Ⅱ　システム開発の手順

到達目標

基本的なシステム開発の手順を理解させ、各システム開発工程で何を行わなければならないのかを認識させる。また、調査・分析や基本計画などの上流工程が、それ以降の工程に対してより重要であることを理解させる。

さらに、システム開発が単なるプログラム開発ではないことを理解させる。すなわち、システム開発では、プログラミングの知識だけでなく、業務に関する知識や事務に関する知識などのより広範なシステム工学的知識が必要となることを認識させる。

1　システム開発の工程

図Ⅰ-3に示したシステム開発の工程を適用範囲を広げた後に詳しく示すと、図Ⅱ-1のようになる（一般に市販されている情報処理技術者試験対策用のテキストでは、調査・分析の項目が基本計画や外部設計の項目に包括されている場合が多いが、本書ではより広範に情報システムをとらえる目的から、システム開発の最初の段階に調査・分析を付加した。）。

システム自体が本来備えていなければならないと考えられる機能やユーザからの要求をもとに、開発対象となっているシステムの目的・目標を明確にするための調査・分析の工程がスタートとなり、その後に、基本計画、外部設計、内部設計、プログラム設計、プログラミング、テストという工程を経て、最終的に運用・保守という工程でシステム開発は終了する。一般に、運用や保守などといった表現を用いると、開発とは程遠い印象を受けるが、ここでいう運用・保守は、次のシステムへの要求を抽出するための最も優れた工程であるので、包括することとする。なお、内部設計までの工程については、本章で詳しく説明するが、プログラム設計以降の工程に関しては、本章でその概要を説明し、Ⅳ以降で内容を詳しく述べる。

また、各開発工程では、Ⅰで述べたドキュメントが作成されるが、このドキュメントのチェックのためにレビューと呼ばれる検証作業を行うこととなる。ソフトウェアは目に見えないものであり、唯一、その内容を共通の仕様で表したものがドキュメントであるので、レビューには、各開発工程のリーダーは必ず参加するように心がける。

（1）開発工程の考え方

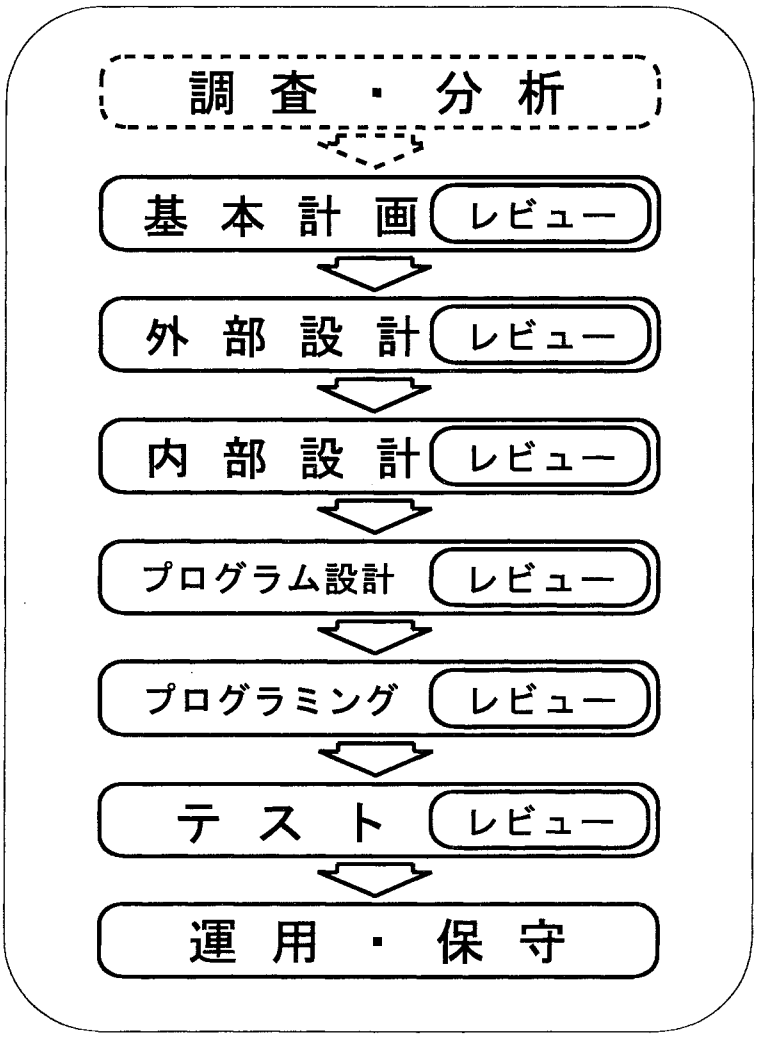
システム開発の工程は、調査・分析からプログラミングまでは、図Ⅱ-1に示したようにトップダウン的に順次行われていく。トップダウンとは、このように概要から徐々に詳細へと作業を進めていく方法であり、設計が段階的に次第に詳細となっていくことから、段階的

詳細化技法とよばれている。

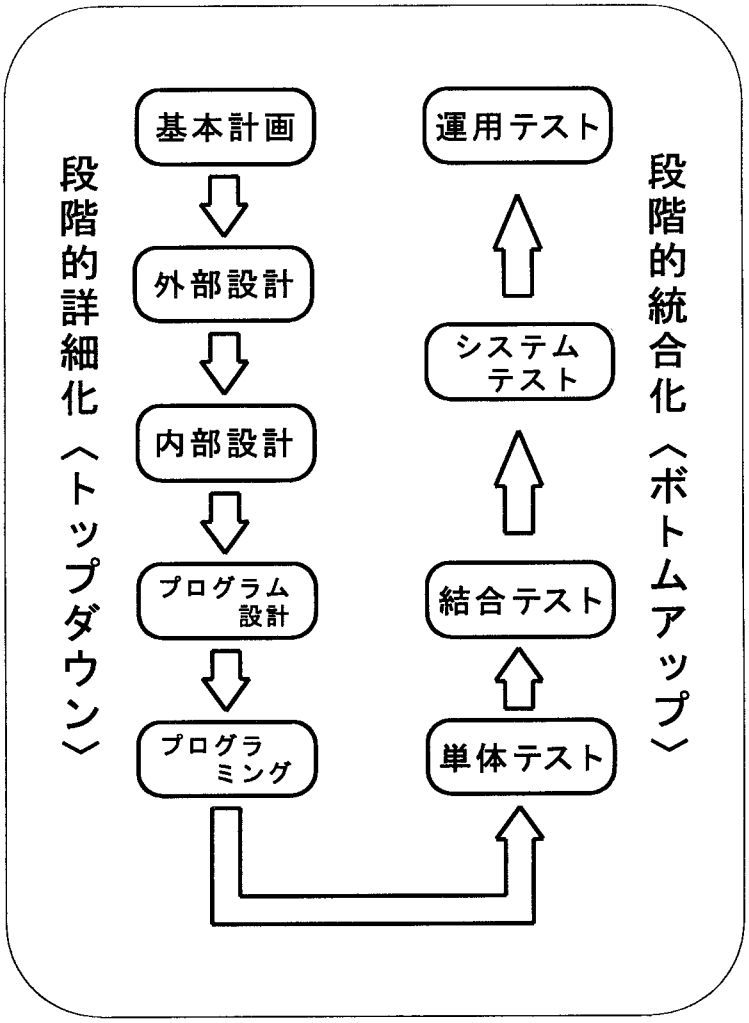
テスト工程においては、この逆で、ボトムアップ的に細部から徐々に全体へと作業を進めていく。この方法は、テストの内容やそれぞれのプログラムが、段階的に次第に全体へと統合していくことから段階的統合化技法とよばれている。

なお、これは単なる一例であり、全ての企業においてこの開発工程が採用されているという訳ではない。各企業やその企業内の各プロジェクトチームでは、その企業およびプロジェクトチーム独自のシステム開発法を用いている場合が多くあるが、その基本となるのが、ここで示す開発工程の考え方であるということに注意して頂きたい（システム開発工程の種類に関してはⅢ章で詳しく述べる。）。

図Ⅱ-１で示したシステム開発工程をトップダウン開発と、ボトムアップ開発に分けると図Ⅱ-２のようになる。



図Ⅱ-１　システム開発工程



図Ⅱ-2　トップダウン開発とボトムアップ開発

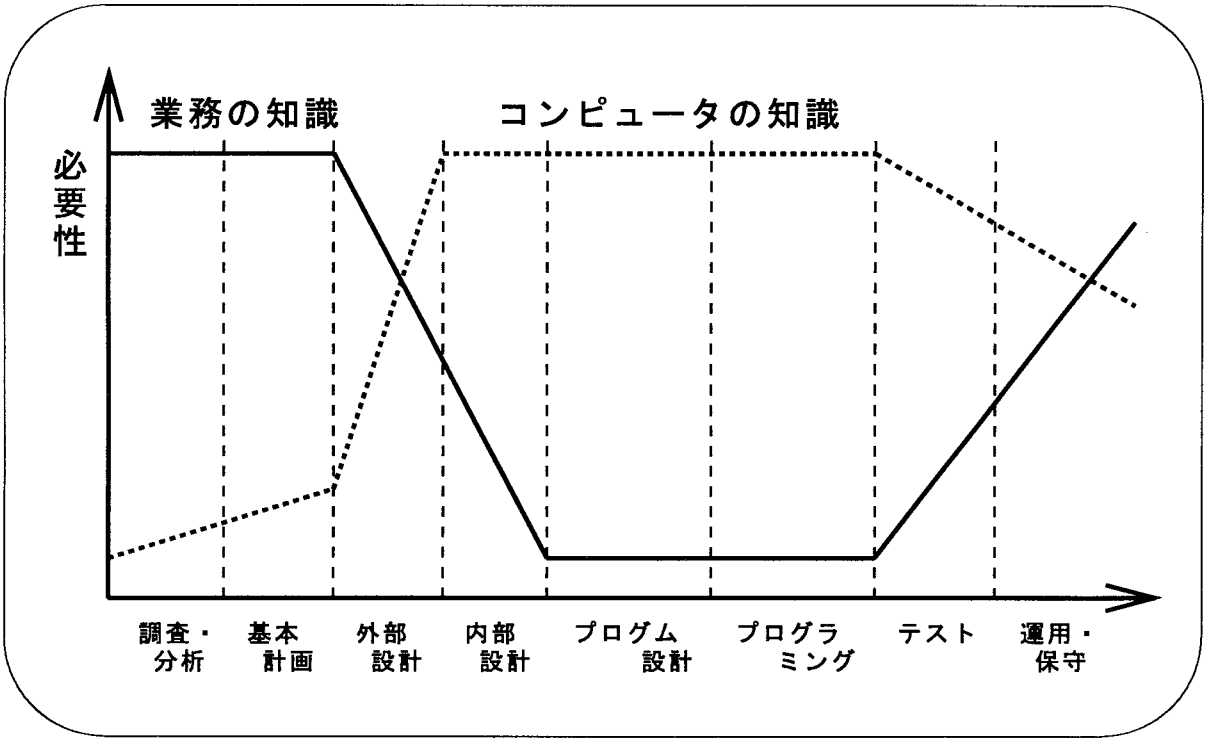
（2）レビューの必要性

レビューは、各開発工程の終了段階で行われるもので、作業の成果物であるドキュメントの見直しをするためのものである。

図Ⅱ-2に示したように、システム開発のほとんどの工程でレビューが行われるが、特に重要となるのが、上流工程である基本計画や設計の工程でのレビューである。計画ミスや設計ミスは、プログラムミスに比べて、はるかに大きな問題をシステム開発全体の作業に及ぼすからである。計画ミスや設計ミスの発見が遅れれば遅れるほど修復が困難となり、ときには後戻りすることもできない状況に陥ることがある。これは、そのまま費用の面、すなわち修正コストにひびいてくることとなる。ミスの程度にもよるが、テストの工程で発見されたミスの修正コストは、設計工程での数倍、また、運用工程で発見されたミスの修正コストは、設計工程での数十倍かかるといわれている。当然のことながら、計画ミスが下流工程で発見

されるようなことが生じた場合は、さらに大きな修正コストが必要となり、場合によっては、システムを開発する人員の再選択・再配置を行い、最初からやり直したほうが早いということも起こりうる。

レビューを実施するときは、上記のような欠陥を避けるという目的から、各開発工程のメンバーだけでなく、さまざまな立場の人員で構成されたメンバーで行わなければならない。図Ⅱ－３に示すように、各開発工程には、通常、その工程の専門家集団が配置されている（どのようなシステムを開発するかにより、その専門とする人員の必要度は変わってくるので参考程度として頂きたい。）。



図Ⅱ－３　業務とコンピュータに関する知識の必要性

大別すると、最初と最後の工程を担当する事務寄りの人員（業務専門）と、プログラム設計およびプログラミングを担当するプログラマ（コンピュータ専門）、そして、その引継の部分を担当する人員（システム開発に関するインタフェース専門）となっている。基本計画や運用の工程では、業務の専門家が多くを占めているので、レビュー実施時は、さらに多くの業務の専門家が出席するのではなく、コンピュータの専門家やインタフェースの専門家が多く出席することが理想的である。逆に、プログラミングに関する工程においても、プログラマばかりが出席するのではなく、業務の専門家やインタフェースの専門家が多く出席することが望ましい。このようにして、開発すべき新システムにおける各開発工程での考え方の

くい違いを吸収することができる。

そして、参加したメンバーは、それぞれの立場から計画・設計・実行結果を検討し、その結果を各開発工程の担当者にフィードバックする。これにより、担当者が気づかなかったミスをなくすことができる。

なお、レビューにおいて、各工程の担当者の間でくい違いが生じ、どうしても意見の一致がなされなかった場合は、レビューを行っている工程のリーダーや、より上流工程を担当しているリーダーの意見を尊重する。間違っても、賛成多数などといった手段をとって、以降の作業を進めてはならない。

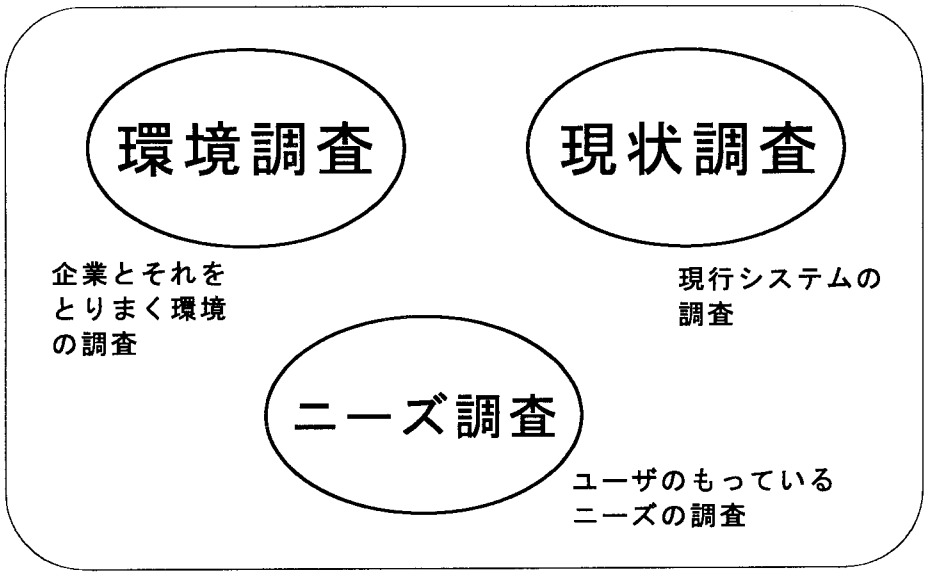
2　調査

システム開発の最初の工程となるのが調査である。現行のシステムでは業務に耐えることができない、もしくは耐えることはできてもそのための人件費が膨大にかかるなどの理由から、新しいシステムを開発する必要が生じてくる。したがって、その理由を明確にするための調査が必要となるが、充分な調査結果を得ることは開発期間や経費の面からみてかなり困難である。したがって、限られた期間・経費で効率よく調査を行うために、調査の目的、対象、範囲を前もって明確にしておくことが必要となる。

また、調査を効率よく行うための手段についても知っておかなければならない。ただ漠然と大量のデータを収集するのは時間の無駄である。体系立てた調査法をあらかじめ選定しておき、後の分析作業が効率よく的確に行えるようにしなければならない。

なお、調査対象となるのは、当然のことながら現行のシステムであるが、そのシステムが使用されていた背景を知るために、それが関連していた業務全体の調査を行うことも必要となる。したがって、単に調査といえども、企業全体をみた調査と現行システムの調査の二つを行わなければならないこととなり、また、ユーザ自身がもっているニーズもあるので、ニーズ調査も行わなければならない（図Ⅱ-4）。

さらに、システム開発を終了した時点でのタイムラグなども考慮に入れ、新システムが近未来の動向にもできる限り対応できるように、システムの将来動向についても調査しておく。開発するシステムが、果たして何年間業務に対応できるのかといった問題は、システムの規模が大きくなればなるほど重大となってくる。システムの入れ替えは、コンピュータなどの機械の入れ替えとは違い、簡単にできるものではない。新しいシステムが導入された場合、必ずそのシステムを有効利用するための導入研修が必要となる。３年間しか利用できないようなシステムに、導入研修を１年もかける訳にはいかない。これは、企業であっても、大学などの教育機関であっも同様である。システム開発者は、先に述べた調査以外に、社会動向もあわせて調査しなければならない。なお、あまり先読みをしたシステムは、その開発期間・経費が莫大になり、逆にシステム開発の意味がなくなってしまうので注意を要する。

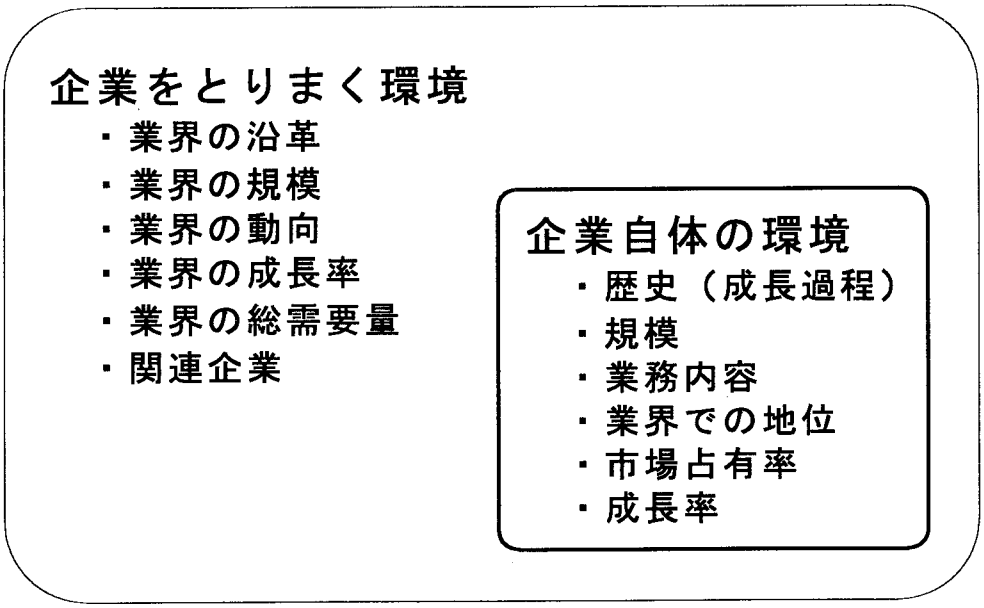


図Ⅱ-4　調査項目の大分類

（1）環境調査

対象とする企業に関する各種の知識を習得することが、その企業のシステムを理解する上で有用な手段となる。この知識習得のために行う調査を環境調査という。

環境調査は、現行システムの調査を行うのではなく、企業をとりまく環境や企業自体の調査を主に行う。すなわち、その企業の役割や属している業界の規模、動向などの「企業をとりまく環境」と、その企業の歴史的変貌と現在の構造などの「企業自体の環境」の二つを調査すればよい（図Ⅱ-5）。この調査は、一般にいう企業研究に属し、資料を利用した調査によって行われる。

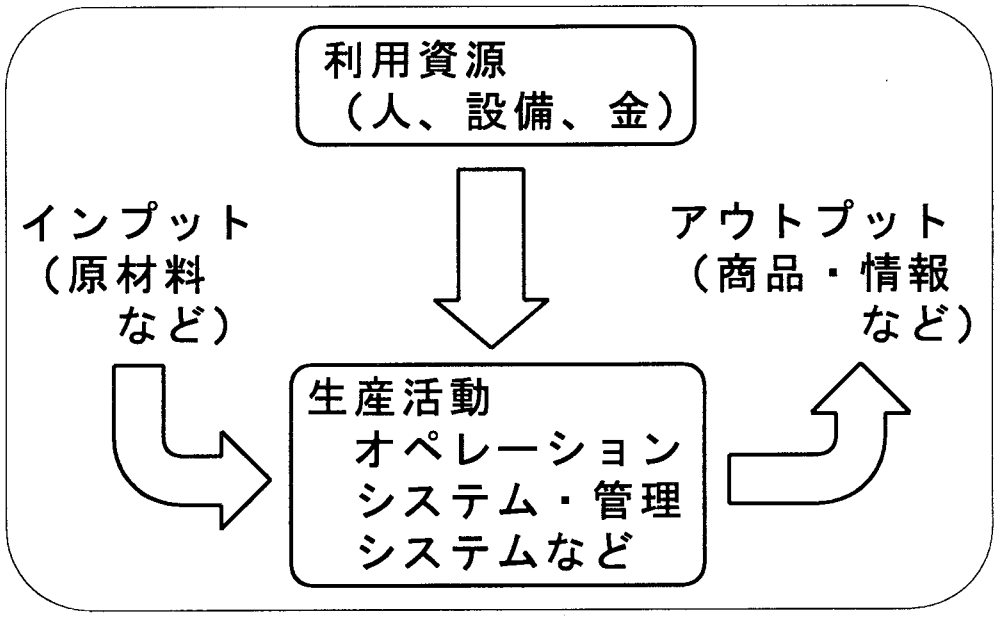


図Ⅱ-5　環境調査

（2）現状調査

環境調査が企業やその周辺の予備知識収集のためのものなら、現状調査は現行システムに関する知識収集のためのものといえる。すなわち、環境調査で企業の概観をつかんでおいてから、よりミクロな部分へと調査を移行していく訳である。

ここでの調査で重要となるのは、現行システムの入出力およびその中間のプロセスを明確に把握しておくことである。システムの入出力は、以降の開発工程における基本設計やファイル設計、コード設計に必須となるのもである。一つでも抜け落ちていたら、後の開発工程で大きな修正を迫られることになるので、細心の注意を要する。一例として、図Ⅱ-6に製造業における生産システムの入出力とプロセスを示す。



図Ⅱ-6　入出力とプロセス

図Ⅱ-6をもとにその調査項目を抽出すると、以下のようなものが挙げられる。

イ　システムの目標と目的

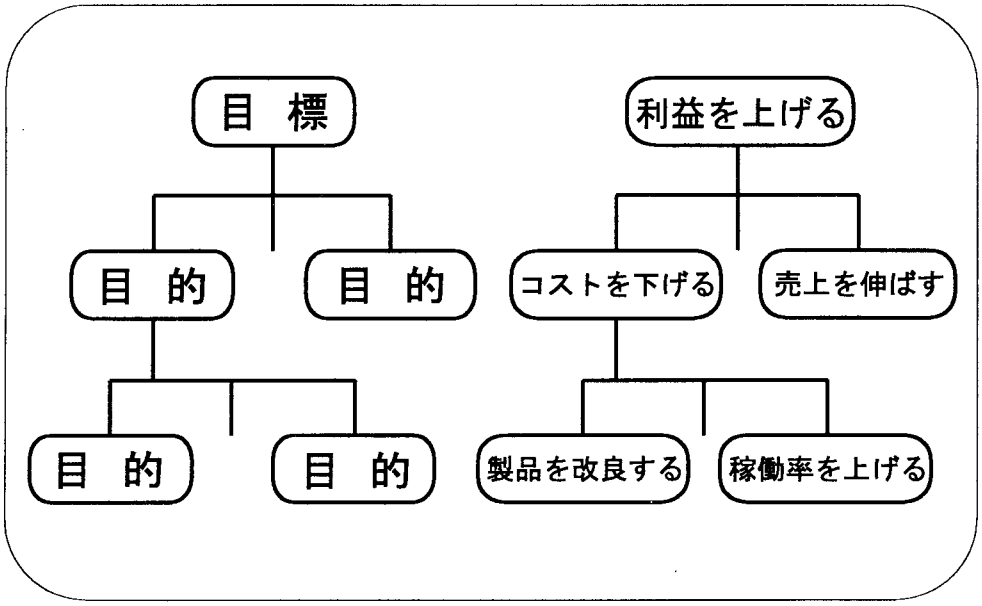
システムが求めている目標とそれを実現するための目的（図Ⅱ-7）を調査する。

ここでいう目標とは、企業が求めている最終到達点であり、目的とは、それを達成するために設けられたチェック・ポイントである。

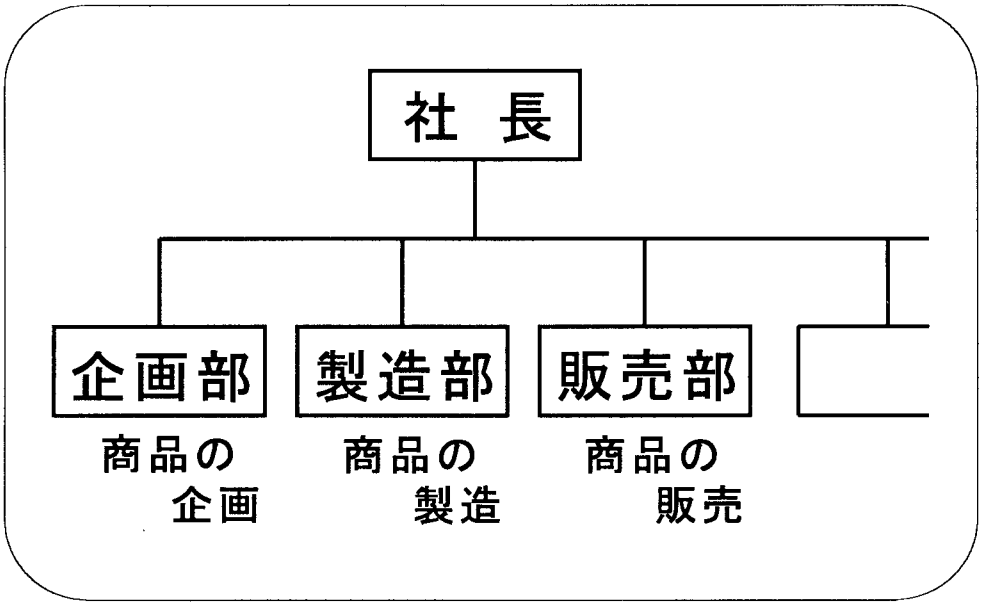
ロ　システムの組織形態

システムに関連している組織の形態（図Ⅱ-8）を調査する。

図Ⅱ-6に示した製造業を例にとってみると、製造業の生産システムだからといって、関係している部門は製造部だけではないということが分かる。製造する製品の原材料を入



図Ⅱ-7　目標と目的



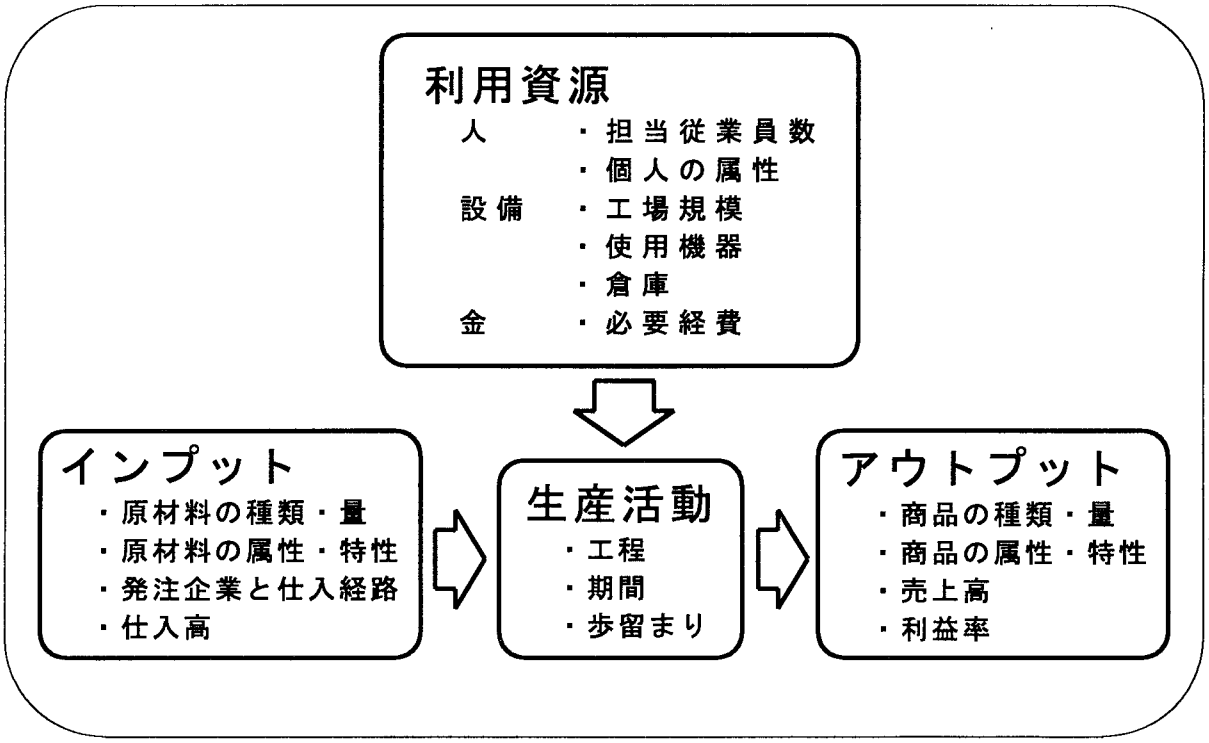
図Ⅱ-8　組織形態

荷する購買部や、できあがった製品を出荷する販売部、また、全体の経理状況を把握する経理部など多岐にわたる。どの部門からどのような情報を入手し、どの部門にどのような出力を行えばよいかを明確にするため、企業の組織形態とその業務を正しく認識しておくことが必要となる。なお、システムが非常に複雑な場合は、さらに調査する階層を深くし、課や係などの組織形態とそこでの業務を調べることとなる。また、逆にシステムの対象が１部門内だけで行われている場合は、全体の組織形態より、むしろ、その部門のなかでの組織形態を重点的に調査することが望ましい。

ハ　システムのインプット／利用資源／アウトプット

システムが必要としている原材料（インプット）、また、それを加工・組立するための利用資源（人、設備、金）、さらに加工された製品や情報（アウトプット）などの種類、量、特徴など（図Ⅱ-9）を調査する。

先にも述べたが、ここでの調査結果が以降の開発工程での基本設計やファイル設計、コード設計と大きく関わりをもつ。特に入出力の部分は、そのままデータの流れとなるので最重要項目となる。また、アウトプットでの売上高や利益率は、利潤追求という企業本来の目的を定量的に評価することのできる項目であるので、新システム導入時に現行システムと対比するため、過去数年間分は必ず調査しておく。

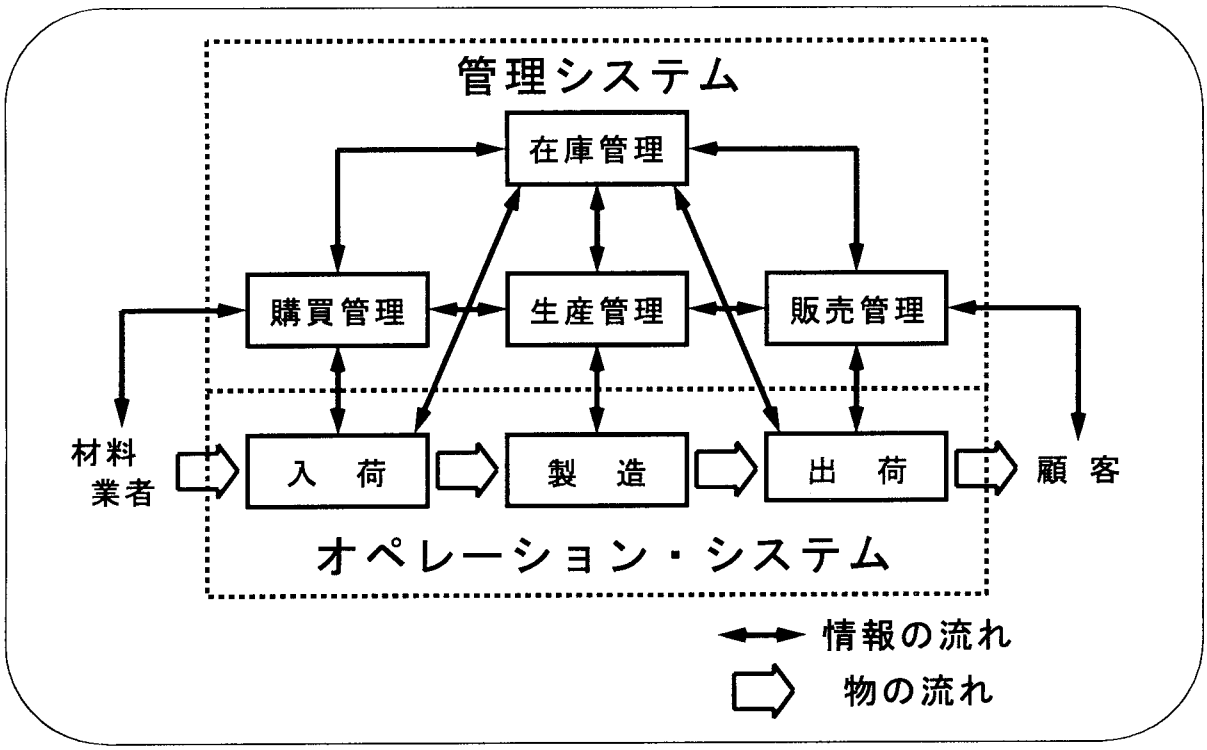


図Ⅱ-9　インプット／利用資源／アウトプット

ニ　システムのビジネス・オペレーション

システムを構成しているオペレーションシステムおよび管理システム（図Ⅱ-10）を調査する。

システムは、その活動を大別すると、二つのサブシステムから成り立っているといえる。一つは、商品を社会に提供するという企業目標達成のための活動（オペレーション・システム）であり、もう一つは、その活動を円滑に進めるために各種管理情報を把握し、また、それらを効率よく操作する活動（管理システム）である。



図Ⅱ-10　オペレーション・システムと管理システム

ホ　(イ)～(ニ)全般にわたって存在する各種の問題を調査する。

システムの機能と活動がうまく調和しているかどうかを調査する。特に、システムの機能に大きな問題があるのか、また、システムの活動に大きな問題があるのかを明確に区分するための調査を行う。

（3）ニーズ調査

現行のシステムにどのような問題点が生じていて、また、他にどのような影響を及ぼしているかを調査するには、直接、その業務の担当者から意見を聞くのも一つの方法といえる。当然のことながら、新しいシステムの開発を要求しているのだから、ユーザ側にそれなりのニーズがある訳である。しかしながら、人間というものは、日々同じ業務を繰り返していると「慣れ」というものが生じてきて、逆にその本質が見えてこなくなるところがある。また、ユーザ側は、基本的にシステム開発に関しては素人である（プロならば自分自身で新しいシステムを開発する。）。システム開発者からみれば、とんでもないニーズが多くあることも予想される。したがって、ユーザ自身が問題を深く認識するために、以下に示すニーズ調査を行うこととなる。なお、ニーズ調査を行う上で重要となるのが、その日程の調整である。開発者側およびユーザ側の日程を考慮した上で、アンケート形式によるニーズ調査を行うのか、それとも会議形式によるニーズ調査を行うのかを判断する。

イ　アンケート形式による問題点の調査

ＰＮカード（Problems－Needs カード：図Ⅱ-11）とよばれるカードを用いて、現存する問題を、影響／結果、問題点、背景／原因、目的／狙い、ニーズに区分し、アンケート形式で調査する。ＰＮカードは、より問題を鮮明化させるために、定量的に表現できる部分はその値をできる限り記入するように心掛ける。また、どこで問題やニーズが発生しているのかを明確にするために、部門や課なども記入する。

なお、カード収集時に、その主旨や表現が不明瞭もしくは担当者に分かりにくいと感じた場合は、記入者の同意を得て適切な表現に訂正する。

ロ　会議形式による問題点の調査

短期間で問題を収集しなければならない場合や、また、ユーザ側の都合によりアンケート形式が困難な場合は、会議形式で問題点を調査する。

まず、現状業務の関係者に集まってもらい、業務が抱えている問題点に関して、できるだけ多くの意見を自由に発言してもらう。ここでは、以下の点に注意する。

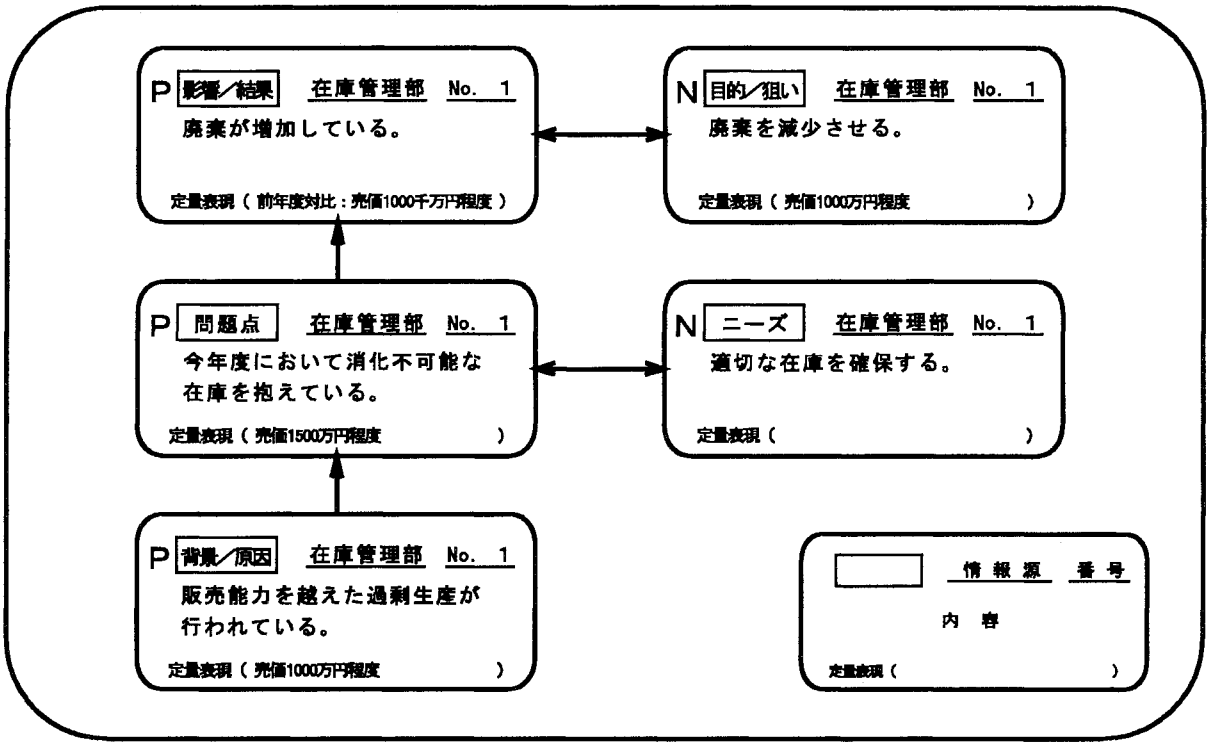
・　他人の意見は批判しない。

　できるだけ多くの意見を発言してもらうという目的から、他人の意見を批判することは控えてもらう。ここで行っているのは、分析ではなく調査なので、事の良し悪しはあまり考えずに発言してもらう。

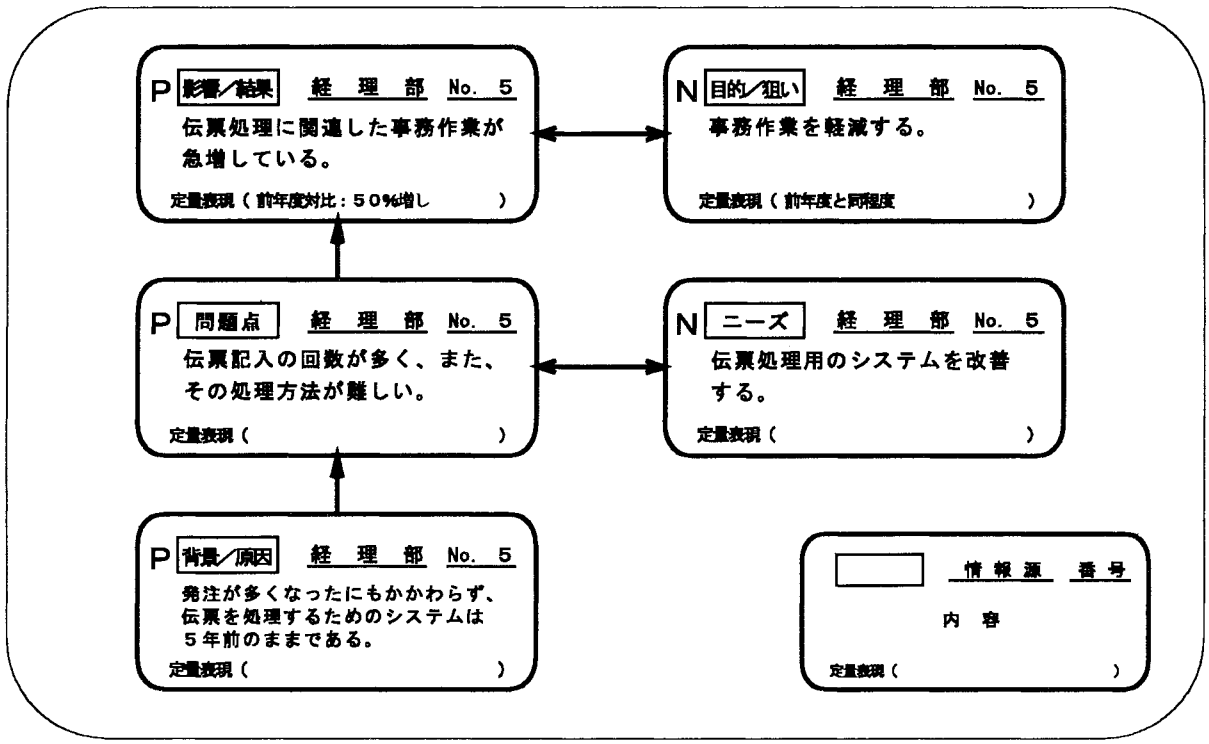
・　意見は質より量を重視する。

　問題点にあまり関係がないような意見でも、以降の工程を進めるに当たり、参考になることもあるので、質より量を重視した発言をしてもらう。

一般的に、ニーズ調査が現場サイドの声を直接聞ける最良の調査法と思われ、また、これで事足りそうな感じを受けるが、あくまでこの調査法は、ユーザの主観的な声しか聞けないというところを認識しておかなければならない。システム開発者は、ユーザのニーズを大切にしつつ、システムが本来の理想的な形態から離れることのないように、先に述べた環境調査と現状調査を独自に行う必要がある。



図Ⅱ-11(a)　ＰＮカードの記入例１

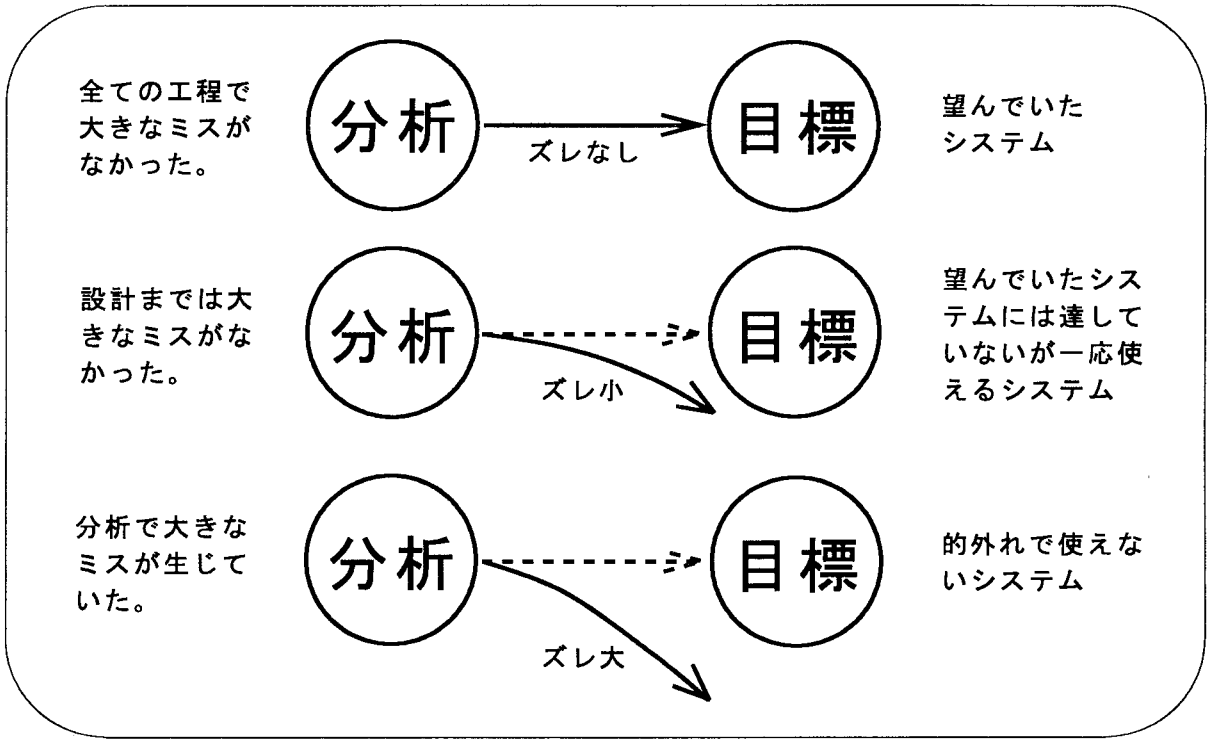


図Ⅱ-11(b)　ＰＮカードの記入例２

3　分析

環境調査や現状調査などの調査結果に基づいて、現行システムの機能分析を行い、問題となる部分を抽出していく。また、ＰＮカードによるアンケート調査結果で得られたユーザからのニーズも含めて分析することも必要となる。ここでは、業務ばかりに気をとられると、偏ったシステム分析しかできないので、データの流れなどの他の部分にも充分気をつける。その結果、浮かび上がってきた問題点に対する解決策を考える。分析がうまくいかなければ、それ以降の工程がうまくいっても望んでいるシステムを開発することはできない。「終わりよければ全てよし」の発想は、システム開発において通用しないのである。

分析でのズレは、下流工程に行けば行くほど大きくなり、ときには修正だけでは対応することができず、はじめからやり直さなければならないといった状況に陥る場合もある（図Ⅱ-12）。分析工程では、新システムにおいて何をするのかを具体的にする。



図Ⅱ-12　分析の重要性

（1）環境調査と現状調査からの分析

独自に行った環境調査と現状調査をもとにシステムを分析する。ここでは、特に、現状調査で得た情報を中心に分析していくこととなる。

システム開発者は、現行システムの調査を進めていくうちに、何やら道理のかなっていな

い不思議な状況に遭遇することが多々ある。例えば、本来、経理部で発行するはずの伝票が購買部で処理されていたり、販売部で顧客管理の一部として行われるべき業務が製造部で行われていたり、また、いかなる業務とも無関係な情報が存在するなどの状況に直面することがある。これは、企業とそこで行われる業務が、企業内外の環境変化や各部門の力関係に左右されていることを示しており、また、企業自身がシステムの見直しを怠っているという事実を示している。また、情報に関しても同様で、それを処理する方法や道具などにより、本来必要とされている項目が抜け落ちていたり、また、逆に必要でない項目が付加されていたりするという事実を示している。

このような事実を無視して分析を進めても、システム内に存在している諸問題を解決することは不可能である。現状調査の結果をもとにした分析では、その企業が、どのような組織形態で動いているか、誰が何をしているか、また、どのような道具を用いてどのようなことを行っているかなど、現状システムでの方法を全て取り除いて、本質的にどういう機能が満たされていなければならないのか、また、その機能が満たされるためには、どのような情報が必要であるのかを明確にしていく。これは、システムの定性的な分析であり、この分析結果により、システム開発者は、その企業における情報システムの理想的な形態をイメージすることができる。そして、これらの機能を、どの部門で、誰が、どのような道具で、どのような方法を用いて行うかについてを、以降の工程である基本計画や外部設計で定量的に考えていくこととなる。

（2）ニーズ調査（ＰＮカード）からの分析

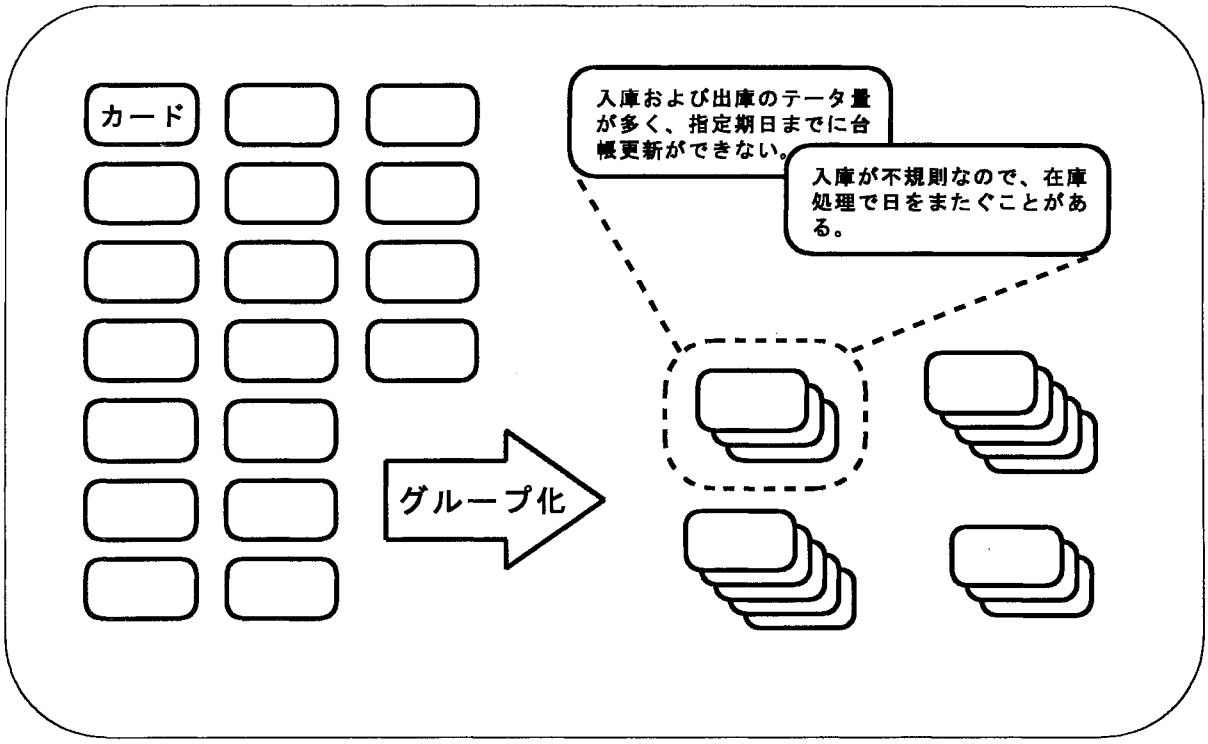
ユーザ側からの要求であるニーズ調査の結果をもとにシステムを分析する。ここでは、ＰＮカードを中心とした調査結果をもとに分析を進めていく。

ｲ　カードのグループ化

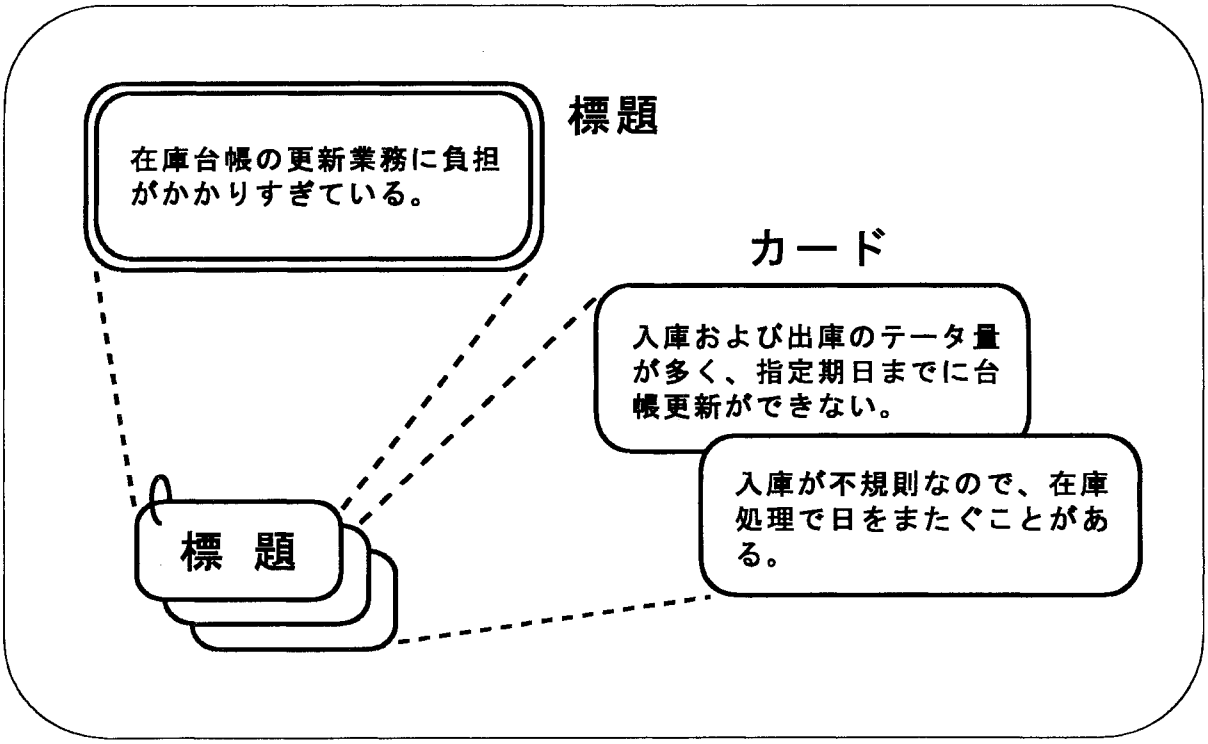
先に収集したPNカードをグループ化する。すなわち、個々の問題点が記入されているカードを、内容の対象が似ていると感じるもの同士にまとめていく（図Ⅱ-13）。さらに、グルール化した後に、各グループごとにサブテーマ名（標題）を付けていく（図Ⅱ-14）。サブテーマ名は、そのグループの抱える問題を的確に表現することのできるものにする。なお、あまり長いサブテーマ名は、逆に問題の意図を分かりにくくするので、できる限り簡単に記載する。

ﾛ　問題点分析

カードをグループ化することにより、幾つかのサブテーマに分割できたので、ここでは、そのサブテーマごとの問題点を分析する。



図Ⅱ-13　ＰＮカードのグループ化



図Ⅱ-14　サブテーマ名の設定

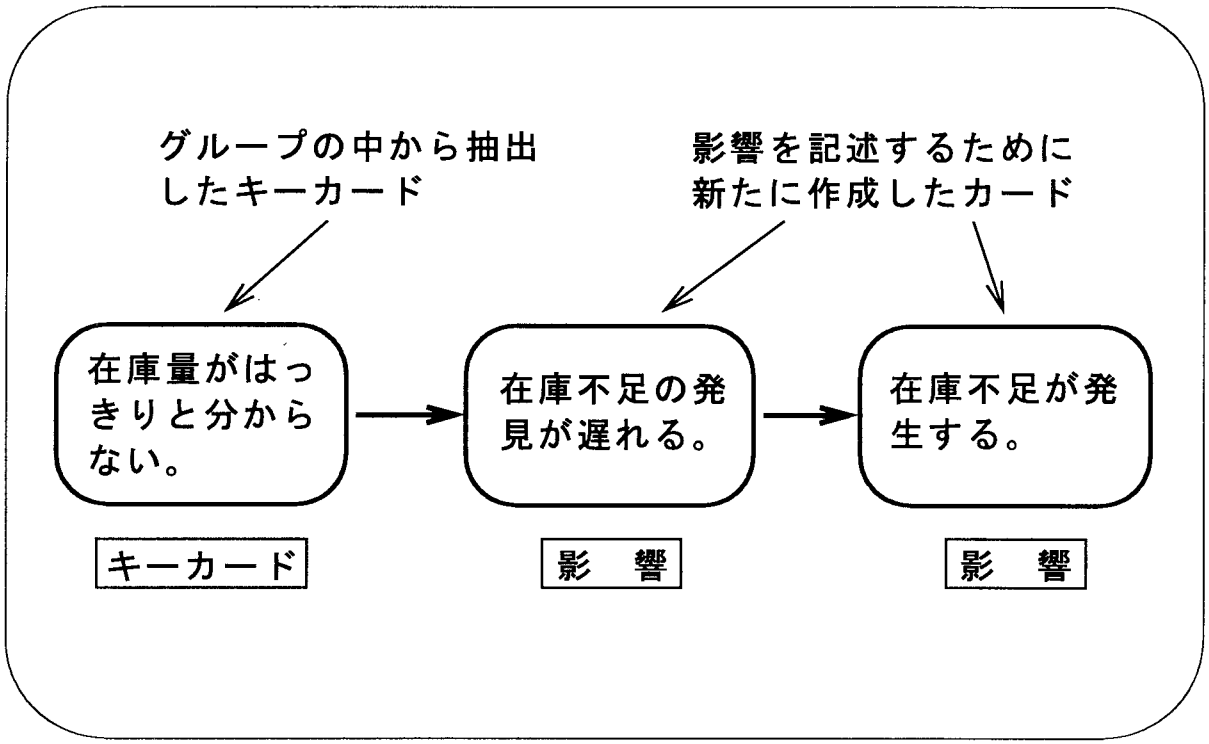
サブテーマにグループ化された問題の中から、テーマに最も近い表現、もしくは最も重要であると考えれる問題を一つ取り出し、これをキーカードとする。次に、キーカードを中心に置き、その右側に、問題点を放置しておくことによって引き起こされる各種の悪影響を順に考えていく。また、左側には、キーカードの問題がなぜ発生するか、その原因となるものを順に考えていく。これにより、問題点に対する原因と結果の因果関係が示され、問題の構造が明らかになっていく。

(イ) 影響分析

問題点の影響を考えるときは、図Ⅱ-15に示すように、段階的により大きな影響へと発展させていく。

影響分析は、その最終点がコストやサービス、売上の低下というような経営との関係が明確になる段階まで進めていく。なお、分析していく課程においては、論理がいきなり飛躍しないように注意し、他人が見ても論理の展開が明確に分かるように心掛ける。

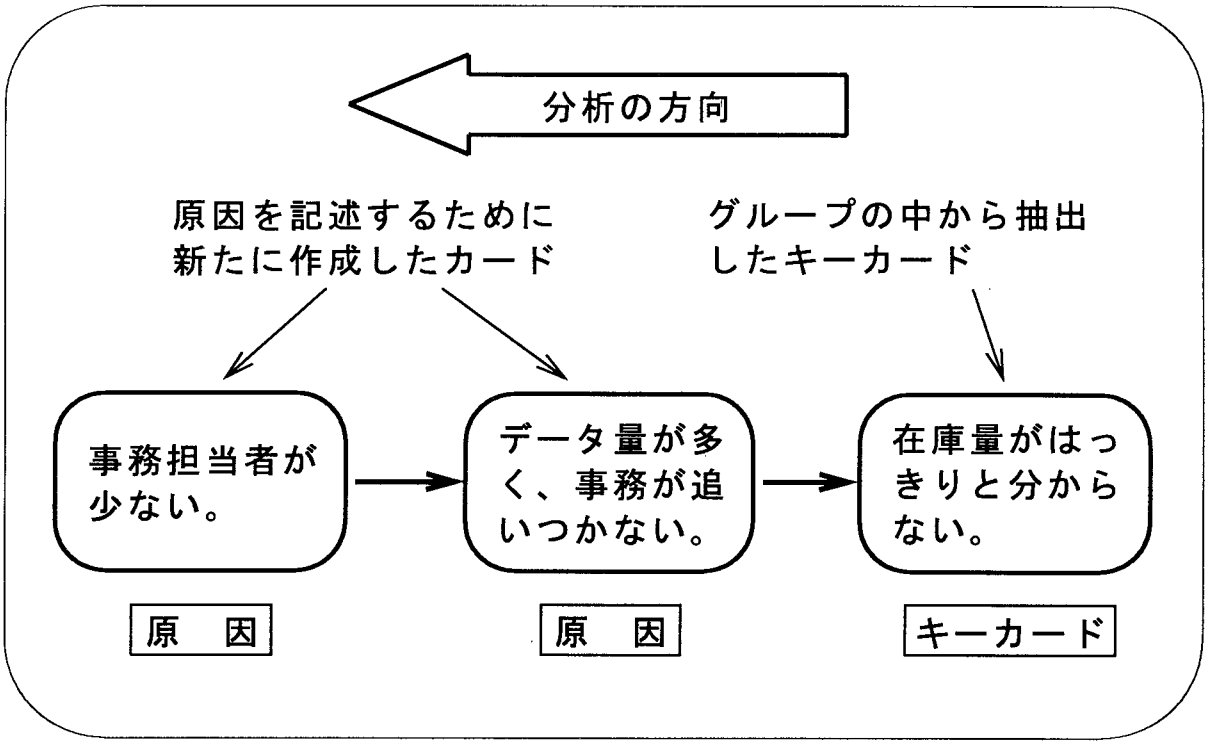
このように、問題点が業務へ及ぼす影響を分析していくことで、問題をどういった観点からとらえるべきなのか、また、何が本当に重要な問題となっているのかなどを系統立てて明らかにすることができる。



図Ⅱ-15　影響分析

(イ) 原因分析

ここでは、影響分析と逆方向に、問題がなぜ発生するのか、その原因を考える。分析方法は、影響分析のときと同様で、段階的に分析していくわけだが、図Ⅱ-16に示すように、原因分析のときは、問題をより細分化していく。

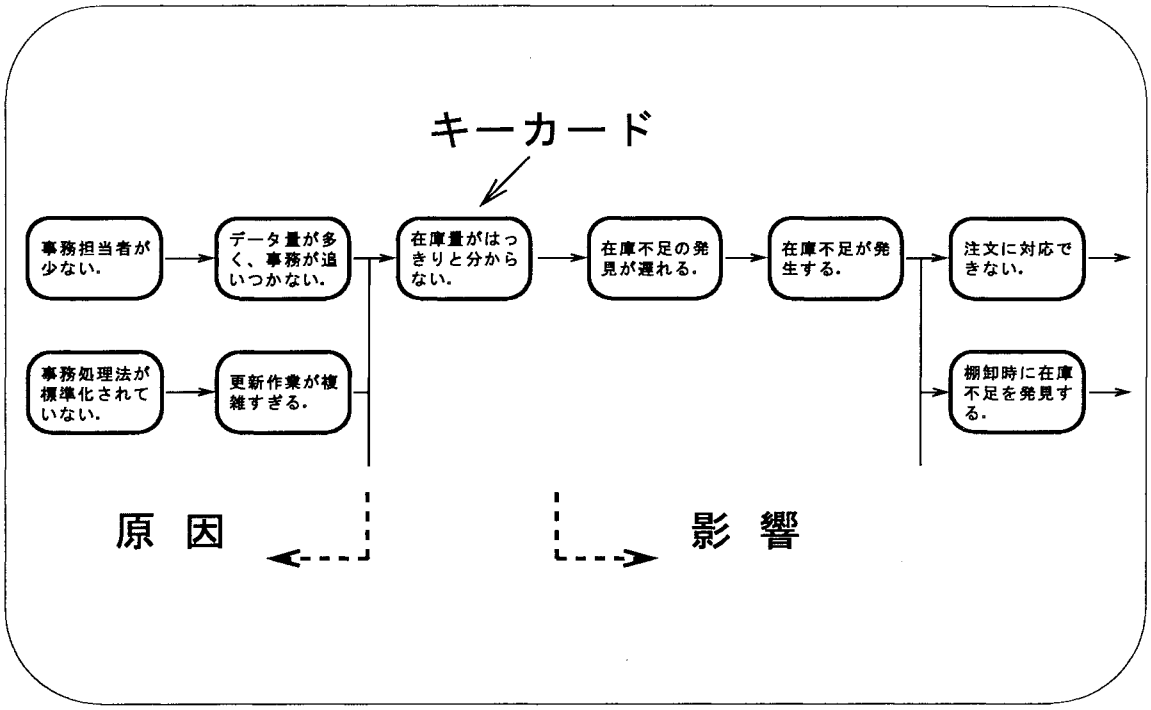


図Ⅱ-16　原因分析

原因分析は、その最終点が、担当者レベルでは対応できないという内容になるまで進めていく。すなわち、経営者レベルが動かないと、問題が解決できないというところまで分析を進める。

以上の影響分析および原因分析により、問題点の業務に対する影響と問題点の原因を明確にすることができ、問題の全体的な構造が明らかとなる。なお、影響分析においても、原因分析においても、その影響や原因となるものが一つとは限らないので、このような場合は、一つのカードから二つ以上の影響や原因が発生してもよい。なお、このときは、影響や原因の度合い（どちらにより重みがあるか。）を考える必要はない。

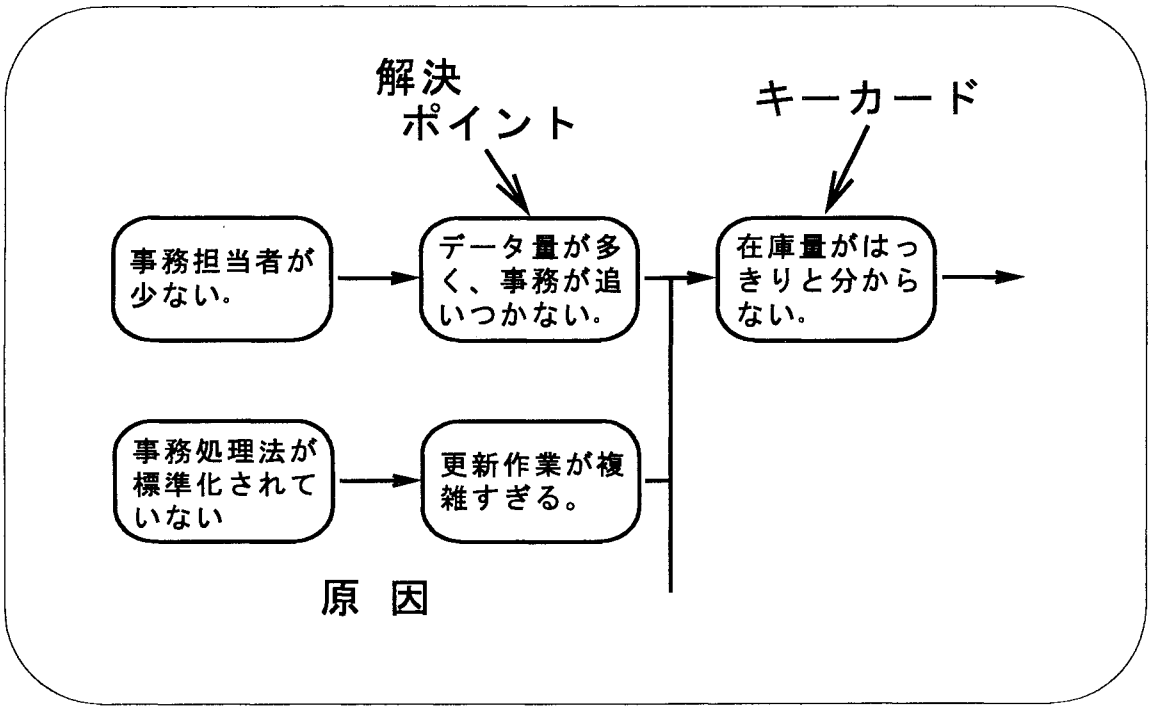
問題点分析においては、最終的に影響分析と原因分析を組み合わせた問題点ネットワーク図（図Ⅱ-17）を作成する。



図Ⅱ-17　問題点ネットワーク図

ハ　解決ポイントの決定

問題点ネットワーク図の原因側から、キーカードに対する最大の問題、すなわち問題全体の根本原因となっている問題を抽出し、これを解決ポイントと選定する（図Ⅱ-18）。



図Ⅱ-18　解決ポイントの選定

以上の手順を各サブテーマごとに行い、それらの解決ポイントを決定する。

ニ　目的展開

ここでは、先に決定した解決ポイントをもとに、問題解決の目的を見直す。目的とする新システムが達成すべき内容は、当然、解決ポイントの解決案を含んでいなければならないが、それだけでは充分でなく、より広い範囲で解決案を検討する必要がある。この範囲の決定が目的展開の作業である。

目的展開は、以下の手順に従って行う。

(イ) 解決ポイントの問題を解決するための特定手段を決める。

解決ポイントの内容は否定表現を用いているので、それを肯定表現に変え、特定手段とする。

特 定 手 段 ： 更新事務を予定時間で済ませる。

解決ポイント ： データ量が多く、事務が追いつかない。

(ロ) より上位の目的を展開する。

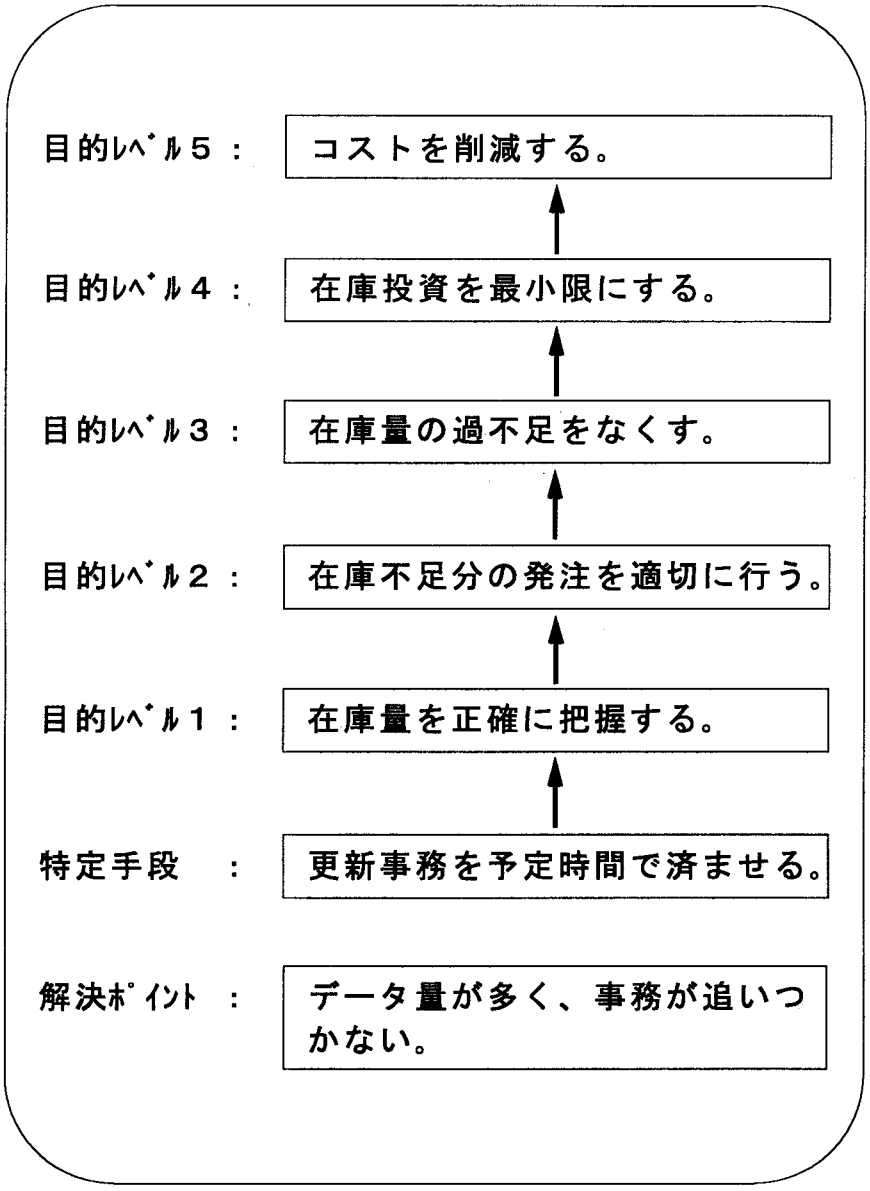
特定手段を出発点とし、より上位の目的を展開する。すなわち、特定手段の目的は何であるのか、また、その上位の目的は、何であるのかをより上位の目的に向かって展開していき、これを目的展開図（図Ⅱ-19）としてまとめる。さらに、その中から、どの目的レベルを新システムが達成すべきかを決定する。すなわち、目的レベルの決定は、新システムによって解決する機能の範囲を決定することと同等となる。

例として、ここでは、目的レベル３に位置する 在庫量の過不足をなくす を新システムが達成すべき目的とする。当然のことながら、より上位のレベルを達成すべき目的とするのがよいが、上位になればなる程、その目的は具体的なものから抽象的なものへと変化していき、その実体が分かりにくくなる。また、範囲が非常に広くなってくることから、その目的を達成することはより困難となる。ここでは、システムの機能が、実際にどこまでなら対応できるのかというところをよく考え、達成すべき目的レベルを決定する。

次の手段の検討においては、ここで決定した目的を達成するための手段を検討していく。

(ハ) 手段の検討

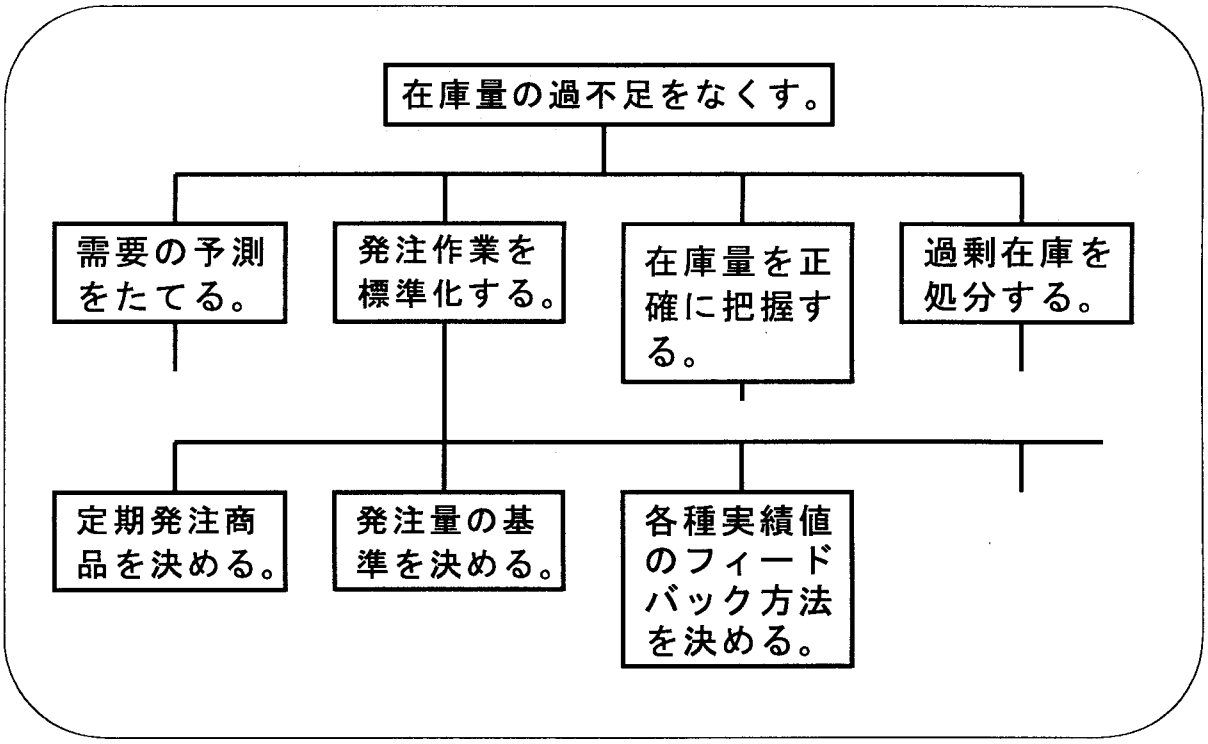
解決すべき目的レベルに対して、その目的を達成するための手段を検討する。ここでは、業務方法の改善だけを考えて手段を検討するのではなく、その業務で処理するデータおよび各業務間のデータの流れも考えた上で、目的達成のための手段を検討する。当然のこと



図Ⅱ-19　目的展開図

ながら、現実性や経済性などが含まれていることはいうまでもない。なお、検討していく過程においては、必ずより多くのメンバーで、複数の案を具体的に作成するようにする。これは、より多くのメンバーが集まることにより、より多くの意見が出され、その結果、より多くの解決案が提示されるという利点があるからである。また、より多くのメンバーが集まることにより、問題を収束させるための意見が多く求められ、さらに、問題を具体化する手助けとなる。

このようにして体系化されてきた手段を手段体系図（図Ⅱ-20）としてとりまとめる。



図Ⅱ-20　手段体系図

（3）分析結果の統合化

環境調査・現状調査からの分析とニーズ調査からの分析を統合化する。通常の論理展開からいくと、これらの分析結果は、ほぼ同等と考えられるが、分析するに当たったグループやそのメンバーの特徴により、全く異なった結果が出てくる可能性がある。

基本的に、環境調査や現状調査からの分析は、システムにどういう機能が満たされていなければならないのかといった問題を、現状のシステムで用いている方法を全て取り除いて、1から考えている。それに対し、ニーズ調査からの分析では、基本的に、ユーザ側からの要求や改善してほしい点を中心に、現状システムをどう改良したらよいのかという観点で分析を進めている。

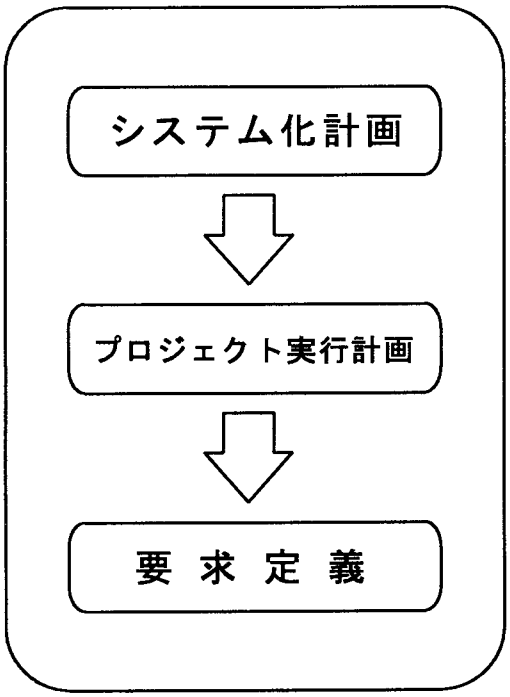
ここでは、Ⅰの1で述べたように、あくまで「お客様」という相手の要求を最優先して分析結果の統合化を図る。

4　基本計画

ここでは、Ⅱの3で述べた分析結果をもとに、新システムが実現可能なシステムであるかどうかを検討する。特に、図Ⅱ-20などで示した業務方法や、そこで行われるデータ処理および各業務間のデータの流れが本当にスムーズにいくのかどうかというところを重点的に検討する。

また、新システムにおいて、どこまでコンピュータ化を行うかなどの事柄についても検討する（コンピュータは定型業務には向いているが、あいまいな要素が加わる業務には向いていない。全てをコンピュータで置き換えることは、現在の技術では基本的に不可能である。したがって、費用対効果を考えた上で、どこまでコンピュータ化するかが重要となる。）。

基本計画は、図Ⅱ-21に示すようにさらに三つの工程に分けられ、それぞれの工程で表Ⅱ-1に示すドキュメントが作成される。



図Ⅱ-21　基本計画の手順

表Ⅱ-1　基本計画で作成するドキュメント

工　　程	作成するﾄﾞｷｭﾒﾝﾄ
シ ス テ ム 化 計 画	システム化計画書
プロジェクト実行計画	開 発 計 画 書
要　求　定　義	要 求 仕 様 書

（1）システム化計画

システム化対象業務についての基本計画を立案する。

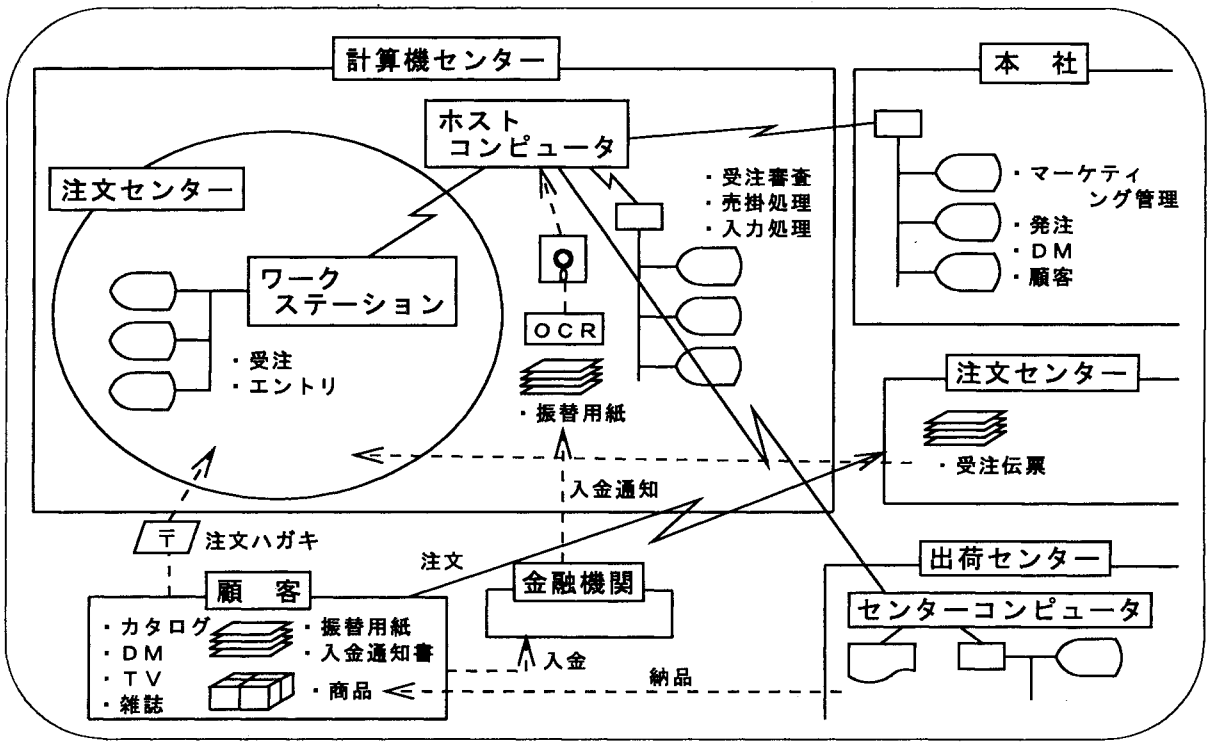
目　的

開発対象となっているシステムの目標と目的をより明確にし、システム開発を行うかどうかを決定する。

内　容

調査・分析結果をもとに、現状業務の問題点の解決案を検討する。そして、その検討結果から、新システムが実現可能かどうかを決定する。このため、開発するシステムの概念設計を行う必要がある。

概念設計とは、システムの完成予想図（図Ⅱ-22）を作成することである。つまり、システム化後の状態を大まかに予想・整理し、現状との相違点を明確するとともに、新システムの基本機能や開発目的を明確に定義することである。なお、このときは、利潤追求という企業本来の目的、すなわち費用対効果も明確にしておく。



図Ⅱ-22　システム完成予想図（概略イメージ図）

主な作業内容を以下に示す。

・　現状業務の整理

　現状の処理の流れやデータの流れを整理する。

・　システム化の領域の明確化

どこまでシステム化するかを明確にする。特に、どこまでコンピュータを導入するかを明確にする。

・ 開発計画の概要把握

開発に関わる費用、期間、体制、規模などを把握する。特に、費用については、開発するシステムのハードウェアとソフトウェアのコストだけでなく、開発に携わる人員のコストを充分考慮しておく。

・ 費用対効果の検討

開発するシステムが、果たして費用に見合った効果が得られるかどうかを検討する。

・ 類似システムや最新の技術動向の調査

近年のソフトウェアは、ハードウェアの高速化や大容量化、そして低価格化などにより、非常に大規模・複雑多岐なものになってきている。開発するシステムに、より最適なソフトウェアおよびハードウェアを調査する。

・ 開発における制約事項の整理

後々のトラブルを避ける目的から、ユーザ側と開発者側との間で各種の制約事項を設けておく。システム開発の結果が、ユーザ側の求めていなかったものであった場合、その全てが開発者側の責任であるとは一概にいえない。

また、開発期間に関する問題が往々にして起こるということも考えられる。開発が予定より遅れてきたからといって、過剰労働を行うことは好ましくない。

さらに、その企業独自の業務方法や生産物があった場合は、特に注意をはらわなければならない。他の企業に秘密を漏らすということは絶対に行ってはならない。

ここで設ける制約事項の違反は、最悪の場合、法的な部分が導入されるということにもなりかねないので、開発者側は綿密に検討しておく必要がある。

類似システムや最新の技術動向の調査は、特に費用対効果と関連がある。類似システムの調査により、開発するシステムに似たシステムを見つけだすことができれば、それを導入することにより開発費用を抑えることができ、また、開発期間も短縮することができる。特に、近年のソフトウェアは、高機能化・細分化、そして、追加機能およびネットワーク対応が当たり前となっているので、少し手を加えることで、かなり大規模なシステムに対応できる。

さらに、最新の技術動向を調査することにより、同性能でより安いコンピュータを導入することができれば、開発費用をさらに抑えることができる。複雑な科学技術計算を行うためのスーパーコンピュータなどは例外であるが、近年は、パソコンが飛躍的に性能向上してきたことから、従来ワークステーションが必要であった部門でもパソコンで対応できるという可能性もでてきている。ハードウェア自体の価格差はそれほどないが、その上で稼働するソフトウェアの価格には依然として大きな開きがあるので、開発費用を考えた上で慎重に検討すべきである。

以上の調査・検討結果を「システム化計画書」としてとりまとめ、開発を行うかどうかの判断を仰ぐ。

（2）プロジェクト実行計画

システムの開発決定に基づき、実行計画をたてる。

目　的

プロジェクトを実行するためのチームの結成および作業の準備を行う。この段階で開発計画は明確となるが、開発予算や資材調達、そして開発に関わる人員などの問題から、場合によっては、開発が中止されることや開発時期がずらされることもある。

内　容

システム開発の体制やスケジュール、作成するドキュメントの内容、資材調達の計画などを決定する。これにより、以降の工程がスムーズに行われるようにする。

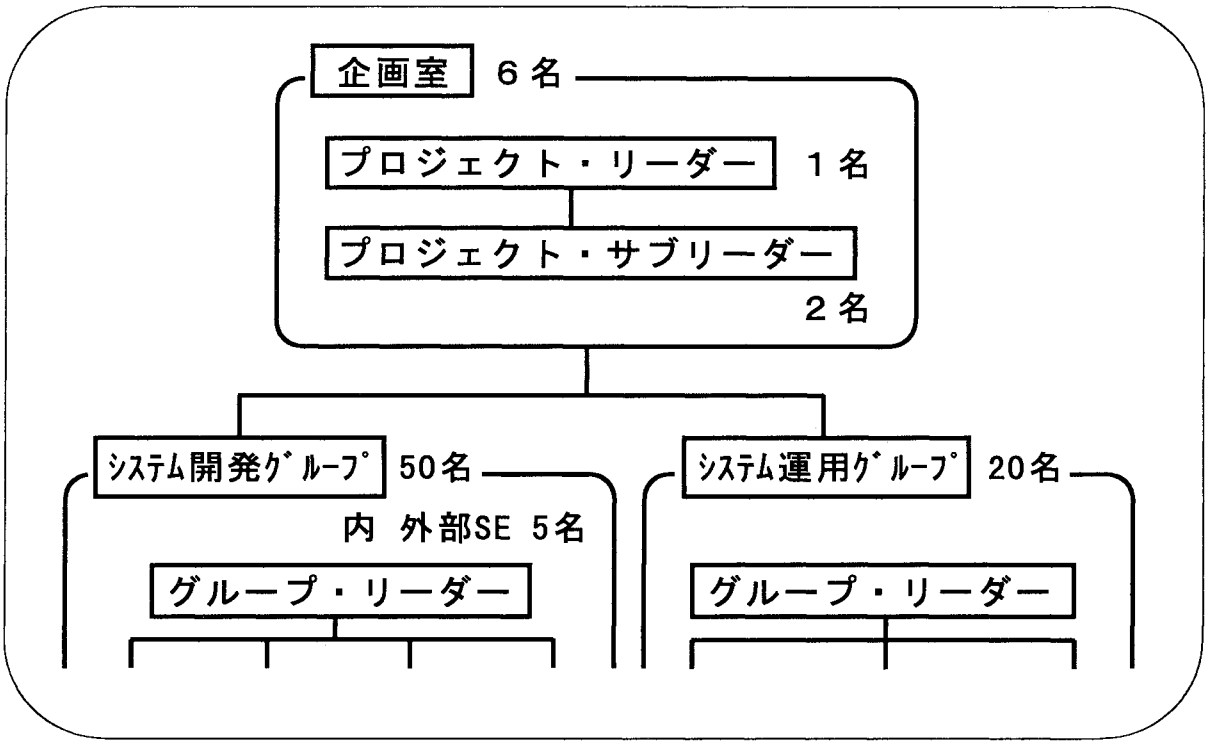
イ　開発体制の決定

ここでは、図Ⅱ-23に示すように、大きく分類された各工程ごとにどのようなチーム編成で取り組むかを明確にする。比較的小規模なシステムの場合は、体系的に各個人レベルまで人員配置を決定することが可能だが、大規模なシステムになればなるほど、後の工程であるプログラム設計からテストまでを担当する人員が増加し、的確な人員配置を決定することが困難となる（図Ⅱ-24）。このような場合は、予測される開発進度にあわせていくつか回避策を立てておく必要が生じる。

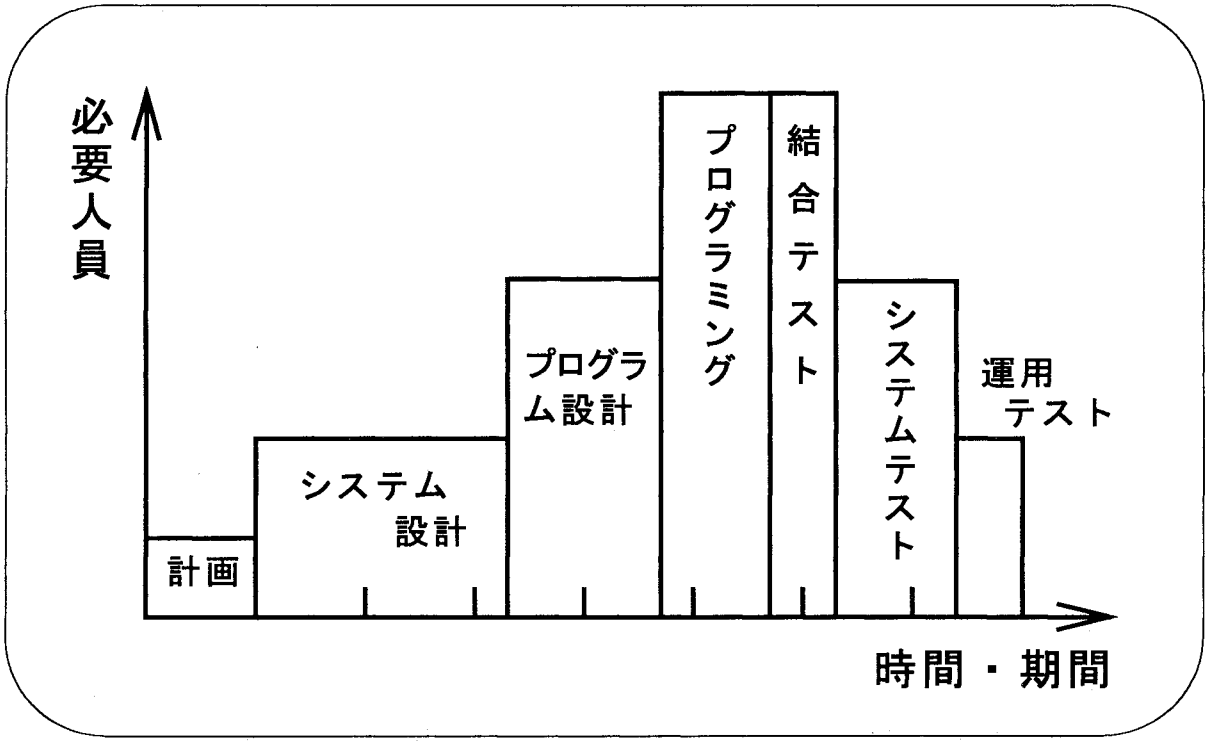
また、各工程には必ずリーダーが存在しなければならないが、そのチームの意見が全てリーダーに集中するようでは、逆に、リーダーが本来行わなければならない業務が緩慢になることが考えられる。また、各メンバー間での情報伝達についても同様である。このような状況が生じたときは、場合によって、図Ⅱ-25に示すようにサブリーダーをおくことでリーダーの業務集中を回避し、さらに各メンバー間での過剰な情報伝達をカットする。

なお、開発体制を決定するときは、図Ⅱ-23に示したように、必ず体系的に図で表現することが重要となる。文書だけで、このグループは何人、このチームは何人という設定をすると、実際にシステム開発を行っていくときにグループ間やチーム間で不整合が生じることがある。

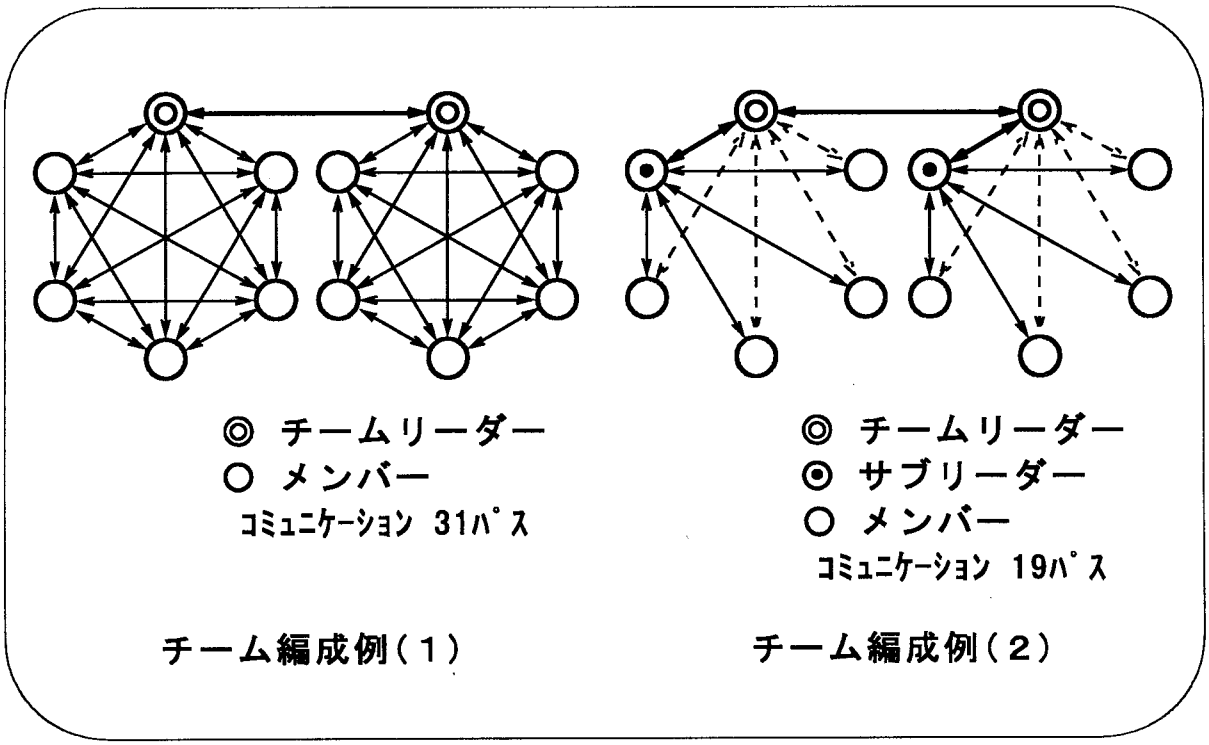
また、企業は、一つだけのプロジェクトを受けもっているのではなく、通常、複数のプロジェクトが並列に進行している。一度決定された開発体制は、途中で人が足りなくなったからといって、容易に追加変更できるものではないので、各個人の能力や適性、スケジュールなどもよく加味して開発体制を決定する。



図Ⅱ-23　開発体制



図Ⅱ-24　各開発工程における必要人員と時間の関係



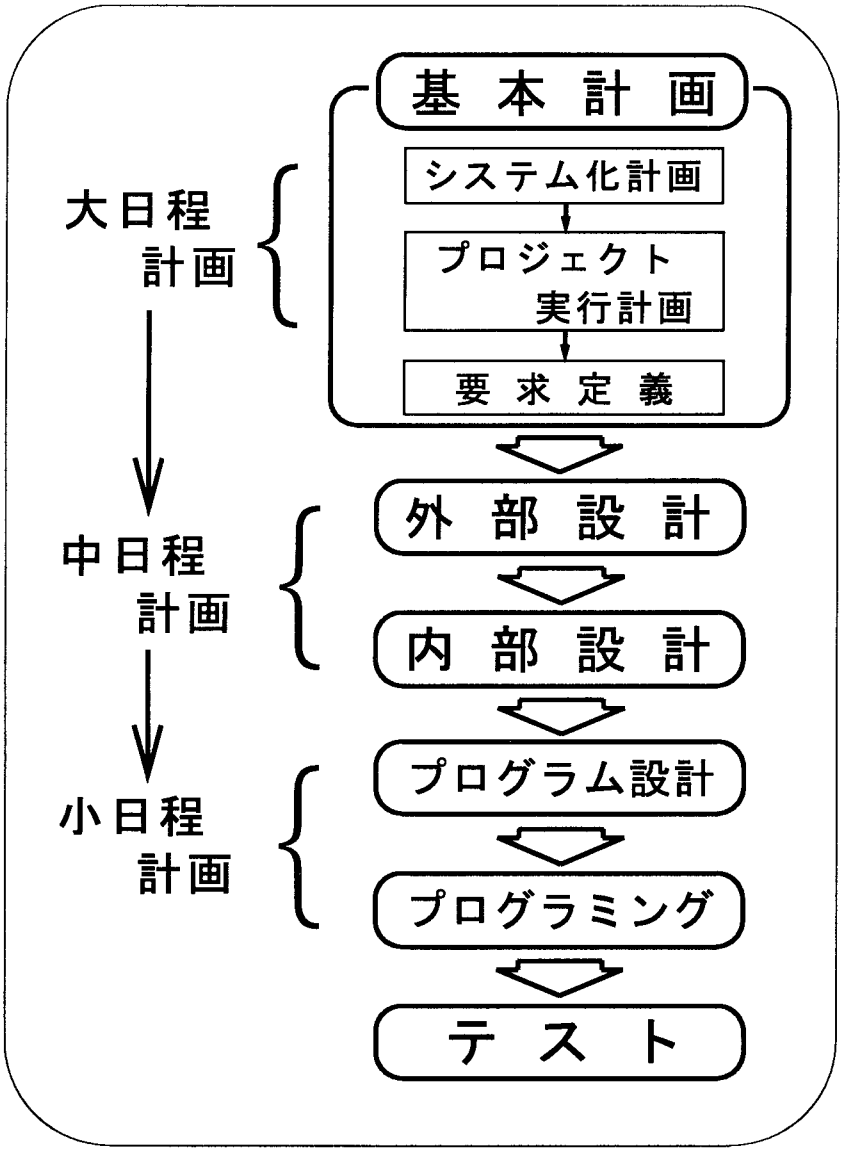
図Ⅱ-25　チーム編成

□　スケジュールの決定

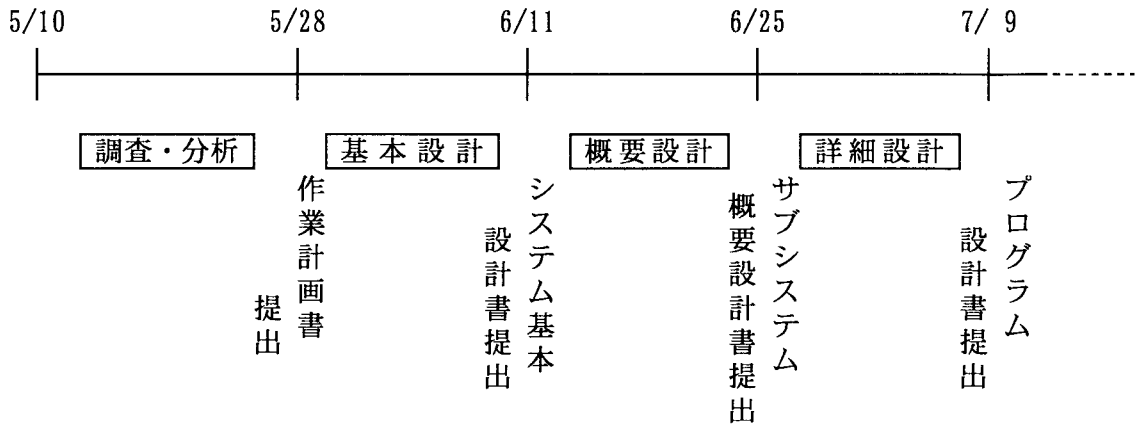
これについても、開発体制の決定と同様、後の工程のことを考えて、全ての日程計画（図Ⅱ-26）をこの段階で作成しておくことが望ましい。しかしながら、現実では、システム開発が進むにつれて明確になっていくことが多い。なぜならば、上流工程で立案された様々な計画によって、下流工程の必要人員や必要時間が決定されるからである。また、システム開発は、単なるプログラム開発とは違うので、その複雑さゆえに問題発生はつきものである。修正すべき問題点が挙がってきたような場合は、日程調整が必要なることは言うまでもない。要するに、最初からあまり綿密な日程計画を立てても意味がない場合があるということである。

したがって、ここでは最低限、「大日程計画」を作成することとする。図Ⅱ-27、図Ⅱ-28に大日程計画および中日程計画の一例を示す。

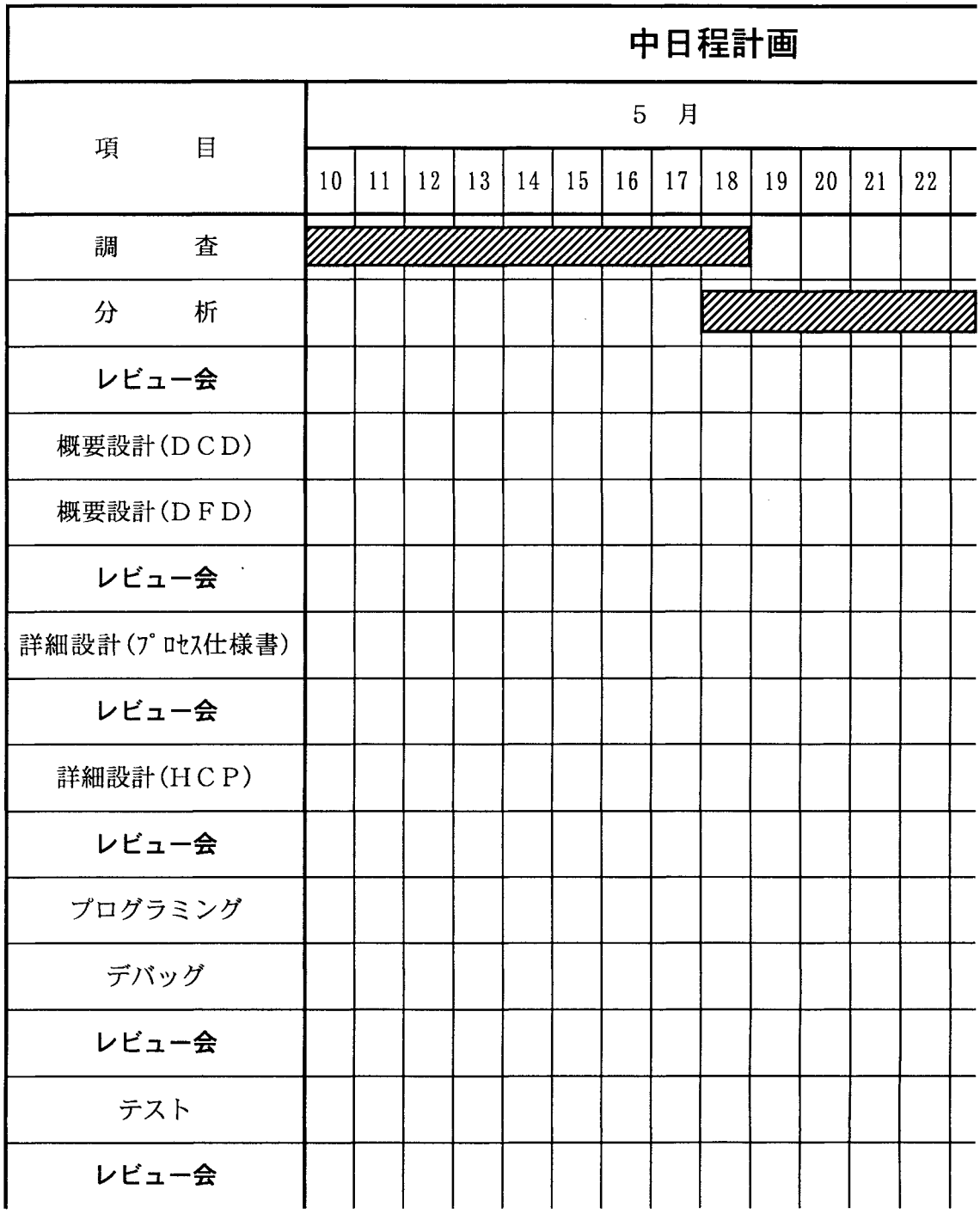
なお、大日程計画とは、システム開発の全行程におけるスケジュールを表した日程計画であり、中日程計画は各作業工程ごとの日程計画、小日程計画は個人別の日程計画を表したものである。ただし、小日程計画まで進まないと個人別の日程を組んではならないという規定はなく、むしろ、開発体制をより明確にするために、中日程計画においても個人別の項目を導入する方がよいと考えられる。



図Ⅱ-26　日程計画の種類とそれが作成される工程



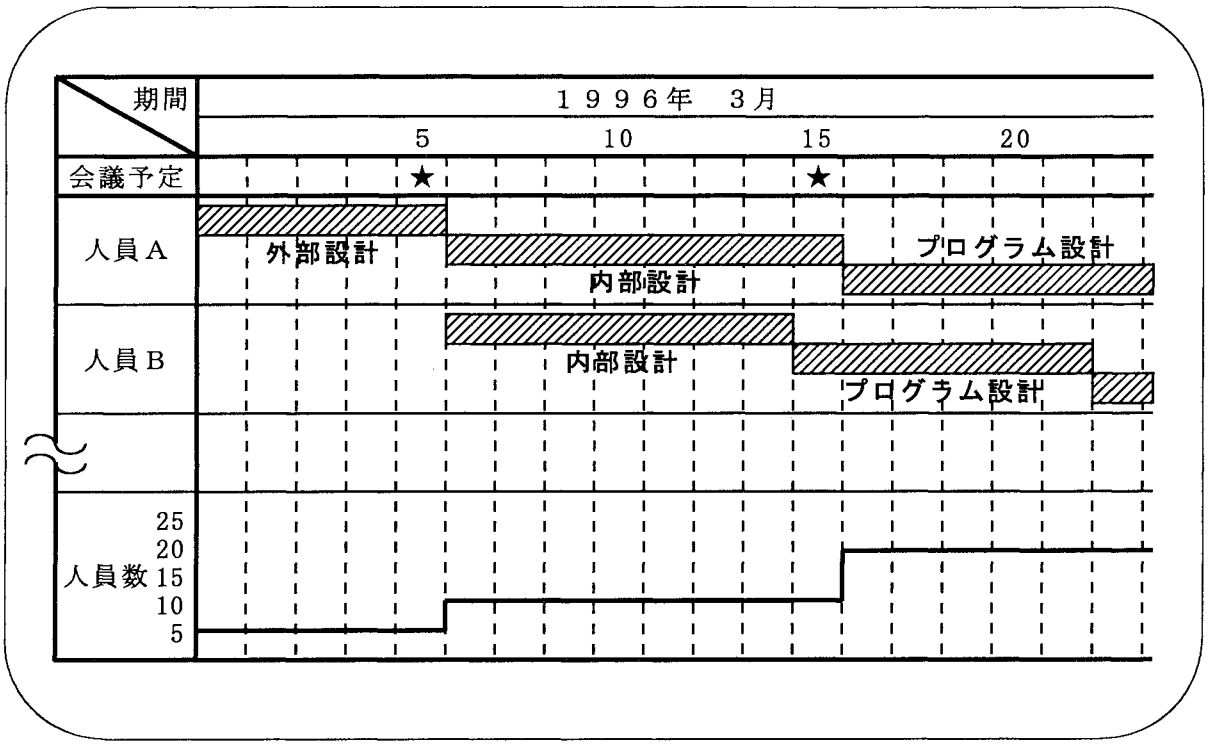
図Ⅱ-27　大日程計画



図Ⅱ-28　中日程計画

(イ) ガントチャート

先程述べた日程計画のそれぞれの長所を統合したガントチャート（図Ⅱ-29）というチャートがある。ガントチャートは、横軸を時間、縦軸を担当者とし、チャートに作業内容と作業時間を棒グラフで示す。



図Ⅱ-29　ガントチャート

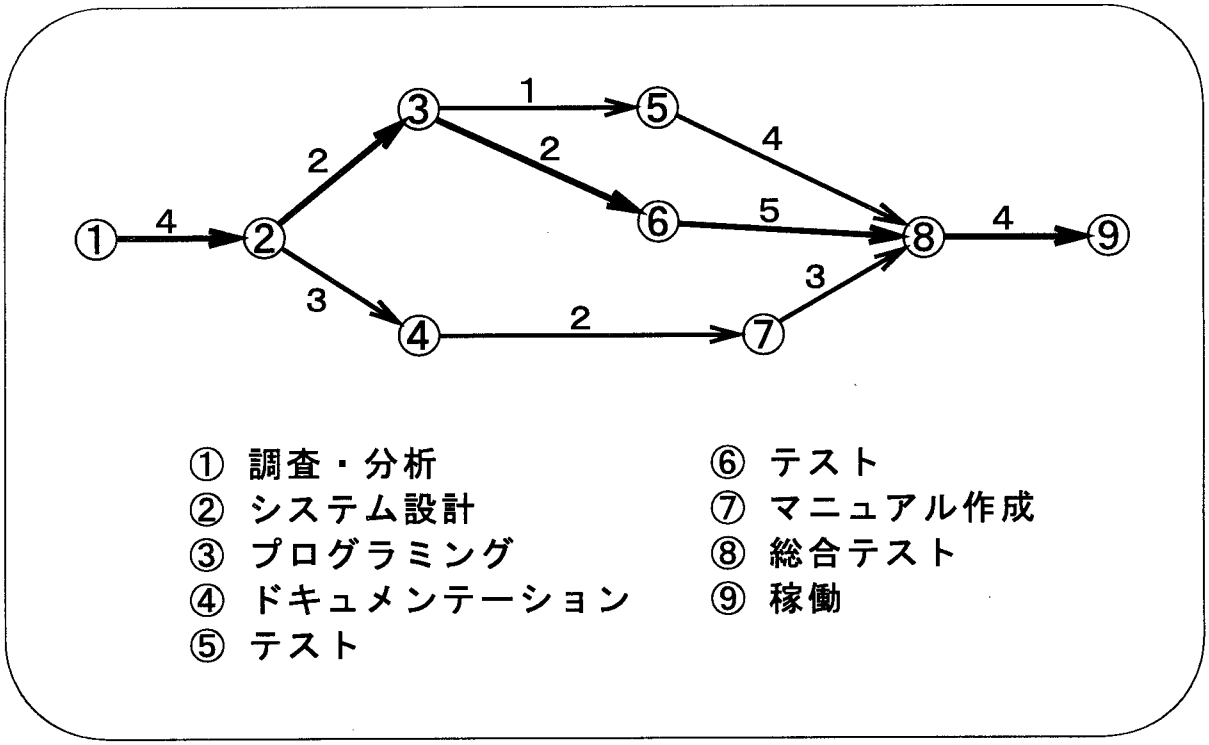
ガントチャートの特徴は、システム開発全行程における日程とその進捗状況を一つのグラフで表せることである。システム開発には、開発の開始時期と納期が決まっているので、各工程でどのくらいの時間が必要かを想定すれば、それに必要となる人員の割り当ても、だいたい決まってくる。

また、納期やレビューなど、日程上移動することが困難な項目に対しては、マイルストーンとよばれる目印を記入する（目印の種類は何でもよいが、一目で分かるものにする。ここでは★印を用いている。）。

このように、ガントチャートでは、各工程において必要となる人員やその作業時間を明確に表すことができるので、全体的な作業量の把握がし易くなる。しかしながら、ガントチャートでは、各工程間の作業の関連を表すことができないので、ある工程で遅れが生じたときに、全体に対してどのような影響が生ずるかを示すことが不可能である。したがって、大規模なシステム開発を行うときには、そのような問題点を回避するため、次に述べるＰＥＲＴが用いられる。

(ロ) ＰＥＲＴ

ＰＥＲＴ（Program Evaluation and Review Technique：図Ⅱ-30）は、従来のガントチャートに代わる新しい計画手法として、複雑なシステムの開発をより弾力的に達成するために用いられるようになった。



図Ⅱ-30　ＰＥＲＴ図

ＰＥＲＴは、開発作業の順序や相互関係を矢印（アロー）で表したものである。○はイベントとよばれ、その中に一連の番号を記し、この番号に対応した作業内容は別の場所に記される。アロー上に設定された数字は作業（アクティビティ）とよばれ、その作業に要する時間は、日数や相対的な期間で示される。時間的に余裕のないイベントを結んだ経路はクリティカルパスとよばれ、このパス上にあるイベントの遅れは、開発工程全体の遅れにつながることとなる。したがって、このパスを重点的に管理することにより、システム開発がスムーズに行えるようになる。ＰＥＲＴを用いた工程管理では、ネットワーク形態を採用していることから、作業開始後の進行管理や統制が弾力的に行われ、また、開発工程全体の関連がシステム的に考えられていることから、ある作業の遅れに対してのクリティカルパスの変化はもとより、スケジュールを組み直すときの統制ポイントなどが明確となる。

ＰＥＲＴによる開発作業では、まず、システム開発に必要となる作業を明確にし、その順序や必要となる期間や時間をネットワーク形態で表現した後、ノードタイムの計算およびトータルフロートの計算を経て、クリティカルパスを見つけるという方法がとられる。

なお、ＰＥＲＴ図は、システム開発の全体工程を示す大日程計画表に対応し、ガントチャートは、各工程の作業を示す中日程計画表および個人別の作業を示す小日程表に対応する。

プロジェクト実行計画で行う主な作業内容を以下に示す。

・　プロジェクトチームの結成、開発体制の決定、責任者の選出

プロジェクトチームの結成は、担当者の能力をよく考えた上で決定する。特に、チーム内のメンバーが事務的知識に豊富なのか、それともプログラムに関する知識が豊富なのかを明確に把握しておく。可能ならば人事部などと相談して、適性のあった者同士を同じチームに配属することも考えられる。いくら能力のあるもの同士が集まったとしても、基本理念が異なっていたら進むものも進まなくなる。これは、ごく一般的な社会現象である。

・　大日程計画表の作成

中日程計画表や小日程計画表を含む全ての日程計画表が作成できればよいが、最低限、大日程計画表だけは作成する。なお、大日程計画表は、各工程間の関連を知るため、ＰＥＲＴ図を用いて作成する方がよい。

・　ドキュメントなどの成果物の書式やそのとりまとめ方法の決定

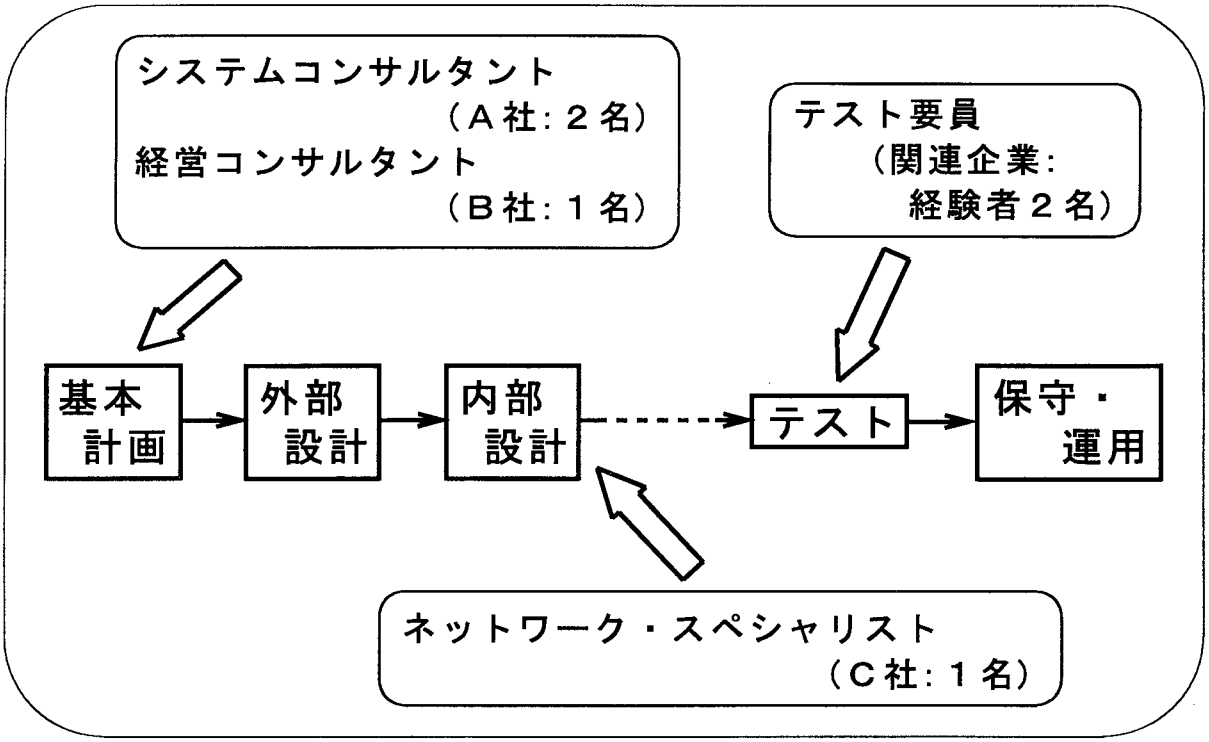
作成するドキュメントは、その企業独自のものが非常に多い。したがって、その企業のスタイルにあったドキュメントを、全てのドキュメント作成者が作成できることが前提となる。また、開発工程が進むにつれて、作成したドキュメントの量は日増しに膨大になる。全てのドキュメントを全ての工程に引き継ぐというのは、あまりにも無意味であり、逆に業務混乱を起こしかねない。どの工程にどのドキュメントを渡せばよいのかなど、管理体制を含めて考えなければならない。

・　利用資源（人、金、物、情報）の決定と調達およびチームへの割り当て

いくら人がいてもシステム開発はできない。また、いくら開発予算があっても、システム開発はできない。新システムの開発に必要となる人員や予算、機器、情報などがうまくかねあうよう設定しなければばらない。特に、ハードウェアに関しては、開発したプログラムがその上（そのシステム全体の上）で正確に動くような機種を選定をしなければならない。近年、ネットワークシステムが急速に発展し、そして多角化・多様化してきたことから、その形態はより複雑なものとなってきている。また、ユーザの要求もとどまるところを知らない。ハードウェアは、ソフトウェアのように簡単に修正できるものではないので、代替え機種も含めて広く検討しておく必要がある。

また、よほどの大企業でない限り、自社内で全てのことを行うのは不可能である。情報が漏れるという心配はあるが、自社で対応するのは不可能または困難と判断された部分に関しては、外部のＳＥを含めた開発体制を考える必要もある（図Ⅱ-31）。

以上の結果を「開発計画書」としてとりまとめる。



図Ⅱ-31　外部ＳＥを含めた開発体制

（3）要求定義

基本計画以前に充分な調査・分析を行っていた場合、この工程は無視してもよい。ここでは、充分な調査・分析が行われていなかったものとして話を進める。なお、要求定義とは、あくまでユーザの要求のみを重視して新システムの方向性を決定する（「お客様」の満足を大前提に設計することが大切という理念：図Ⅱ-32）ものなので、システム開発者自らが新システム構築に関して独自に調査・分析した事柄については、あまり考慮せずに話を進める。

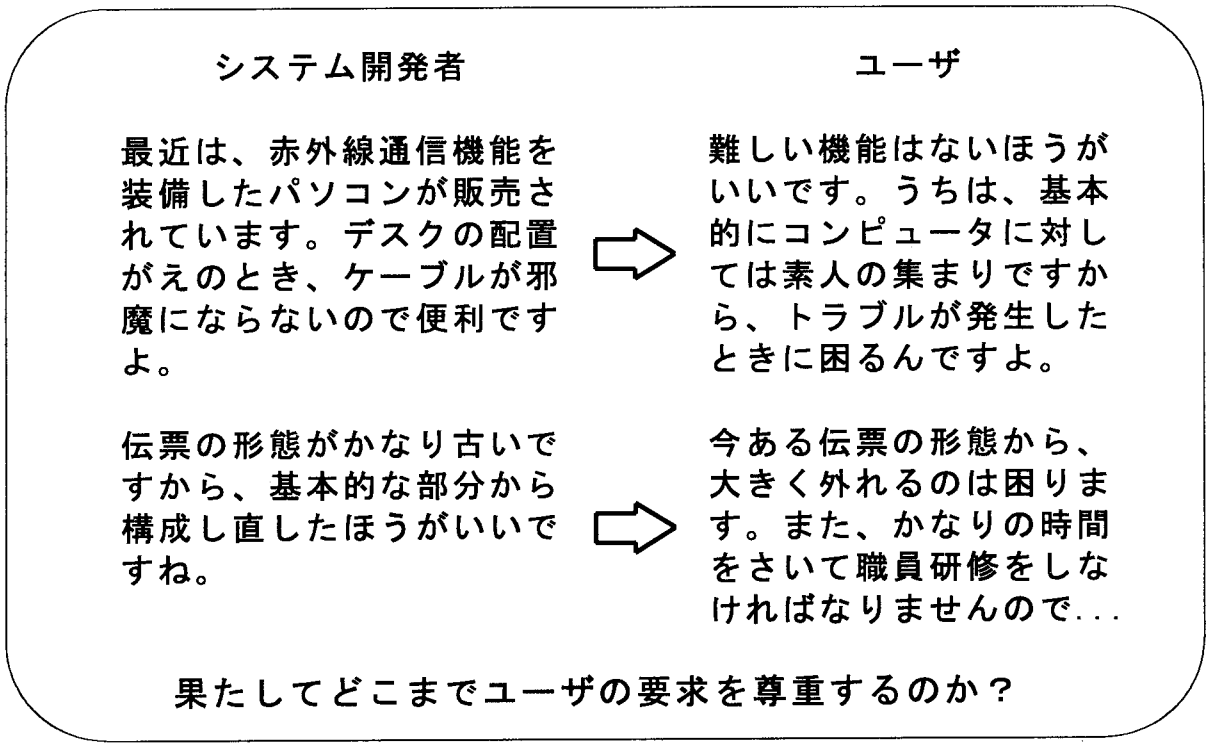
要求定義は、システム開発の基本となる対象システムの要求を定義する工程である。

目　的

システム化すべきユーザの要求を明確にする。ユーザの要求は、曖昧なものが多々あるので、この工程で曖昧さのない表現に置き換える。なお、ユーザの要求事項をユーザ要件といい、このユーザ要件を文書あるいは図表を用いて曖昧さのない形式で記述したものを要求定義という。

内　容

現行業務の調査・分析結果からきた問題点に対し、さらに新しいユーザ要件を加える。ここでは、開発するシステムで取り入れる機能、排除する機能、入出力に必要となる情報、



図Ⅱ-32　ユーザの要求重視

他のシステムとのインタフェース、性能目標、監査および統制上の要件などを明確にしていく。

要求定義における主な作業内容を以下に示す。

・　機能や性能、運用要件などのシステム全体に対する要求の定義

ユーザが求めている機能や性能、そして運用形態が実現できるシステムを定義する。ここで、特に重要となるのは機能である。いくら速いコンピュータを導入したからといっても、その機能が貧弱ならば意味がない。これは、開発するソフトウェアの仕様と密着した問題である。また、ハードウェアと関連が深いのは性能である。ソフトウェアで実現した性能が、本来の目標値に達していなくても、ハードウェア、すなわちコンピュータ自体やその周辺の機器で、ある程度まではカバーすることができる。ただし、これは最終的な手段であり、できればこのようなことは避けて通ることが望ましい。

・　ハードウェア、ソフトウェアに対する要求の明確化

ユーザがどのようなハードウェアやソフトウェアを欲しているのかを明確にする。例えば、コンピュータはＡ社製のＸＸ、ネッワーク関連の機器はＢ社製の○○でなければならないといった具体的な要求がでてくることもある。また、メーカーは問わないが、ホストには△△バイト以上のハードディスクを装備してほしいとか、プリンタはネットワーク対応のものでなければならないといった要求がでるかもしれない。これは、先に述べた他のシステムとのインタフェース（他の現有システムとの整合性）

の問題からであり、開発者は、よほどの理由がない限り、意見することは差し控えるべきである。また、ソフトウェアに関しては、ハードウェアのように具体的な要求はでてこない場合が多く、ただ漠然とこういう機能がほしいとか、こういったことはできないかという要求が多くを占める。システム開発者は、その漠然とした要求を的確に具体化していくこととなる。

以上の検討結果を「要求仕様書」としてとりまとめる。

5　外部設計

外部設計は、ユーザ側の見地におけるシステム設計である。つまり、ユーザとのインタフェースに重点においたシステム設計である。使用するコンピュータのことはあまり考えず、ユーザの業務の機能を中心に考えた設計を行う（業務を考えるということは、そこで処理されるデータや各業務間でのデータの流れも考えるということである。）。なお、コンピュータのことはあまり考えないと述べたが、例えば、超大規模なオンラインシステムであるとか、膨大なデータを瞬時にして解析しなければならないシステム（カオシックなモデルの解析や流体解析など）であるという場合は、まず、コンピュータ、すなわちハードウェアの選定を第一とする。この場合、業務だけを考えて外部設計を行うと、次の内部設計が不可能になることがある。

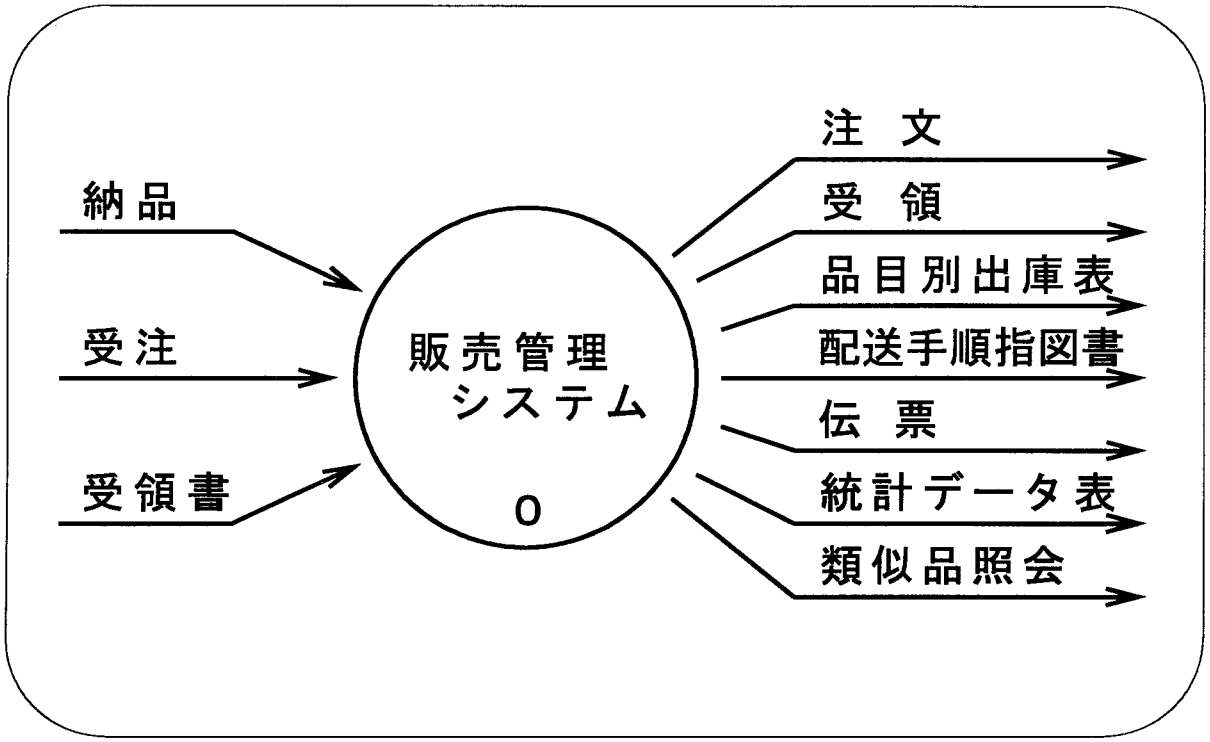
目　的

分析や基本設計の工程で明確となった各種の分析結果をもとに、新システムの機能を確定する。ここでは、使用するコンピュータのことはあまり意識せず、ユーザの要求している機能をいかにしてコンピュータで実現するかという考えのもとに設計を進める。

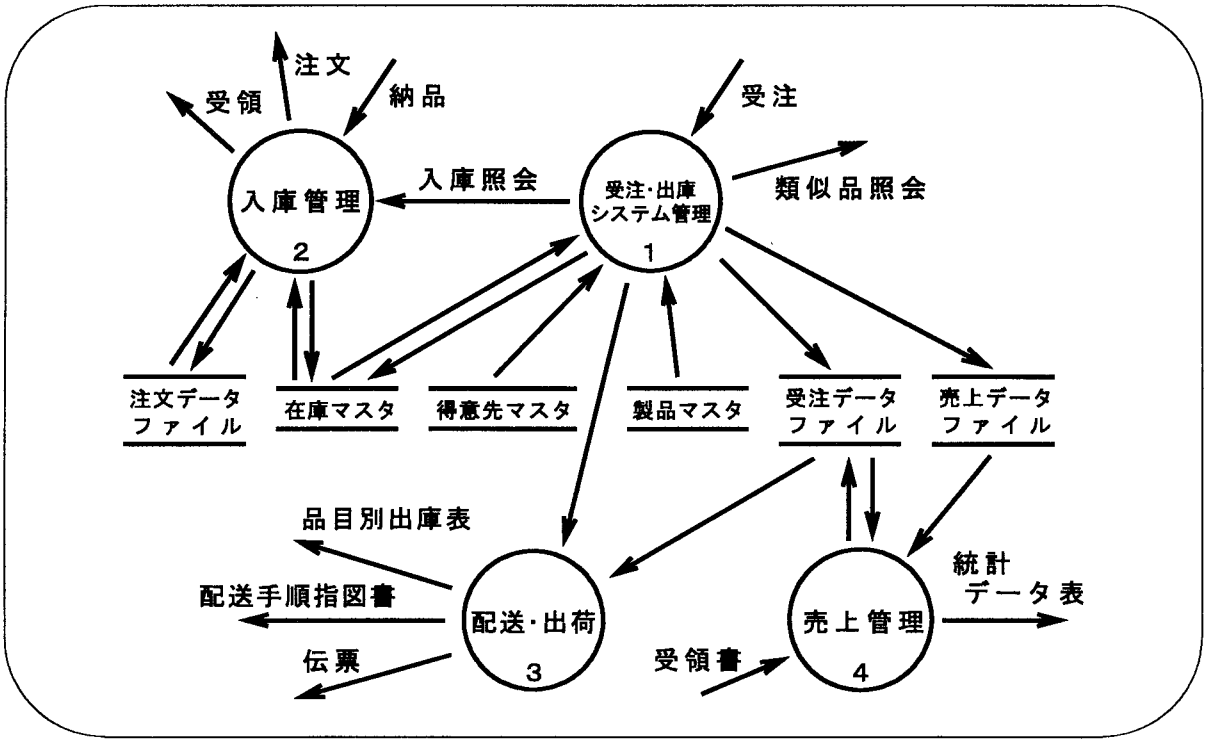
内　容

基本計画で作成した各種のドキュメントをもとに、業務の流れやそこでのデータの流れを整理し、最も効果的・効率的な処理手順を検討する。そして、その検討結果を業務フロー図としてとりまとめる。場合によっては、データコンテキストダイアグラム（ＤＣＤ：Data Context Diagram：図Ⅱ-33）およびデータフローダイアグラム（ＤＦＤ：Data Flow Diagram：図Ⅱ-34）を作成することも考えられる（ＤＦＤは５の(1)で詳しく説明する。）。理由は、こちらの方が後の工程を進めていく上で、より効果的になる場合が多いからである。また、いずれの場合にせよ、処理された出力情報をどこで、または、どの時点でユーザが必要としているのかを明確にする。この結果、データを入力するタイミングおよびリアルタイム処理にするのかバッチ処理にするのかが決定され、さらに入出力形式なども決定される。なお、データフローダイアグラムは、データフロー図やバブルチャートとも呼ばれる。

また、入力を簡略化するために、コード化が可能な項目の決定を行い、あわせてその体系の



図Ⅱ-33　ＤＣＤ（データコンテキストダイアグラム）



図Ⅱ-34　ＤＦＤ（データフローダイアグラム）

決定も行う。これをコード設計といい、コード設計に関しては、データの入力者全員が同じレベルで各項目を入力することができるという利点を考慮した上で設計する。

さらに、各データの関連性を分析した後に、新システムを構築する上で必要となるデータの選択と統合化を行う。これを論理データ設計という。論理データ設計においては、データの統合化という意味から、ファイル化すべきデータ群のグループ分けを行うこととなるが、実際にファイル化するかどうかは、後の行程である内部設計で行う。ただし、データフロー図を作成していた場合は、この限りではなく、もともとデータフローの概念がデータのファイル化やデータベース化（ここでは、比較的小さなファイルを「ファイル」と表現し、関連のあるファイル同士が集まって大規模なファイルを構成しているものを「データベース」と呼ぶ。）を考えた上のものなので、何がファイルやデータベースになるかが決定される。

外部設計における主な作業内容を以下に示す。

・　開発対象となっているシステムをサブシステムに分割

開発対象となっているシステムを整理し、機能別により細かくシステムをとらえたサブシステムに分割していく。通常、システムというものは、複雑極まりないものなので、全体を対象としたシステム開発をいきなり行うことは不可能に近い。したがって、システムを幾つかに分割し、その分割されたシステム、すなわちサブシステムの中で求めるべき機能を追求していく。なお、サブシステムに分割する際は、業務の流れという感覚より、むしろデータの流れという感覚をもとに分割していく。なぜならば、求めるべき機能を実現するために最も必要となるのはデータだからである。業務ばかりに気を取られ、データの流れを考慮するのを怠った場合、開発すべきシステムの中がファイルの山となり、それらを関連付けるために、後の工程で莫大な労力が必要となる。

このように、サブシステムに分割する際は、データの流れを重視した考え方が必要となる。この考え方をさらに押し進めたものが、データ中心アプローチ（ＤＯＡ：Data Oriented Approach）である。

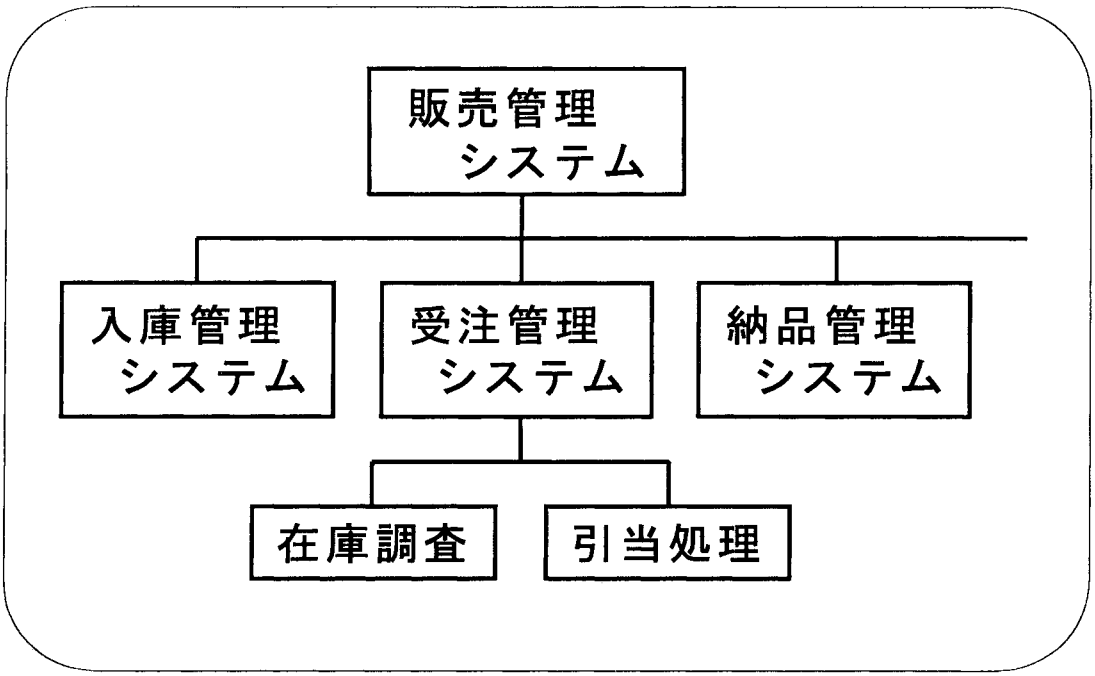
なお、サブシステムに分割する際は、一気に多数のサブシステムに分割するのではなく、何階層かに分けて分割していく（図Ⅱ-35）。このようにすることで、システムの保守・運用が楽になる。当然、内部設計やプログラム設計・プログラミングが簡単になることは言うまでもない。

・　各サブシステムの業務フロー図の作成

分割された各サブシステムの業務フロー図を作成する。ここでも先に述べたデータフロー図を作成しておくことが望ましい。データフロー図を作成することにより、サブシステムで本来行わなければならない業務がさらに明確になるからである。

・　入出力様式の設計、画面設計および画面遷移の決定

データの入出力様式を決定する。特に、金銭的な部分に関しては、開発したシステムがエラーを含まないように厳重なる注意が必要となる。例えば、単価が１円以下の桁を含む商品も世の中には多くあり、この場合、１円台からの入力しか行えないということになる



図Ⅱ-35　システムとサブシステムの関連

と、全く意味のないものを設計したということにもなりかねない。また、出力様式についても同様で、コンピュータの中では正確に計算できていたものの、出力で１円未満の桁が表示できなかったということになると、とんでもない問題である。ここでは、コンピュータに関する知識より、むしろ事務的な知識が必要となる。

次に、画面設計およびそこでの画面遷移であるが、これは、システムに用いるＯＳ（Operating System）に大きく依存する項目となる。Ｉでも述べたが、最近はＯＳとしてワークステーション上のUNIXやパソコン上のWindowsを採用したＧＵＩシステムが広く普及してきており、また業界での事実上の標準となっているので、特に問題はないように思われるが、導入コストの問題などから、現在でも大型汎用機やミニコンで独自の形態のシステムを構築しているところも多数ある。このような場合、ユーザインタフェースを向上させる目的から、グラフィクスをふんだんに用いた画面設計を行うと、後の工程であるプログラミングに膨大な時間が必要になるとか、また、最悪の場合、プログラミングが不可能になるということも生じかねない。外部設計では、コンピュータのことはあまり考えないと述べたが、あくまで常識的なレベルは考慮していなければならない。

また、画面遷移についても同様で、ユーザが使い易いからといって、このキーを押せばメニューが現れるとか、このキーを押せばメニューが閉じるとかを安易に決定してはならない。コンピュータの世界に関わらず、いかなるシステムにおいてもその業界標準は必ず存在する。ユーザが使用するシステムは一つとは限らないので、他のシステムとの整合性

と考えた上で画面遷移を決定する。

また、コンピュータ内で処理されたデータをどのような様式・形式で、プリントアウトするのかを決定しなければならない。画面上に表示されたもの全てをそのままプリントアウトするというようなことは、通常考えられない。どの帳票にどのようなデータが必要なのか、また、そのデータを帳票のどの位置に配置するのかなどを決定しなければならない。

・ コード設計

コード化可能なデータの選択およびコード体系を決定する。コードに関しては、JIS などで標準規格として定まっているものが多くあるので、他のシステムとの整合性の問題から、極力このような規格化されたコードを採用する（表Ⅱ-2(a)～(d)）。なお、企業の抱えている様々な理由から、規格化されたコードをそのまま使うことができないといった場合もある。このような場合は、企業が独自に用いるコードを、規格化されたコードに変換するためのコード変換に関する表などもあわせて作成することとなる。特に、近年は情報犯罪が頻繁に起きていることから、コードの暗号化を考えてシステム設計を行うことが重要となっており、また、ユーザもこのようなことを多く望んでいる。

また、コードには、一連番号コードや均等余量コード、区分コード、桁別分類コード、十進コードなど様々なものが存在するので、システム開発者は、これらの特徴を充分認識した上で利用用途に適合したものを選択決定する。

なお、コード化可能なデータは、あくまで具体的なものとし、抽象的な要素を含むものはコード化しない方がよい。なぜならば、抽象的な要素を含むものをコード化した場合、ユーザが、それがどの項目に対応するのかといった問題を抱えることとなり、逆にコード化したことがデメリットとなることがあるからである。

表Ⅱ-2(a)　一連番号コード（都道府県コード）

01	北海道	11	埼玉県	21	岐阜県	31	鳥取県	41	佐賀県
02	青森県	12	千葉県	22	静岡県	32	島根県	42	長崎県
03	岩手県	13	東京都	23	愛知県	33	岡山県	43	熊本県
04	宮城県	14	神奈川県	24	三重県	34	広島県	44	大分県
05	秋田県	15	新潟県	25	滋賀県	35	山口県	45	宮崎県
06	山形県	16	富山県	26	京都府	36	徳島県	46	鹿児島県
07	福島県	17	石川県	27	大阪府	37	香川県	47	沖縄県
08	茨城県	18	福井県	28	兵庫県	38	愛媛県		
09	栃木県	19	山梨県	29	奈良県	39	高知県		
10	群馬県	20	長野県	30	和歌山県	40	福岡県		

表Ⅱ-2(b)　区分コード（市町村コード）

101 千代田区	201 八王子市	（西多摩郡）
102 中央区	202 立川市	301 福生町
103 港　区	203 武蔵野市	302 羽村町
104 新宿区	204 三鷹市	303 瑞穂町
～	～	～
122 葛飾区	216 田無市	308 奥多摩町
123 江戸川区	217 保谷市	（南多摩郡）

表Ⅱ-2(c)　桁別分類コード（勘定科目コード）

1000 資産勘定	2000 負債勘定
1100 （流動資産）	2100 （流動負債）
1110 現　金	2110 短期借入金
1111 当座預金	2111 買掛金
1112 受取手形	2112 支払手形
～	～
1200 （固定資産）	2200 （固定負債）
1210 建　物	2210 社　債
1211 機械装置	2211 長期借入金
～	～

表Ⅱ-2(d)　十進コード（図書コード）

摘　要	コード
哲学	100
宗教	200
社会科学	300
法律	320
商法	325
会社法	3252
株式会社	32524
合名会社	32525

・　論理データ設計

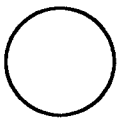
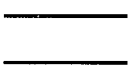
必要なデータの整理およびそのデータの統合化、データの関連性の分析、ファイル化すべきデータ群のグループ分けを行う。すなわち、論理データ設計では、データの関連性の分析結果から、どのデータを一つのグループとして管理するのかを明確にする。この一つのまとまりが、一つのファイルになる可能性があり、また、そのファイルの集まりが巨大なデータベースになる可能性もある。なお、実際にファイルにするかどうか、データベースにするかどうかの決定は内部設計で行う。

以上の検討結果および設計結果を「外部設計書」としてとりまとめる。また、外部設計におけるレビューの結果を「外部設計レビュー報告書」としてまとめる。

（1）ＤＦＤを用いた外部設計

外部設計において最も重要となる作業は、開発対象となっているシステムをサブシステムに分割することである。従来は、業務（処理）を中心に考えてシステムを分割していたが、最近では、情報（データ）を中心に考えて分割することが多くなってきている。このとき有用となるのがＤＦＤである。ＤＦＤは、表Ⅱ-３に示すように、たった四つの図式記号（ソース／シンク、データフロー、プロセス、データストア）で表すことができ、そのため、システム開発について詳しい知識を有していないユーザにも簡単に理解できるという利点がある。したがって、分析結果および新システムの構成をユーザ側や開発スタッフに提示するとき、非常に説明しやすいという利点をもっている。

表Ⅱ-３　ＤＦＤで用いる図記号

記　号	名　称	意　　味
	ソース シンク	ソース：データの発注減を示す。 シンク：データの行先（吸収先）を示す。
	データフロー	各処理およびソース／シンク、データストア間のデータの流れを示す。
	プロセス	処理（プロセス）を示す。
	データストア	データが保存（蓄積）される場所を示す。

イ　ソース／シンク

ソースとはデータの入力源のことであり、システムの処理対象となっている大本のデータの発生源を示す。シンクとは、データの出力先、すなわち最終的なデータの行き先・吸収先のことであり、システムが処理したデータの受け渡し先を示す。

ソースおよびシンクは、いずれも長方形または正方形で示し、その中に対象となる組織名や人名を記入する。

なお、入力源および出力先が一つずつとは限らないので、ソース／シンク共に複数個存在してもよい。

ロ　データフロー

データフローとはデータの流れのことであり、プロセス間のデータの流れを矢印を用いて示す。それぞれのデータフローには、矢印の近傍に個別のデータ名を記入する。

ハ　プロセス

プロセスとはデータの処理のことであり、入力データを出力データに変換するための処理を示す。それぞれのプロセスは、円（バブル）を用いて示し、その中に処理概要を記入する。

ニ　データストア

データストアとはデータの保存（蓄積）場所のことであり、具体的にはデータを記憶するファイルを示す。データには、入力された後、処理（変換）されて出力されるといった形態のものだけではなく、各プロセスが共通で使用するために保存しておくデータや、処理時間にずれが生じた場合に一時的に記憶しておくデータ、すなわち一時記憶データなどがある。これらのデータをデータストアとして表す。データストアは、２本の直線を用いて示し、その間にファイル名を記入する。

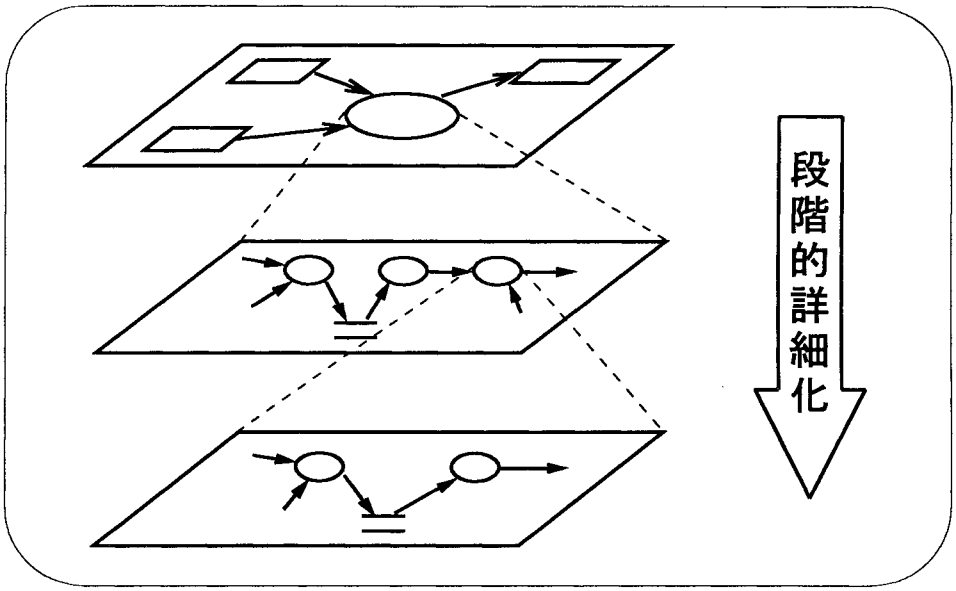
ホ　作成手順

ＤＦＤを作成する際、いきなり処理の明確化やそこでのデータ受け渡しを考えてはならない。簡単な部分から複雑な部分へと詳細化していく構造化の概念に従うことが基本となる（図Ⅱ-36）。

ＤＦＤの作成手順を以下に示す。

(イ)　ソース／シンクの設定

開発の対象となっているシステムにおいて、そのデータの提供元となっている入力源、すなわちソースとデータの最終的な受け手となっている出力先、すなわちシンクを決定する。



図Ⅱ-36　ＤＦＤの段階的詳細化

(ﾛ)　ＤＣＤの作成

データの入出力が設定されたら、次にその変換工程を決定する。ここでは、入力源から発生した入力データを、それぞれに対応する出力データに変換するための処理を一つのバブルで表現する。すなわち、開発するシステムにどのようなデータが必要であり、どのようなデータを出力すればよいのかということを明確にする。このとき、入力データと出力データの整合性を考えることとなる。すなわち、これだけの入力データがあれば、この出力データが得られるという発想である。システム内部での処理はあまり考えずに、必要となるデータに重点をおく。このようにして作成された図をＤＣＤとよぶ。

(ﾊ)　バブルの階層化（詳細化）

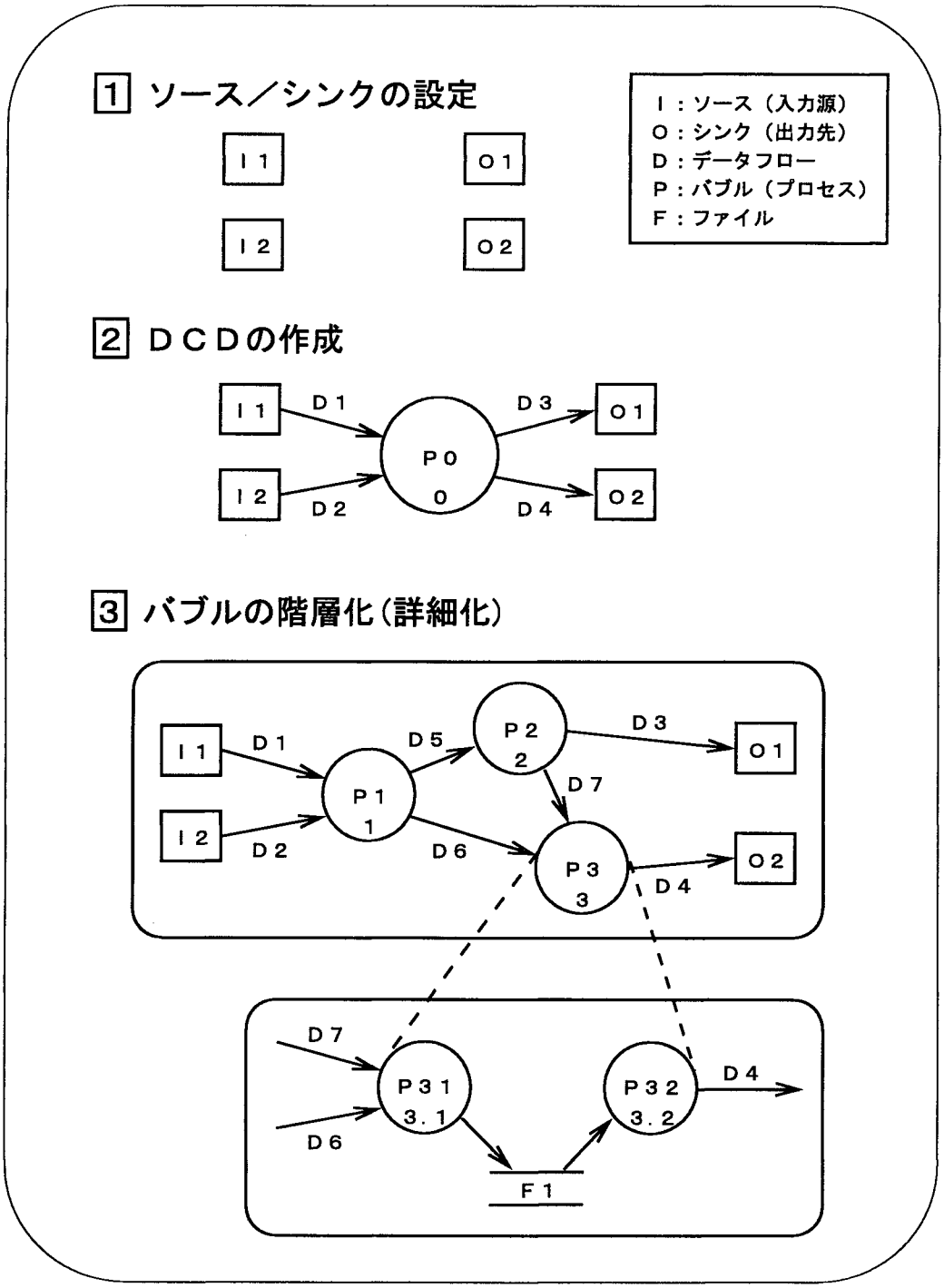
ＤＣＤのバブルを幾つかのより小さい機能に分割する。このとき作成されたＤＦＤをレベル０ダイアグラムといい、その中の各バブルはサブシステムに対応する。さらに、レベル０ダイアグラムの各バブルを、より小さい機能に分割したのもをレベル１ダイアグラムとよぶ。このように、段階的詳細化技法を用いて次のレベルのＤＦＤを作成していく。

このとき、レベル１のＤＦＤは、レベル０のＤＦＤの主要プロセスを構成する詳細な幾つかの子プロセスに分割したものになっている。また、データフローについても、レベル０のＤＦＤから基本的に引き継ぐこととなる。

同様にして、順次下位レベルのＤＦＤを記述していき、プロセスへの入出力データフローが項目単位で表され、プロセスが単一機能となるところまでレベルを下げていく。すなわち、最下位レベルのＤＦＤのバブルは、一つのプログラムを示すこととなる。なお、

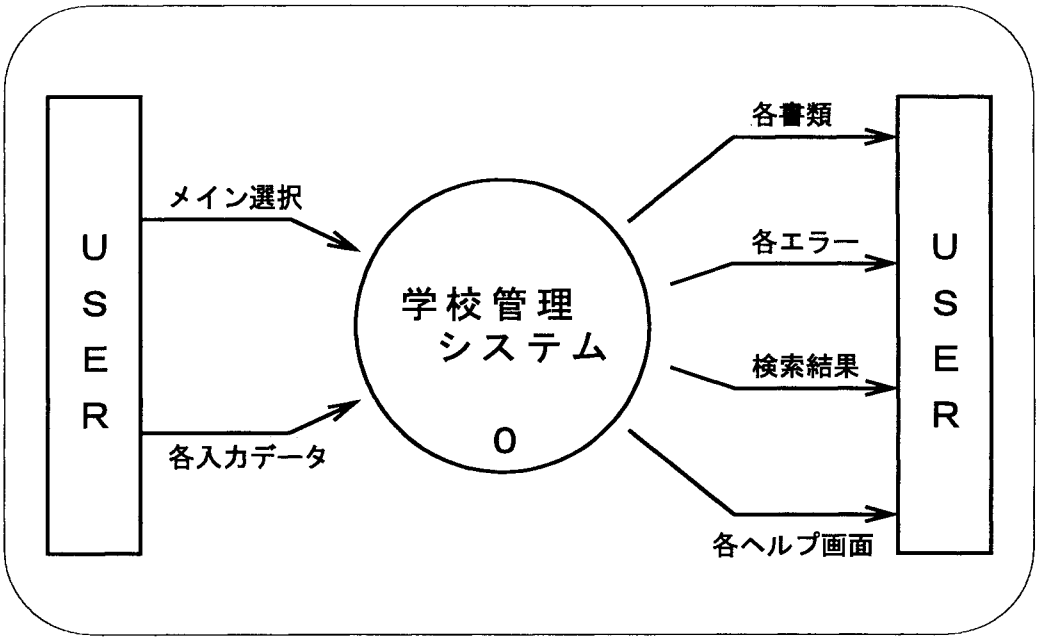
外部設計においては、サブシステムまでの分割でとめておき、次の内部設計でより詳細なプログラム分割を行う。

以上、(イ)～(ハ)によりＤＦＤが詳細化され、システムが明確化されていく（図Ⅱ-37）。

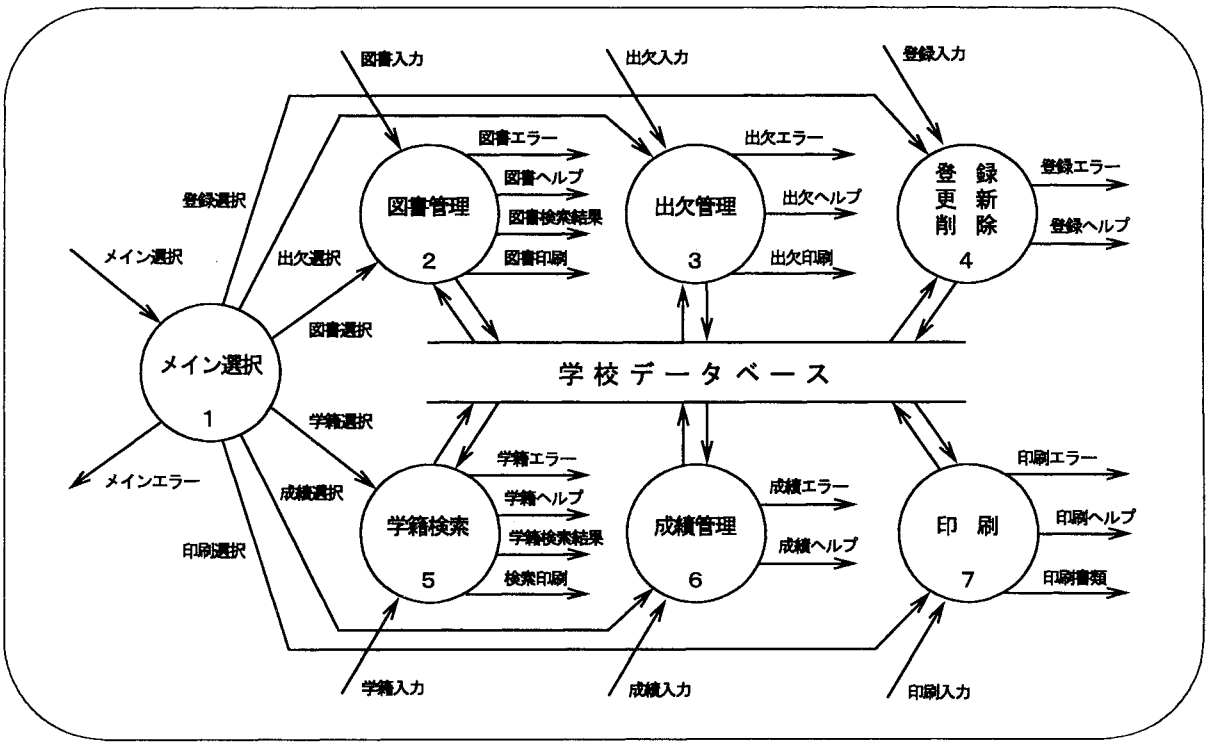


図Ⅱ-37　ＤＦＤの作成手順

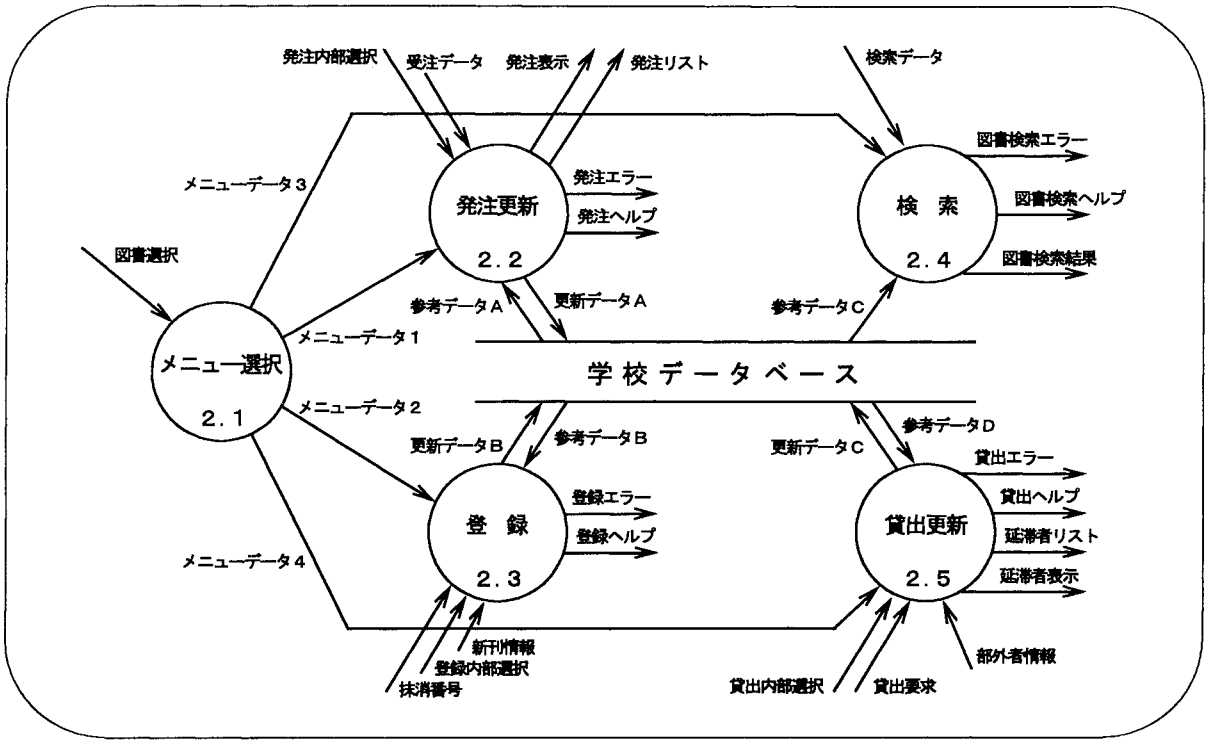
実際のシステム開発の用いたＤＦＤの一例を図Ⅱ-38に示す。これは、平成６年度に開発された、福山職業能力開発短期大学校の学校管理システムおよび図書管理サブシステムのＤＦＤである。本システムは、プロのＳＥによって開発されたものではなく、学生が開発したものである。



図Ⅱ-38(a)　学校管理システムのＤＣＤ



図Ⅱ-38(b)　学校管理システムのＤＦＤ（レベル０ダイアグラム）



図Ⅱ-38(c)　図書管理サブシステムのＤＦＤ（レベル１ダイアグラム）

ＤＦＤの適用範囲は非常に広範であり、何らかのデータとそれを処理するためのプロセスさえあれば、どのようなモデルでも表現することが可能である。例えば、システム開発工程をＤＦＤで表すと図Ⅱ-39のようになる。

このように、外部設計や後に示す内部設計において非常に有用となるＤＦＤであるが、時間の経過や事象の発生による状態変化を表すことができない点が唯一の欠点となる。

以下にＤＦＤ作成時の注意点を示す。

・　データの明確化

　全てのデータフローに、そのデータが何であるかが具体的に分かるようなデータ名をつける。

・　図の見やすさ

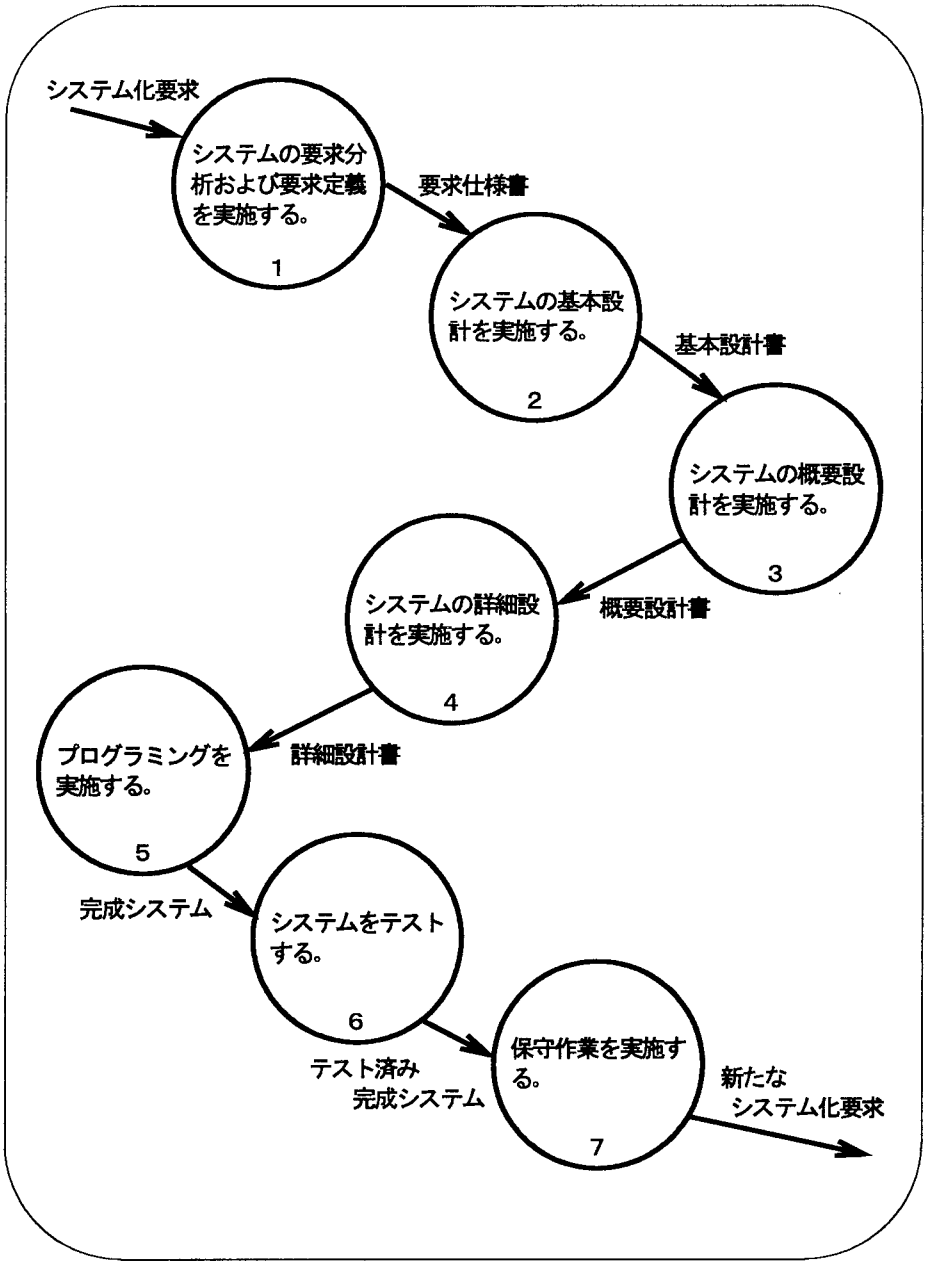
　データフロー同士は、できる限り交差させないほうがよい。

・　タイミングは記述しない

　データフローにタイミングを示すものは記述しない。例えば、前の処理が終了してから何分後とか、データの更新は３日おきであるというような処理のタイミングは記述しない。

・　正常処理を中心に考える

　記述の際は、正常処理を中心に考えてよい。エラー処理や例外処理に関しては、バブルやデータフローで必ず示す必要はない。



図Ⅱ-39　ＤＦＤで表したシステム開発工程

・　バブルの番号付け

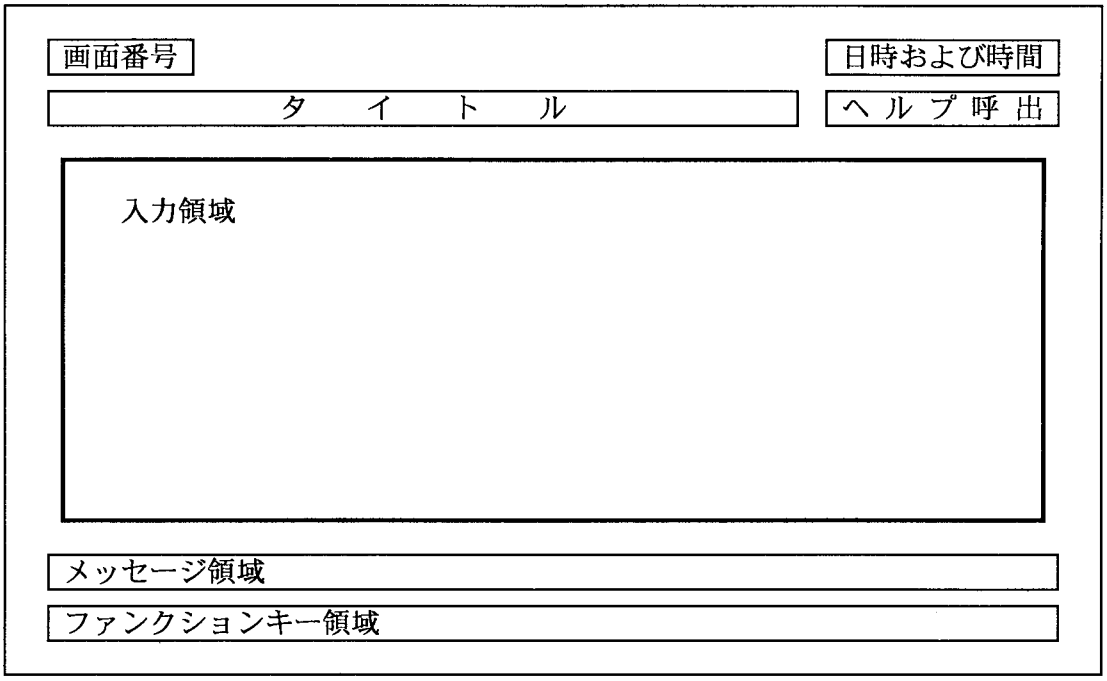
各バブルには、そのバブルが唯一識別できる番号を付加する。通常、データコンテキストダイアグラムのバブルは０、次のレベル０ダイアグラムには、０、１、２、･･･ と番号を付加する。さらに、１のバブルを詳細化した次のレベルの各バブルには、1.1、1.2、1.3、･････ と番号を付加し、同様に２のバブルを詳細化したものについては、2.1、2.2、2.3、･････ と番号を付加する。

（2）画面設計・画面遷移および帳票設計

ハードウェアの高速化およびネットワーク機能の充実から、最近では、ほとんどのシステムがオンラインリアルタイム処理となってきている。オンラインリアルタイム処理では、ディスプレイ画面とのやりとりが入出力処理の中心となり、このため、画面の移り変わりが処理の進行を表すといっても過言ではない。したがって、入力のしやすさや画面の遷移状態が、処理の良し悪しを決める重要な要素となっている。

ここでは、ヒューマンインタフェースを意識した設計が重要となる。

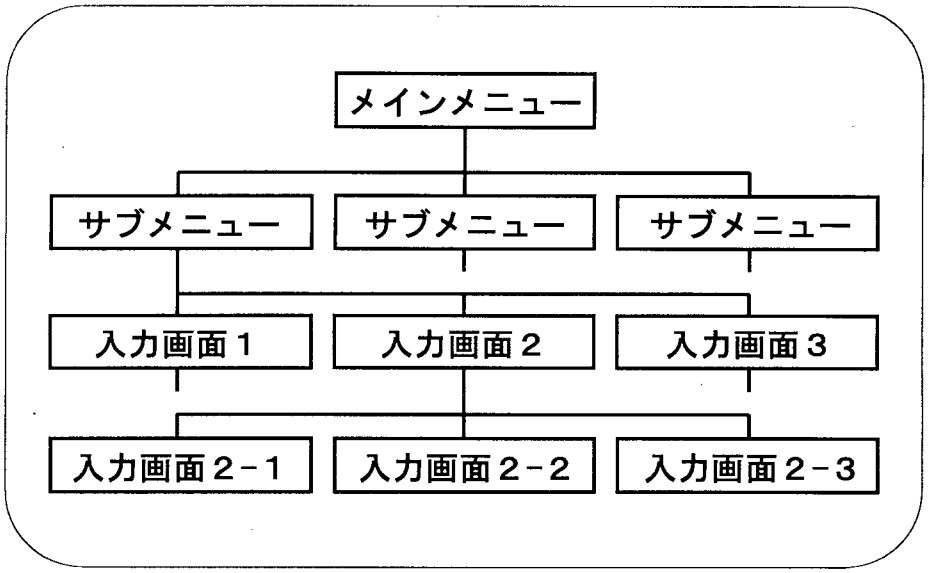
まず、最初に行うのが画面のレイアウトの統一である。同一システム内で、画面が変わるごとに画面レイアウトが変わるのでは、ユーザが入力しにくいという問題が発生する。そこで、できる限り画面のレイアウトを統一し、画面が様々に変化した場合でも同一条件でユーザが入力しやすいように考慮する（図Ⅱ-40）。



図Ⅱ-40　画面レイアウトの一例

次に検討するのが画面遷移である。一般的なシステムでは、まずメインメニューがあり、そこからサブメニューが呼び出され、その後、それぞれの入力画面に処理が移る（図Ⅱ-41）。

しかし、ユーザは、オペレーションに慣れてくると、入力画面に移るのに一旦サブメニューやその入力画面の前の画面を経由するのが面倒となる。このような場合は、ファンクションキーなどを用いて、メインメニューから直接入力画面に移れるような機能を付加することとなる。また、同様に、入力画面からメインメニューに直接戻れるような機能も付加した方がよい。



図Ⅱ-41　画面遷移の一例

なお、画面設計においては、先に示したＤＦＤを参照すると、その表示項目が明確となり、また、画面遷移においても、その表示の遷移状態がより明確となる。

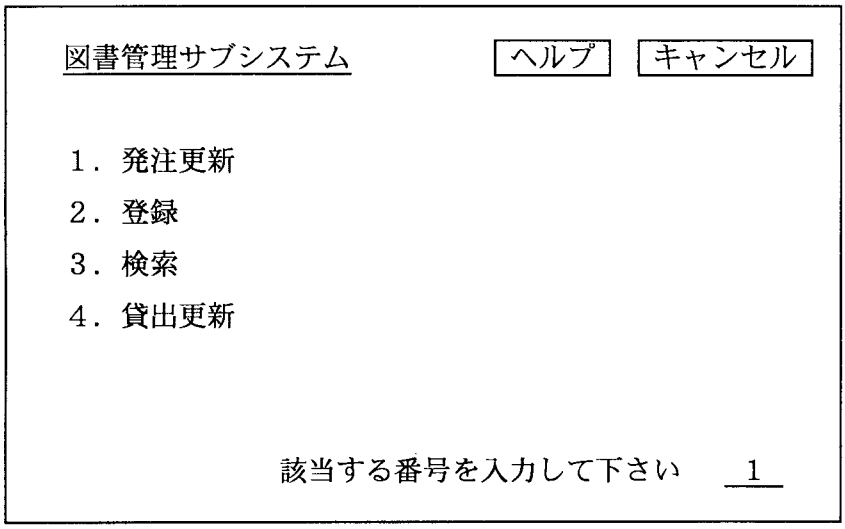
図Ⅱ-38で示した学校管理システムおよび図書管理サブシステムの画面設計の一例を、図Ⅱ-42に示す。

福山職業能力開発短期大学校　学校管理システム

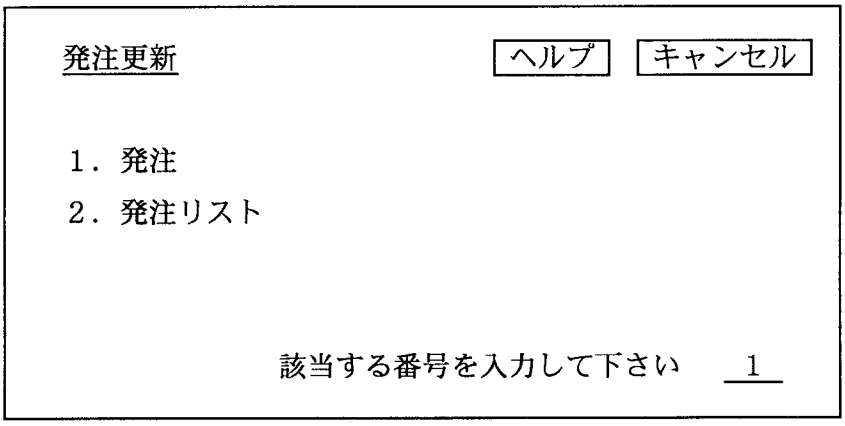
１．図書管理
２．出欠管理
３．学籍および成績の登録・更新・削除
４．学籍検索
５．成績処理
６．印刷
７．終了

該当する番号を入力して下さい　1

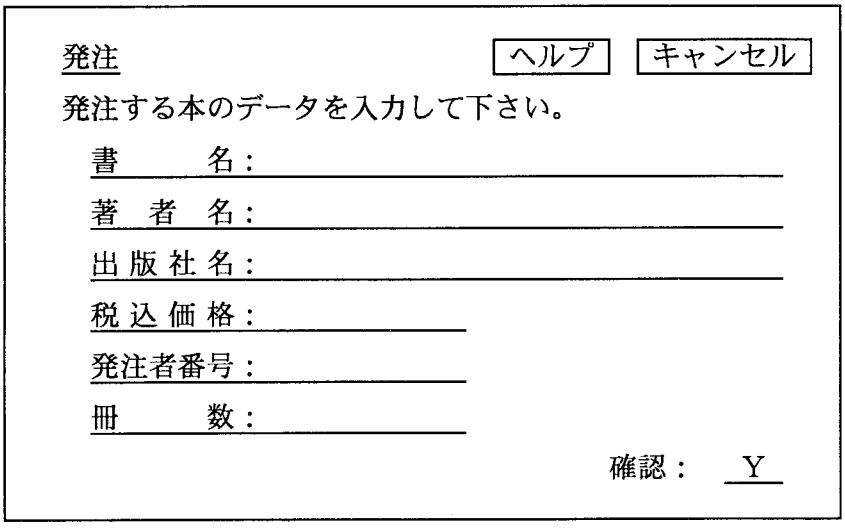
図Ⅱ-42(a)　メイメニュー選択画面



図Ⅱ-42(b)　図書管理メニュー選択画面



図Ⅱ-42(c)　発注更新選択画面



図Ⅱ-42(d)　発注入力画面

帳票設計においては、ユーザが必要としている情報をいかに分かりやすく正確に提供するかということを基本に設計する。

出力内容については、項目やレイアウトは勿論のことその属性や桁数をユーザーと打ち合わせの上、決定することとなる。出力桁数において、桁落ちや桁あふれが生ずることは論外であるが、各項目について、左右どちらに空白を設けるのか、それとも中央に寄せるのかなどの細かい設定も必要となる。

図Ⅱ-38で示した図書管理サブシステムの帳票レイアウトの一例を表Ⅱ-4に示す。

表Ⅱ-4　帳票レイアウトの一例（図書管理サブシステム：図書発注リスト）

図書発注リスト

NNNN 年 NN 月 NN日　　　No. NNN／NNN

発注者番号	書　名	著者名	出版社名	冊数	単価	金額
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
A-NNNN	KKKKKKKKKKKKKKKKKKKKKKKKKKKKKK	KKKKKKKKKKKKKKK	KKKKKKKKKKKKK	NNN	NNNNNN	NNNNNNN
				合　計		NNNNNNN

Ａ：アルファベット
Ｎ：数字
Ｋ：漢字

6　内部設計

内部設計は、システム開発者側の見地におけるシステム設計である。つまり、コンピュータの機種・性能などの仕様を考え、システムを実現するための手順を具体的に設計していくものである。ここでは、システムを構築していく上で必要となる機能をプログラムに分割し、プログラムの処理手順や各プログラム間の関連を検討し、その結果をもとに設計を進めていく。

目　的

開発するシステムに必要となる機能全てを整理し、各機能をプログラムに分割する。さらに、分割された各プログラムの処理を明確にし、各プログラム間のインタフェースを決定する。

内　容

外部設計で作成したシステムの概観をもとに、実際のシステム構成要素（システム構成機器）との対応関係を考え、より詳細な設計へと発展させていく。ここでいうシステム構成要素とは、非常に広範なものであり、例えば配送業務に必要となるトラックなどもその要素の一つとなる。配送手順を導出するプログラムが作成できても、実際に物を運ぶトラックの台数やその積載量に問題があれば、システムを構築した意味はない。このようなことを充分考慮した上で、正確かつ効果・効率的に処理するための方法を考える。

ここでは、まず、外部設計の概要的な機能をいかにしてコンピュータで実現するかを考え、その上で、システムの各機能をプログラム単位に機能分割していく。このとき、システムの各機能は、トップダウン的に細分化されていく。これを構造化設計（ＳＤ：Structured Design）といい、この作業によって処理の詳細がより明確になっていく。

次に、外部設計で行った論理データ設計の結果から、具体的なファイルの編成や、それを格納するディスクなどの媒体、またファイル自体のレイアウトを決定する。これを論理データ設計に対して、物理データ設計という。また、これは、具体的にファイルを決定することからファイル設計ともいう。

さらに、外部設計で作成した業務フロー図をもとに、入出力の手順と、その正確さや効率を考えた処理方式を設計する。これを処理の詳細設計といい、詳細設計を行うためには、各機能間の入出力が具体的に決定されている必要がある。したがって、処理の詳細設計と同時に、入出力の詳細設計も行う必要がある。

内部設計における主な作業内容を以下に示す。

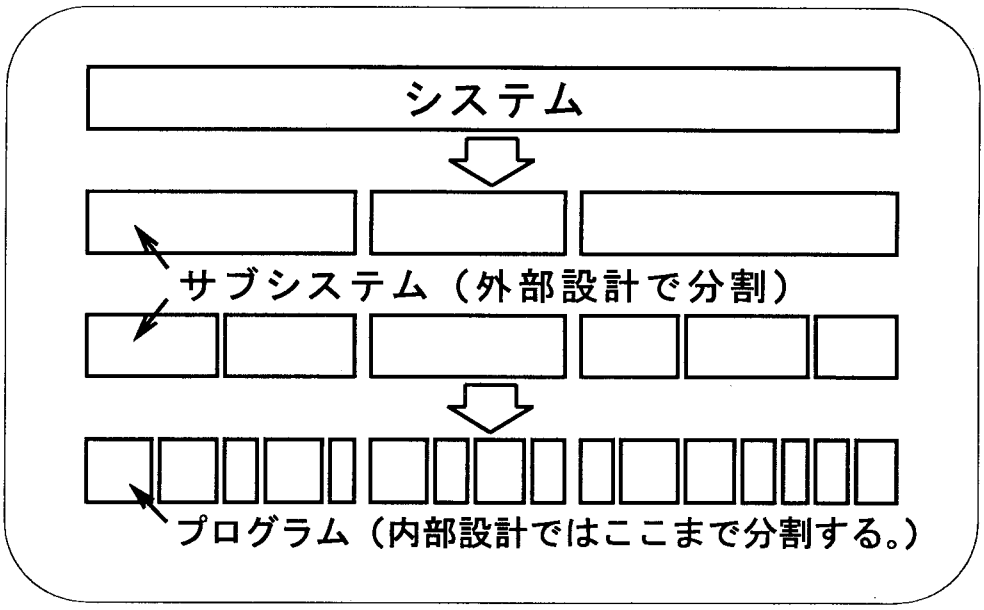
- システムの機能分割（構造化設計）

  外部設計において分割されたサブシステムを、さらに小さな機能へと分割していく。

  内部設計におけるシステム分割の最終点は、個々のプログラムとなる（図Ⅱ-43）。

- 物理データ設計（ファイル設計）

  実際に使用するファイルの特性について検討し、その結果から、ファイルの編成方法や



図Ⅱ-43　システムの機能分割

記憶媒体、レコードレイアウトなどを仮に決定する。次に、アクセス時間と容量の大まかな値を計算し、その結果、システムに問題がなければ上記の点が決定する。なお、アクセス時間と容量の算出においては、規定値ぎりぎりのものではなく、安全係数を充分考慮した上で算出することが望ましい。

・　処理および入出力の詳細設計

機能ごとに分割されたプログラムの処理内容および入出力を決定する。また、各プログラム間のインタフェースを決定する。

以上の検討結果および設計結果を「内部設計書」としてとりまとめる。また、内部設計におけるレビューの結果を「内部設計レビュー報告書」としてまとめる。

構造化設計

ハードウェア・ソフトウェア全体を対象とした情報処理産業は、過去数年間、加速度的な勢いで発展してきた。そのような中で、様々なハードウェアが標準化されていくこととなるが、対照的にソフトウェアであるプログラムを開発するための技法の標準化は大きな遅れをとっていた。これは、Ⅰで述べたように、ソフトウェアがハードウェアのように目に見えるものではないという原因からくるものである。これをもう少し詳細に分析すると、ソフトウェアを作成するためのプログラミング作業自体が創造活動の典型的なものとなっており、個人の哲学や思想が大きく反映されるということである。したがって、これらの要素を統一化するための標準化という枠を設けることは、当然のことながらプログラムから大きな抵抗を受けることなる。

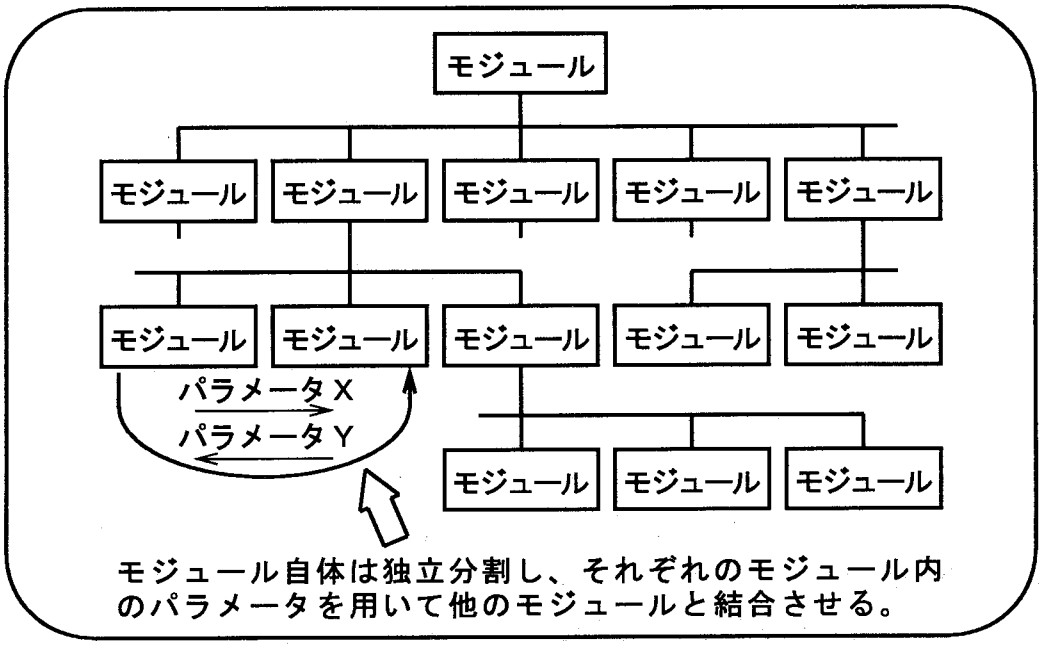
また、現実問題として、システムが稼働しさえすれば、ソフトウェアが標準化に基づいて作成されていたかどうかや、各種のプログラム仕様書が標準仕様に基づいて作成されていたかどうかなどは、あくまで二次的なものとしてしか扱われていなかった。しかしながら、今日におけるシステムの多種・多様・多機能化に対して、従来の職人芸的プログラミングは通用しなくなり、それとともにプログラムの標準化や品質管理の重要性が認識され、システム開発工程の中にこれらの要素を取り入れることが常識となってきた。

今日のソフトウェア開発は、処理内容が複雑化、すなわちユーザのニーズが多様化し、それに伴いプログラムサイズも巨大化する一方となっている。そのような中で、開発能力に差がある大勢のプログラマを組織しながら、いかに効果・効率的にソフトウェア開発を推進できるかが最大のテーマとなっている。このためには、開発技法の標準化が絶対条件となり、この基本理念に基づいて考え出されたのが構造化設計技法である。構造化設計は、元来、プログラム開発の工程で発展してきたものであるが、トップダウンアプローチの概念およびその手法によって、内部設計の工程でも多く採用されつつある。

7　プログラム設計

プログラム設計とは、内部設計で作成されたプログラム仕様をもとに、各プログラムの内部構造を設計するものである。プログラム設計では、プログラムの属性を決定した後、モジュールという機能単位に分割し、さらに各モジュール間のインタフェースを決定する（図Ⅱ-44）。すなわち、各プログラム内の構造化設計を行うこととなる。

なお、プログラム設計に関する詳しい内容は、Ⅳにおいて別に説明するので、ここではその概要を示す。



図Ⅱ-44　階層構造化

目　的

内部設計においてプログラム単位に分割された機能を実現するための仕様を決定する。ここでは、分割された機能単位のプログラムを階層的により詳細な機能単位のモジュールへと分割していく。

内　容

プログラム設計は、プログラム構造設計とモジュール設計の二つに大別される。プログラム構造設計では、データの流れや構造に着目して機能分割を行う。また、モジュール設計では、分割された各機能の仕様決定を行う。

内部設計は、実際のプログラミングを意識していない設計であるが、プログラム設計では、対象となるプログラミング言語で実際にプログラミングするために、どう記述すればよいかを意識した設計となる。

プログラム設計における主な作業内容を以下に示す。

・　プログラム分割およびそれに伴うモジュール階層の決定

　個々のプログラムをさらに詳細な機能単位のモジュールに分割する。このとき、重要となるのが、1モジュールの大きさとモジュール階層の深さである。一般的に、1モジュールの大きさは 300 ステップから 500 ステップ程度であり、モジュール階層の深さは 4 モジュール以下である。

・　モジュール機能の仕様決定

　分割されたモジュールの機能仕様を決定する。これは、先に示したプログラム分割およびそれに伴うモジュール階層の決定と並列して行う。

・　モジュール間インタフェースの決定

　分割されたモジュール間のデータの受け渡し方法、すなわちインタフェースを決定する。ここでは、制御構造に重点をおくとモジュールの独立性が損なわれやすくなるので、データ構造に重点をおいてインタフェースを決定していく。

・　結合テスト計画の立案

　結合テストの計画を立案し、テストケースの設計を行う。結合テストは、次のプログラミングの工程で作成されるモジュールを結合して、プログラムが仕様通りに動作するかどうかをテストするものである。

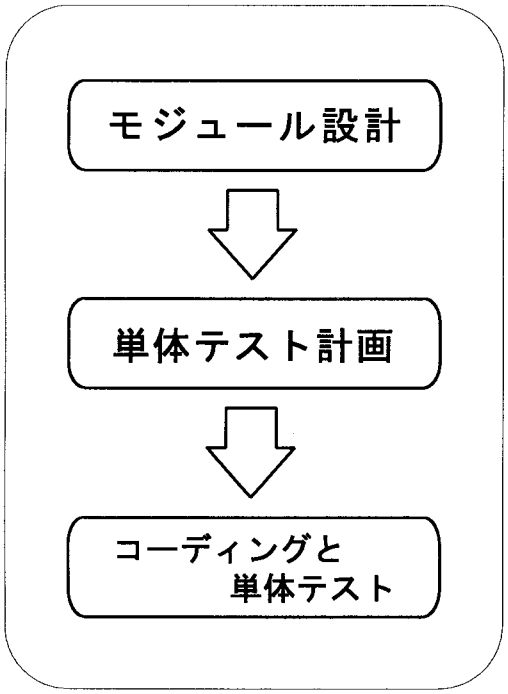
以上の検討結果および設計結果を「プログラム設計書」としてとりまとめる。また、プログラム設計におけるレビューの結果を「プログラム設計レビュー報告書」としてまとめる。さらに、結合テスト計画については、「結合テスト計画書」としてとりまとめる。

8　プログラミング

プログラミングとは、プログラム設計の結果をもとに、実際にモジュールのコーディングとテストを行うものである。

プログラミングは、図Ⅱ-45に示すように、さらに三つの工程に分けられ、それぞれの工程で表Ⅱ-5に示すドキュメントが作成される。

なお、プログラミングに関する詳しい内容は、Ⅴ章において別に説明するので、ここではその概要を示す。



図Ⅱ-45　プログラミングの手順

表Ⅱ-5　プログラミングで作成するドキュメント

工　　程	作成するドキュメント
モジュール設計	モジュール設計書 モジュール設計レビュー報告書
単体テスト計画	単体テスト計画書
コーディングと単体テスト	単体テスト報告書 ソースプログラムリスト

（1）モジュール設計

モジュール設計において最も重要となるのが、論理を構造化することである。具体的には、順次・選択・繰り返しという三つの制御構造だけで、論理を組み立てていく。ここでは、論理の構造化という理念から、goto文を用いないプログラムを作成しなければならない（プロのプログラマばかりならば、goto文がいくらあろうとも全く問題としないが、現実には、そのレベルまで達してしないプログラマがほとんどである。したがって、誰にでも分かりやすいプログラムを作成するため、goto文を用いないようにする。）。

目　的

モジュール設計における最大の目的は、論理の構造化されたプログラムを作成することである。すなわち、誰にでも分かりやすいプログラム、保守のしやすいプログラムを作成することである。

内　容

プログラム設計の結果をもとに、モジュールの内部構造を詳細化し、「モジュール設計書」を作成する。このとき、基本的には、開発言語に依存しないように作成すべきであるが、現実には言語自体に大きな差があるので、あらかじめ開発言語を決めておいた方がよい。

また、論理の構造化においては、データ構造と制御構造を決定しなければならない。ここでは、データ構造に重点を置いて設計することで、制御構造を簡単化することができる。すなわち、論理的に明解なモジュールが作成できる。

モジュール設計における主な作業内容を以下に示す。

- データ構造の決定
- 制御構造の決定
- 論理の組み立て

以上の検討結果および設計結果を「モジュール設計書」としてとりまとめる。また、モジュール設計におけるレビューの結果を「モジュール設計レビュー報告書」としてまとめる。

（2）単体テスト計画

単体テストとは、モジュール単位で行うテストのことである。プログラムは単体で動作可能であるが、モジュールは単体では動作しないことが多い（バッチ処理では、プログラムとモジュールが一致する場合が多いので、その限りではない。）。したがって、モジュール単体をテストするために、テスト方法の選定やテストデータの作成が必要となる。

目　的

単体テストの準備を行う。

内　容

単体テストを行うためのテストスケジュールおよびテストデータを作成する。また、モジュール単体では動作しない場合が多いので、テストツールを作成する。なお、テストツールの作成に多くの時間をかけることは、本来のプログラミングの主旨から外れることとなるので、できる限り現存の汎用的なテストツールを用いるようにする。

単体テスト計画における主な作業内容を以下に示す。

・　テストスケジュールの作成
・　テストデータの作成
・　テストツールの作成もしくは現存テストツールの選択

以上の結果を「単体テスト計画書」としてとりまとめる。

（3）コーディングと単体テスト

モジュールやプログラムの論理構造に基づいて、実際にプログラムを作成することをコーディングという。コーディングを行うためには、当然、特定のプログラミング言語の仕様を知っている必要がある。なお、コーディングしたモジュールやプログラムは、ソースプログラムまたは原始プログラムとよばれる。

目　的

モジュール設計書に基づいてコーディングを行う。さらに、机上デバッグと単体テストを繰り返し行い、モジュールの誤りを修正する。

なお、最近では、生産性の問題から、机上デバッグを行うことは少なくなってきており、実際にモジュール設計書にかかれた仕様を満足するまでコンパイルと修正をコンピュータ上のみで行うマシンデバッグが多くなってきている。

内　容

できる限り標準的なプログラムを作成するために、各企業あるいはプロジェクトごとに決められたコーディング規約に基づいて、コーディングを行う。ここでは、注釈（コメント）や字下げ（インデント）、ネーミング（名前付け）などの記述スタイルを統一する。

また、作成したプログラムをコンパイルし、動作チェックを行う。このとき、机上デバッグやマシンデバッグを併用し、論理の誤りを修正していく。

コーディングと単体テストにおける主な作業内容を以下に示す。

・　コーディング

　ソースプログラムを作成する。

・　机上デバッグもしくはマシンデバッグ

　机上デバッグとは、プログラムをコンピュータの上で実行させながらデバッグするのではなく、机の上（紙の上）でデバッグする方法である。机上デバックでは、テストデ

ータをソースプログラムのロジックの流れにしたがって変換させながら、最終的な出力結果を導き出し、それが与えられた仕様通りかどうかを検証し、修正していくこととなる。この作業は全て机上で行われることとなる。

それに対して、マシンデバッグは、実際にコンピュータを用いてデバッグする方法である。以前は、マシンデバッグを多く行うと、ハードウェアの性能の問題から、資源や時間の無駄な消費を招くことが多かったが、現在では、ハードウェアの高速化・高機能化およびソフトウェアであるデバッガの高機能化から、上記のような問題はなくなり、むしろマシンデバッグを行う場合が多くなっている。

・ 単体テスト

モジュール単体のテストを行う。

一般に、テストとデバッグは連動して行われる。すなわち、テストの結果が求めているものと異なった場合にデバッグを行い、その結果を再検証するために、再度テストを行い、最終的に仕様書の基準を満たすところまで反復してつきつめていくこととなる。

以上の結果を「単体テスト報告書」としてとりまとめる。また、単体テスト終了時の最新の「ソースプログラムリスト」を出力しておく。

9 テスト

テストとは、作成したプログラムが仕様を満足しているかどうかを検証するための工程である。テストの目的は、プログラムが動作することを示すのではなく、できる限り多くのエラーを見つけることにある。すなわち、テストとは、エラーがないことを証明する手段ではなく、むしろプログラム中に存在しているエラーを見つけだすための手段である。

プログラム中にエラーがあれば、それを上流の工程にフィードバックし、修正した後に再度テストする。この作業をモジュール、プログラム、システムへと次第に大規模な段階に発展させていく。したがって、テスト工程はボトムアップアプローチとなる。

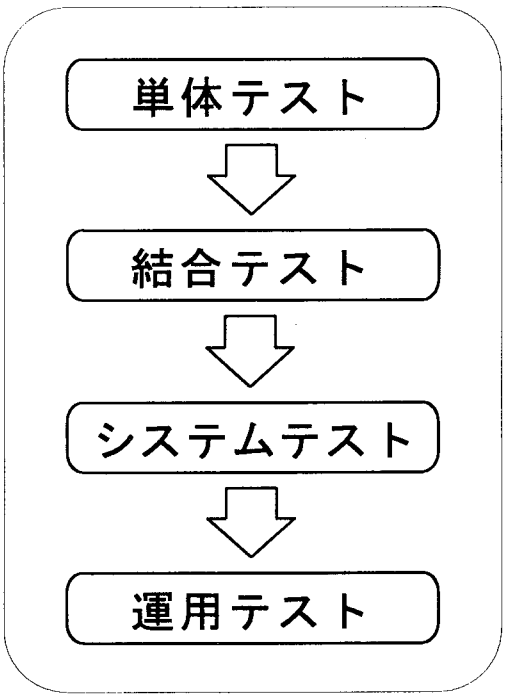
テストは図Ⅱ-39に示すように、さらに四つの工程に分けられ、それぞれの工程で表Ⅱ-4に示すドキュメントが作成される。

(1) 単体テスト

コンパイルしたモジュール単位に、モジュール内に存在する論理的エラーを検出する。

単体テストは、機能テストおよび構造テストの二つに大別される。機能テストは、モジュールの外部仕様をもとに、全ての入出力条件やエラー処理など、モジュールがもつ機能が満足されているかどうかを検証するものである。また、構造テストは、モジュールの内部仕様をもとに、モジュールの論理が正しいかどうかを検証するものである。

機能テストでは、モジュールの論理や内部構造などの詳細な部分に関してはほとんど考慮



図Ⅱ-46　テストの手順

表Ⅱ-6　テストで作成するドキュメント

工　　程	作成するドキュメント
単体テスト （プログラミングで行う）	単体テスト報告書
結合テスト	結合テスト報告書
システムテスト	システムテスト報告書
運用テスト	運用テスト報告書

せずにテストを行う。この場合、モジュールは入力と出力だけしかみえないブラックボックスとして扱われていることから、機能テストはブラックボックステストとよばれる。ブラックボックステストは、ユーザの立場からみた機能テストに向いているが、外部仕様に表れないモジュール内の特殊な処理がテストできないという欠点をもつ。それに対し、構造テストでは、モジュールの詳細仕様やソースプログラムをもとに、モジュールの処理内容を知った上で、処理上の重要なステップを通過するようなテストデータを作成し、テストを行う。この場合、モジュールは内部構造や論理が明確なものとなっているので、構造テストはホワイトボックステストとよばれる。ホワイトボックステストは、プログラマの立場からみた詳細

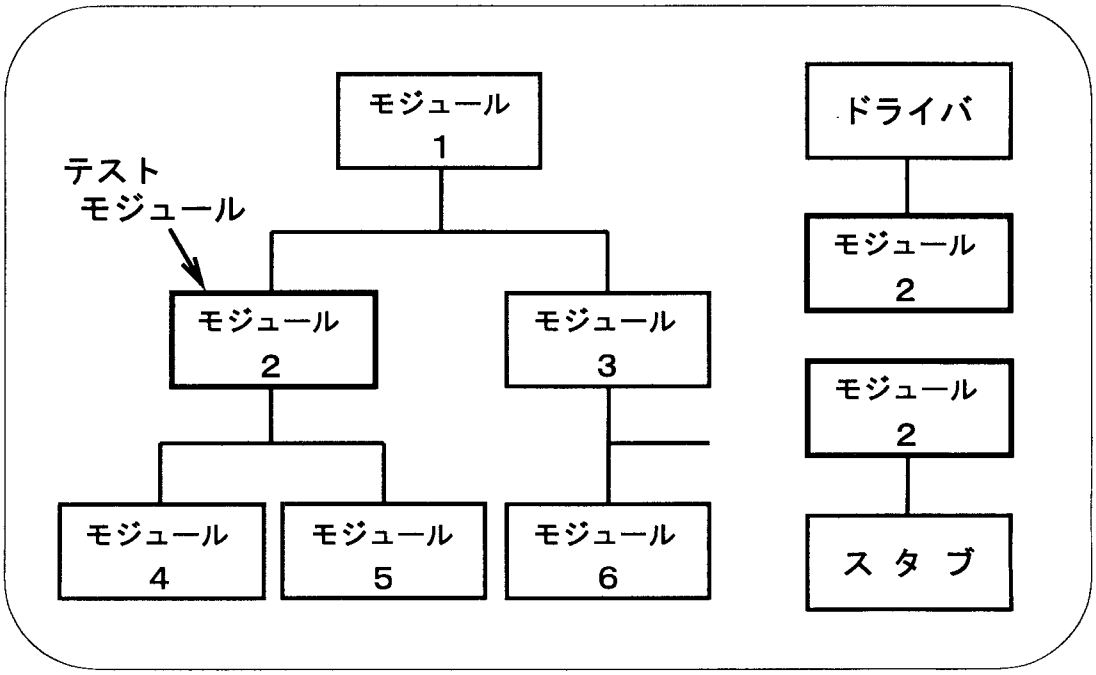
な機能のテストが行えるという長所があるが、ソースプログラムをもとにテストデータが選択されているので、仕様にありながらモジュールで実現されていない機能はテストデータとして選択されず、エラーが発見できない場合があるといった欠点をもっている。

目　的

モジュール単体の動作を確認する。

内　容

モジュールは階層構造をとっている場合が多く、ある一つのモジュールは他のモジュールとデータの受け渡しを行っている。下位のモジュールは、上位のモジュールからデータを受け取って、処理結果を上位のモジュールに返す。このため、モジュール単体のテストを行う場合には、そのデータの受け渡しを行うためのテストツールが必要となる。テストモジュールより上位のテストツールをドライバ、下位のテストツールをスタブという（図Ⅱ-47）。



図Ⅱ-47　ドライバとスタブの関係

単体テストにおける主な作業内容を以下に示す。

- モジュール単体のテスト
- テストデータの見直し
- ドライバおよびスタブの選択もしくは作成

以上の結果を「単体テスト報告書」としてとりまとめる。

（2）結合テスト

単体テストの完了したモジュールやプログラム同士を結合させるテストを行う。また、モジュール間およびプログラム間のデータの受け渡し、入出力のタイミングなどのインタフェースの検証もあわせて行う。

目　的

プログラムの動作を確認し、さらに、システムの機能単位を構成する複数個のプログラムの結合を行う。また、各モジュール間および各プログラム間のインタフェースを確認する。

内　容

結合テストでは、上位モジュールからトップダウン的に結合するだけではなく、各プログラムで共通に使用しているモジュールを先に結合してテストするなど、ボトムアップ的なアプローチも必要となる。ここでも当然、ドライバとスタブが必要となり、結合が進むに従って、順次実際のモジュールと入れ換えていく。また、システムの機能を検証するために、各プログラム間のインタフェースもテストする。

結合テストにおける主な作業内容を以下に示す。

- モジュールの結合テスト
- プログラムの結合テスト
- プログラム間インタフェースの確認
  各プログラム間で、データがうまく渡されているかを確認する。

以上の結果を「結合テスト報告書」としてとりまとめる。

（3）システムテスト

システム全体が要求仕様を満たしているかどうかをテストする。ここでは、システム全体の動作確認を行う。システムテストは、全体的なテストを行うことから総合テストともよばれてる。

目　的

システム全体の動作確認を行い、運用テストを開始してよいかどうかを決定する。

内　容

システムテストでは、結合テストで検証された各機能を組み合わせて、システム全体の結合テストを行う。ここでは、システムを総合的にテストし、目標とした各項目が確実に達成されているかどうかを確認する。さらに、システムの目標とプログラムの内容、各種ドキュメントなどの内容に整合性がとれているかどうかを確認する。

システムテストにおける主な作業内容を以下に示す。

・　システムの総合テストと目標項目の確認

システムテストは、機能テスト、性能テスト、耐久テストなどに大別される。

機能テストでは、開発されたシステムが、ユーザの要求する機能を満たしているかどうかを検証する。ここでは、入出力画面のレイアウトや画面遷移、各種出力帳票などがユーザの要求通りであるかどうかをテストする。

性能テストでは、システムの処理速度やデータ転送速度、応答時間などが目標値を満たしているかどうかを検証する。

耐久テストでは、システムのピーク時に予想される最大の負荷をかけ、処理速度や応答速度がどのように変化するのかを検証する。

なお、システムテストには、故障発生時の回復機能を検証する回復テストや、機密保護機能を検証するセキュリティーテストなどもあり、これらのテストも同様に行うこととなる。

・　目標とプログラム、各種ドキュメントとの整合性の確認

目標としていたシステムを確実に実現するためのプログラムが作成されているか、また、プログラムと各種ドキュメントの間に不整合がないかなどを確認する。

・　品質の最終確認

以上の結果を「システムテスト報告書」としてとりまとめる。

なお、ここまでの工程で、開発部門主体の作業は完了することとなる。

（4）運用テスト

運用テストは、ユーザ部門が主体となって行われるテストである。ここでは、実際のデータを用いて、システム全体の安定度や信頼性を本番の稼働状態で確認することとなる。

運用テストは、開発部門がユーザ部門の承認を受けるテストなので、承認テストや受入れテストともよばれる。

目　的

システムが実際に運用できるかどうかを確認する。

内　容

ユーザ部門のチームが、実際の運用条件と同じ環境を作り出して、機能や操作性などを検証する。また、新旧システムの入れ換え時には、旧システムと並行に、かつ、悪影響を与えないように新システムを運用する。

一連のテストが終了し、ユーザの要求をほぼ満たしていることが確認できれば、運用テストは終了となり、完成したシステムの検収と受領が行われる。

なお、運用テストは、テスト工程の最終テストとなるので、充分に時間をかけて行うことが必要となる。また、ユーザ部門主体で行われるテストであるが、開発部門もできる限り協力しなければならない。

運用テストにおける主な作業内容を以下に示す。

・ 実際の環境と同じ条件下での稼働

実際に新システムが稼働する環境でのテストを行う。ここでは、実際に考えられる負荷を超えたときの動作試験や、誤った操作が行われた場合の想定試験など、ある一定の安全値を考慮したテストを行わなければならない。

・ システム入れ換えの場合、現行システムと並行稼働

システム入れ換え時には、現行システムと新システムが並行稼働する場合が多くある。ハードウェアレベルからシステムを再構築した場合、システムが新旧で完全に二分化されるので問題が発生する可能性は低いが、従来資産の受け継ぎから専用のハードウェア資源を新システムに接続する場合や、職員の研修期間が必要となるためソフトウェアが多くある。このような場合は、一時的に一つの新ハードウェアシステム上で、従来のソフトウェアと新しいソフトウェアが並行稼働することとなり、予想を越えた大きな負荷が生じたり、少しの操作ミスでシステムがダウンするという問題が発生する可能性が高くなる。

運用テストの担当者は、新システムに対して考えられるあらゆるテストケースを実施するとともに、新旧システムがスムーズに移行できるかどうかを確かめなければならない。もし、問題が発生するようなことが考えられた場合は、すみやかにリーダーに報告し、ユーザ側への対応および後の対策を考える。

以上の結果を「運用テスト報告書」としてとりまとめる。なお、運用テスト報告書は、ユーザ部門と開発部門が協力して作成する。