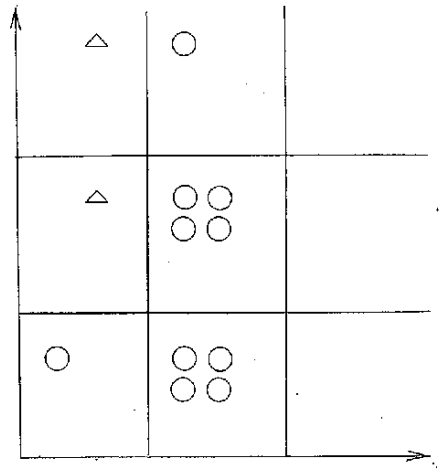
電　力　・　ガ　ス



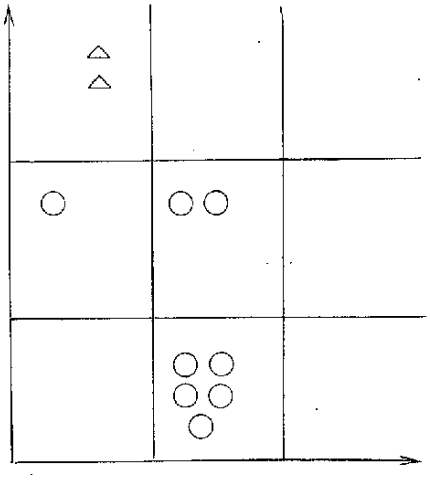
機　　械



化　　学



その他（製造業）



○　進行中

△　計画中

昭和４７年１１月９日

情報システム開発に関するアンケートのお願い

財団法人　日本経営情報開発協会

システム開発委員会

委員長　　松　田　武　彦

拝啓　時下ますますご清栄のこととおよろこび申し上げます。

さて、最近の技術進歩、とくにコンピュータの発達とその適用分野の拡大とによって、さまざまな組織で開発されるシステムが極度に大規模化、複雑化してきております。そうしたシステムの開発は、いかに能力ある人にとっても、もはや個人プレイで解決することは不可能であり、多数の、異なる専門を持つ人々の協力によって行なわれることが常識化して参りました。この、多数の協力によって行なわれるシステム開発を円滑化、効率化するためのマネジメント技法が、プロジェクト・マネジメント、プロジェクト・コントロール、システム・マネジメント・フェイズド・プロジェクト・プランニング、システム・アナリシス、システム・ライフ・サイクル・スタンダード等々各種の名称で研究されております。名称が多様であると共に、そのシステムの開発段階をあらわす用語もそれぞれ異なっておりますが、いずれもシステム開発という長期にわたる作業をいくつかのフェイズに分け、さらにその内部で行なわれるタスクやアクティビティを細分化して明示することによって、

１）マネジメントにプロジェクト全体の進行状況のチェックをやりやすくさせる、

２）必要なリソース（人員、設備、予備、時間）の配分を有効化する、

3）作成されるシステムの品質を保証できるよう、各部分作業の全体に対する意味づけと、そこでの注意点を指摘する、

4）各種のシステム・アナリシス、システム・デザイン技法の使いわけを的確なものとする、

5）標準形式に則ったドキュメンテーションの作成を強調することによって、開発過程相互間、及び将来のレビューおよびメインテナンス段階とのコミュニケーションを確保する。

などを期待している点に共通性があります。

フェイズのとらえ方については、例えばRubin 編著、Introduction to System Life Cycle(Handbook of Data Processing Management, 全6巻、Auerbach, 1966の第1巻）においては、1. Conception, 2. Preliminary Analysis, 3. System Design, 4. System Programming, 5. System Documentation, 6. System Installation, 7. System Operation, 8. System Cessation といった呼び方で、システムの生成・発展・死滅（更新）のサイクルを提示しております。またBenjamin著、Control of the Information System Development Cycle, Wiley 1970, においても、1. Feasibility Study, 2. System Specification, 3. System Engineering, 4. Programming and Procedure Development, 5. Implementation, 6. System Operation, といった6つのフェイズでとらえております。当協会では、これらの考え方を一つの目安とし、その上に我が国独自の状況を考慮して、我が国におけるシステム開発業務のガイドラインとなるべきものを作成したいと考え、本年4月からシステム開発委員会を設置し、研究を進めております。委員は主として、我国において代表的な大規模システム開発に当たってリーダ

ーとしての経験を持たれた方々で、そうした貴重な経験を基にしての研究・討議の経果を報告書として刊行すべく、作業を進めております。

その研究の一環として、我国の主力企業において、こうしたマネジメント技法がどの程度利用されているのかを、調査させて頂きたいと思います。調査はアンケート形式で行ない、結果を上記委員会に所属するワーキング・グループが集計して、わが国独自の問題点をさぐり出したいと考えております。

つきましては、ご多忙のところ恐縮とは存じますが、別紙のアンケートにご回答下さるようお願い申し上げます。

1. 何らかの形式で、システム開発作業の標準化が進められておりますでしょうか？
2. その標準化のレベルはどの程度でしょうか？
3. そうした標準化は、どんな背景・動機で始められたのでしょうか？
4. 標準化の具体的内容についてお教え頂けますでしょうか？マニュアル等ありましたら、頂けませんでしょうか？

   また、ワーキング・グループの訪問によるインタービューをお許し頂けるでしょうか。
5. そうした標準化が組織に及ぼした影響はどんなものでしょうか？
6. 標準化は、企業内のどのレベルまでの承認を得て行なわれたのでしょうか？

情報システム開発に関するアンケート

回　答　用　紙

（なお勝手を申して恐縮でございますが、１１月２５日までにご回答頂ければ幸甚に存じます。）

A	会　　名	
B	ご住所および Tel	
C	ご回答者所属部門名	
D	ご　回　答　者　名	

（当該項目に○印をお願いします）

1. a．進められている。　b．計画中である。　c．全然考えていない。

   cにマークされた場合は２～６をとばして７へ進んで下さい。

2. a．プログラミング段階以降のドキュメンテーションの標準化

   b．Preliminary Analysis（Feasibility）からすべての段階にわたる標準化

   c．ドキメンテーションだけでなく作業方式（Methodology）自体の標準化

3. 〔背　景〕

   a．企業規模の拡大

   b．競争の激化

   c．社外からの要請

   d．スタッフの不足

   e．その他（

〔動　機〕

a．自然発生的

b．社内（部門内）有志による研究

c．外部コンサルタント等の提案

d．海外企業視察

e．その他（　　）

4. 〔マニュアル〕

a．ある — 要求があれば協会に見本を進呈してよい

b．ある — 外部に配布するわけにはいかない

c．ない

〔インタービュ〕

a．応じてもよい　　b．応じられない

5. a．影響なし

b．システム開発部門内の体制変化

c．社内全体にわたる組織変更

d．その他（　　）

6. a．社長レベル

b．副社長レベル

c．専務・常務レベル

d．役員レベル

e．部長レベル

f．その他（　　）

7. （1－cにマークされた場合）

何らかの形式の標準化を考えてない理由をお聞かせ下さい。

a．その必要はない

b．担当者の能力開発の妨げになる

c．担当者のモラルに悪影響がある

d．標準化技法を知らなかった

e．その他（　　）

御回答ありがとうございました。

2. インタビュー調査

システム開発マネジメントのわが国における実状のインタビュー調査を実施した。

◇ 実施期間

昭和47年12月および同48年1月

◇ 調査企業

東京地区

A社 （銀行）

B社 （コンピュータ・ソフトウェア・サービス）

C社 （食品製造）

D社 （商社）

◇ インタビュー対象者

各社システム開発部門の部長を中心としてインタビューを依頼。

◇ インタビュー項目

システム開発プロジェクトのマネジメントにおける、分割した一連の開発フェーズの概念（標準）の用いられ方に焦点をあて、システム・マネジメントの実態についてインタビューをおこなった。その結果を以下に示す項目にそって各社の内容を整理した。

〔1〕 開発プロセスのフェーズ。

各社における開発作業の区切り方（フェーズ）と各フェーズで用いられているドキュメンテーション方式について。

〔2〕 システム部門の組織と要員配置のやり方。

〔3〕 システム・マネジメント面からみた各社の特徴。

A社　　日本開発銀行

〔1〕

事務管理班とユーザーとが一緒になり「システム開発予備調査表」を作成する段階からプロジェクトはスタートする。そして、開発班から企画運営班への「システム登録手続き」が終了する段階で終了となる。ただし、事務処理以外の業務（設備投資研究所の職員がだしたような統計処理などの業務）は作業登録手続き終了で区切られる。

これまでの経験からつぎのようなフェーズの分割概念をもっている。

		ロード配分
1.	現状分析	1／3
2.	基本設計	
3.	詳細設計	1／3
4.	プログラミング	
5.	ドキュメンテーション	1／3

◇　現状分析と基本設計はユーザーと協力しておこなう。その結果「基本設計書」ができる。この段階で、採用についての意思決定がおこなわれる。

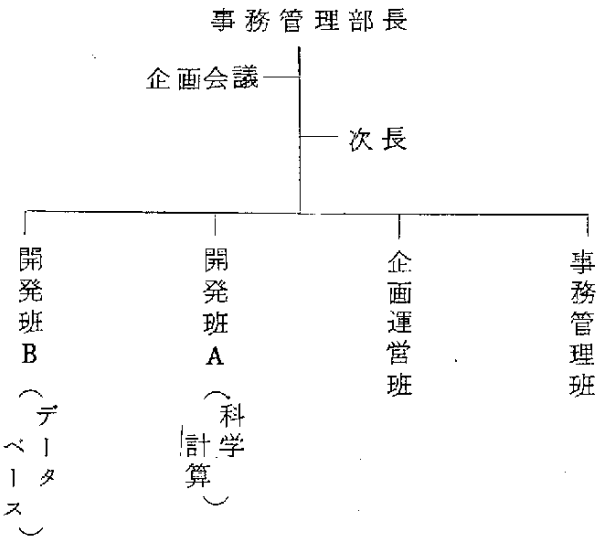
◇　詳細設計が完了すると、プログラミングの2／3は外注する関係もありここで1つのチエック・ポイントとなっている。

◇　上記のフェーズの区切り方は、作業間のコミュニケーションの機能の面が強い。

◇　各フェーズに投入される平均的ロード配分は上記の如くであるが、質的にみると、1、3が重要でそこでのマンパワーが問題である。

〔2〕

事務管理部の組織はつぎのようになっている。



◇　事務管理班 — ユーザーとコミュニケーションを保ちユーザーと共に「作業依頼表」を作成する。その結果を企画運営班に提出する。

◇　企画運営班 — 「作業依頼表」にもとづき新しいシステムを開発するか否かを決定する。決定されたら企画会議で了解のうえ要員の割り当てをおこなう。またシステム登録の受け付け、コンピュータの運営管理をおこなう。

◇　開発班A，B — 原則としてプロジェクトチームとなり、基本設計からプログラミングまでおこなう。

〔3〕

○　部長、次長、各班の班長とからなる企画会議が毎週1回おこなわれそこでスケジュール調整、リソース調整をおこなうことに形式上なっている。しかし実質的には各班長がマネジメントをおこない、会議は報告の形が多い。例外事項および各班間の調整が必要になった場合だけ次長、部長が調整をおこなう。また四半期ごとにプロジェクト別、要員別の調整をおこな

う。

◇　通常３～６本、ピーク時は１０本位（１～２万ステープ以上のプログラムをさす）程度のプロジェクトが進行している。

◇　プロジェクト開発前の評価は、数字にもとづいておこなうことはむずかしい。

◇　企画会議がプロジェクト・コントロールをおこなう体制である。

B社　　NBC

〔１〕

フェーズの区切りはアウト・プット（ドキュメント）の作成をとおして明確化されている。委託計算の場合を示すとつぎのようになっている。

部所…［営　業］　　［プログラム・グループ］

↘

（　見　積　り　）

↙

〔概要設計書〕

〔大日程計画書〕　↘

1.〔　システム仕様書　〕
↓
2.〔システム設計アブストラクト〕
↓
3.〔プログラム仕様書　〕
↓
4.〔オペレーション・スペック〕
↓
5.〔テスト・スペック　〕
↓
6.〔検　査　ス　ペ　ッ　ク　〕
↓
7.〔運　用　ス　ペ　ッ　ク　〕
↓
8.〔　保　　守　　書　　〕
↓
9.〔システム完成アブストラクト〕

・「テスト・スペック」の段階は内容の第1チェックでありプログラマーレベルでおこなう。

・「検査スペック」の段階はプログラムの最終テストにあたるもので、制作者以外の人間がおこなう。

・「運用スペック」ではデータうけわたし、チェック、検査要領が含まれる。

・二重投資をさけるため「システム設計アブストラクト」と「保守書」は社内に登録され配賦される。

各段階におけるドキュメントを作成するための規定、マニュアルとしてつぎのドキュメント作成要領がある。

① ハンドブック

② 工程管理基準

③ データ授受管理基準

（クライエントのデータ取扱いについて）

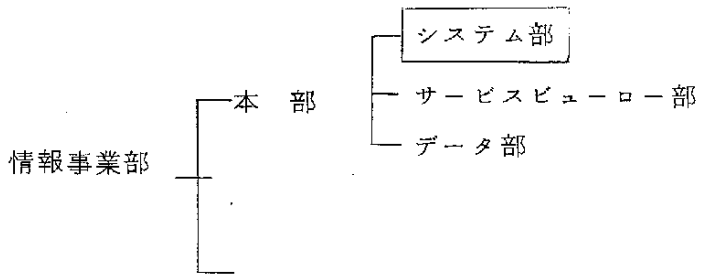
④ 運用管理基準 （前記4と7について）

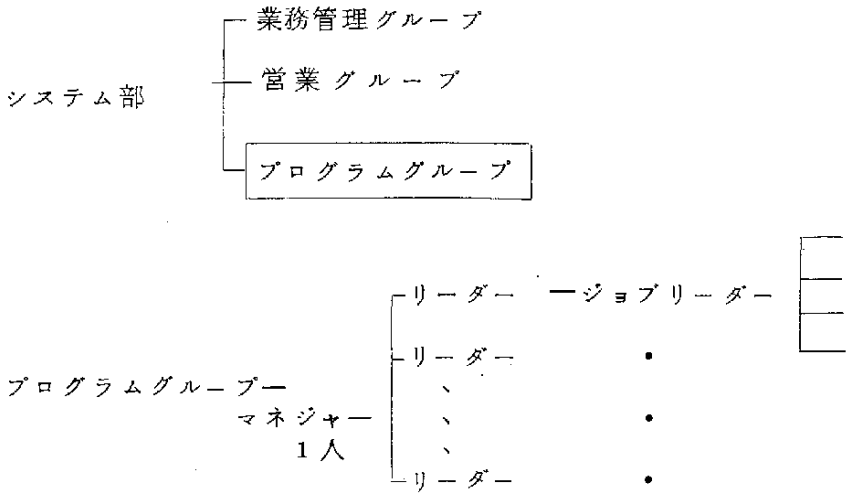
⑤ センター運用管理規定

〔2〕

プロジェクトを遂行するプログラム・グループが編成されている。

プログラム・マネジャー 1人に対して数ユニットが存在し、個々のユニットにリーダーがおりその下にジョブリーダーがいる。





〔３〕

システム・プロジェクト管理面からみた特徴。

○　当社は、システムが商品となっていることから作業の管理体制に神経を使っている。

○　作業プロセスの区切りがドキュメントによって明確化されている。

○　システム作成上の責任体制がある。たとえば工数見積にもとづいた想定原価責任（営業利益ではない）がある。

○　クライエントとの調整が制度化されている。

○　業務管理システムをもっている。

原　価

工　数

マシン工程

プロジェクト

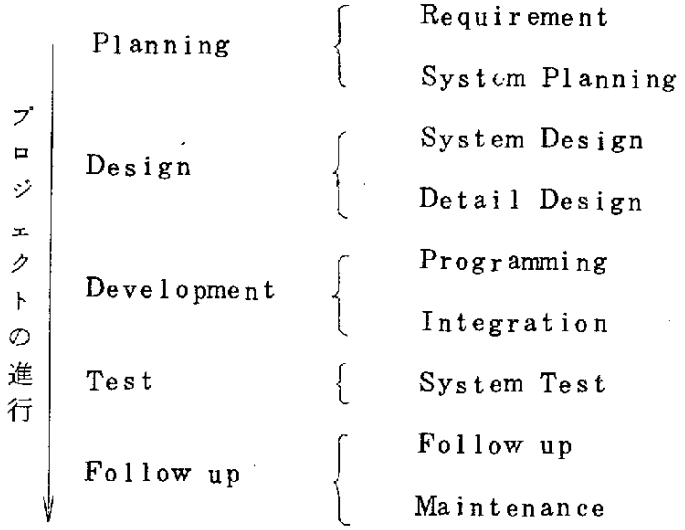
売上統計、業績統計など。

C社　　雪　印

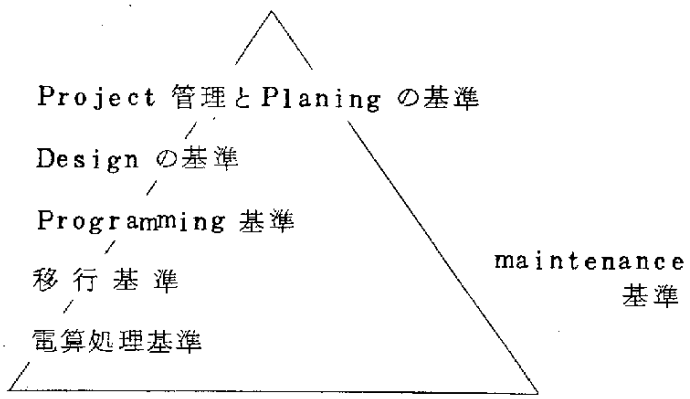
〔1〕

いまのところニーズの発生把握から完成に至るまでのプロセスを明確に区切ってマネジメントが実施されていない。現在当システム部々長の構造のもとにプロジェクト・マネジメントの標準化が検討され準備がすすめられている。以下に示すものは、そこで考えられている構想である。

◇　プロジェクト・マネジメントのフレーム・ワーク



◇　標準化の体系



◇　現在プロジェクト進行の区切りでドキュメント化されている主なものはつぎのものである。

○　基本構想

○　システム素案

○　日程計画表

○　インプット帳票デザイン

○　アウトプット帳票デザイン

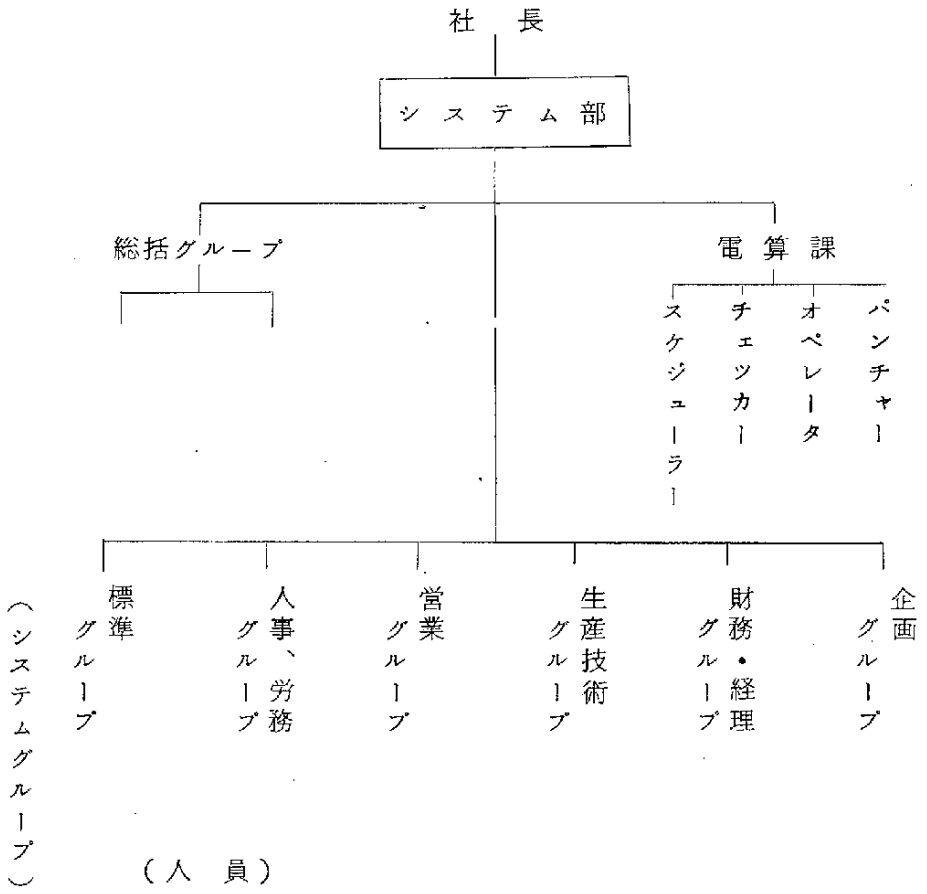
この他作業進行上作成あるいはメモされる関係資料をすべて一括ファイルしプロジェクト別に保存してある。これは、例えばシステム・メインテナンスに有用となる。

◇　47年9月に電算処理基準が作成された。作成目的はシステムの処理効率の向上である。ねらいは、システム構成に関する基準、プログラム及びサブプログラム基準を統一する点にある。

〔２〕

システム部の要員配置はつぎのようになっている。

（組　織）



（人　員）

	男	女
システム・グループ	５４名	６名
電　算　課	１１〃	７〃

（システム・グループの構成）

○　主査　1人

○　プランナー　5人～3人

○　システム・デザイナー

（システム・デザイン
　プログラミング

〔3〕

管理面の特徴

○　システム部長の権限がしめる比重が大きい組織である。

○　全般的ニーズの把握、システム化方針の決定、プロジェクト採用の判断はすべてシステム部長がおこなっている。

○　1つのシステム・グループで同時に複数個のプロジェクトを実行している。管理の単位はプロジェクト別よりも、システム・グループ別、人別の面が強い。

○　前記日程計画表（実績も記入）が個人別に記入され、システム部長はその内容を把握している。

○　ニーズは企業内部門から積極的に発生するよりも、システム部長の方針にもとづいてシステム部が先導する面が強い。

○　主査（グループの）は実行上の日程進行、要員配置のコントロールをおこなっている。また、プログラミング形成の採用は主査の判断による。

○　システム・デザインに必要な現場の知識はシステム部の要員のシステム開発上の経験にもとずいている。

○　システム開発による社内作業への影響力が強い。

D社　　三菱商事

〔1〕

ニーズの発生は、営業部門とのインフォーマル・コミュニケーションからくることが多い。ニーズ発生以後のシステム開発プロセスはつぎのようになっている。

1. インフォーマルな接触によるニーズの発生。
2. チーム要員（1人～2人）がユーザーとのインタビューなどによって調査をおこないチームリーダーに「業務報告」を提出する。この調査は、ユーザーから「依頼書」が提出されてはじまることが多い。またこの段階で進行が中止される場合が多い。
3. チーム・リーダーは勘によって要員算出をおこなう。自分のところで処理できない場合、部長調整がおこなわれる。
4. プロジェクト・チームが編成され、プロジェクト・リーダーは必要工数、期間の概要を決めて発足。この段階で「仕様書」がつくられる。
5. プログラミングの実施。
6. オペレーションは別会社がすべておこなう。

開発進行中、チーム・リーダーが要員算出する段階と仕様書がつくられる段階で費用見積がユーザーに渡される。

標準のドキュメント化は現在進行中である。その内容はEDPシステム部のすすむ方向づけをするためのものである。プログラミングに関係する使用記号など細部のドキュメンテーション標準は作成しない。立案、設計、プロダクション、パーフォーマンスを統括する標準をねらっている。

〔2〕

EDPシステム部長のもとにつぎの4つのチームが構成されている。

（イ）　統括チーム（男６人＋女８人）

（ロ）　ＭＳチーム（男１０人＋女２人）

経営計画関係システム開発

（ハ）　総合システムチーム（男２５人＋女２０人）

経理、総務部門中心のシステム開発

（ニ）　基本システムチーム（男４０人＋女２人）

オペレーションについては別会社を設けてそこにまかしている。

各チームにはチーム・リーダーがいて、その下にグループが編成される。各グループにはプロジェクト・リーダがいる。

現在の組織は昭和４７年１２月に新発足したもので、過去の組織の変遷がみられる。初期の組織は、社内各営業本部ごとに担当者が分かれていた。その後、チーム構成を中心にし社内部門別の分け方でない形態へうつった。この形態だと現場との遊離の問題が生じ、現在は両者の折ちゅう案といえる。

各チームには営業部門ごとの担当者２、３名を設けてそのシステム開発の窓口として働くことになっている。

〔３〕

○　プロジェクト・コントロールに関してはほとんどおこなわれていないといえる。コントロール資料は、グループごとに月１回「業務報告」として部長に提出される。

○　純粋の社内ニーズよりも、営業取引に関係して外部からの圧力からユーザニーズが生まれてくることが多い。

○　ユーザーとのコミュニケーションが重要な要素として認識されているが、問題が多々ある。システム設計上、営業取引、業務知識が必要となる。一方部内の要員は専属に新規採用している状況である。

○　プロジェクト・セレクションに関するファクターは単にマンパワーの面だけであり、コスト面の評価はおこなっていない。

○　開発費用のユーザー部門ごとへの分配方法を検討しているところである。複数部門にまたがる場合は非常に困難である。

ま　と　め

わが国主力企業におけるコンピューターを中心としたシステム開発は、この１０年の間に急速に進歩した。組織のなかにシステム開発部門が設置され要員確保と育成にかなりの力が投入されているのが通常の企業にみられる姿になっている。日本の企業にコンピューター・システムが導入されてから今日の状態をもたらすに至ったプロセスを考えてみよう。そこにはコンピューター・テクノロジーの加速度的進歩の結果を日本の企業が吸収、利用できる力をもっていたことが大きな要因として働いていたことに注目すべきである。日本の企業におけるシステム開発の拡大を支えていたのは技術的要因であるといえるわけである。組織のなかにコンピュータについての知識を持っている過去にみられなかった新しい人間グループが出現した。組織における彼らの行動の背後には、あるいは彼らが行使する力の背後にはコンピュータ・テクノロージーが存在しているわけである。ところが注目すべきことはこのような技術的要因が強く働いている段階からつぎの段階に日本の企業におけるシステム開発の進行が移行しつつあることであろう。コンピューター・テクノロジーの適用分野が拡大するにつれ、システム開発の成果を左右する要因は技術的なものからマネジメント的要因へ移行する。開発グループ内部での分業体制とそのコントロールの問題、開発グループと組織内他部門との間に発生する問題をいかに解決するかがシステム開発の成功を決める１つの大きな要因として認識されることになる。

このような観点から日本の企業におけるシステム開発の現状をみるとつぎのような特徴と問題点を指摘することができる。

○　システム開発グループの要員が専門化されている度合いは少ない。

○　システム開発をおこなっている人間がもっているシステム概念はプログラミング指向のものが多い。

○　システム開発プロセスの分割は実質的にはデザイン、プログラミング、テストの面でおこなわれている。

○　可能性研究（feasibility study）のフェーズ概念が存在しない。

○　開発グループの他にコンピュータ・テクノロジーについて詳しい人間がほとんどいない。

○　開発グループと他部門との関係のなかにいろいろな問題が潜在化する可能性が強い。

○　システムを組織内の第三者が監査できる体制にない。

○　ユーザー（現場）とのコンタクトはインフォーマルな人間関係を中心にとられる傾向がある。