01－R016

CASEに関する調査研究報告書

平 成 2 年 3 月

JIPDEC

財団法人 日本情報処理開発協会

この報告書は、日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて平成元年度に実施した「ＣＡＳＥに関する調査研究」の成果の一部をとりまとめたものであります。

はじめに

１９６８年にＮＡＴＯ後援の国際会議でソフトウェア工学という用語が初めて使われたと言われており、そのことからするとソフトウェア工学は約２０年の歴史を持っていることになる。

１９７０年代には、構造化プログラミング、構造化設計法、ライフサイクルモデル等の優れた研究がなされ、それらはソフトウェアの生産性及び品質の向上等に寄与してきた。

しかし、この１９７０年代の技術は主に「紙」と「エンピツ」を道具としており、考え方としては優れていても、実施という面では、工数が掛りすぎる、一貫性が保てにくい等の理由により、見送りにされていたこともまた事実である。

１９８０年代に入るとパーソナルコンピュータ（ＰＣ）、ワークステーション（ＷＳ）の高機能・低廉化が非常な勢いで進み、その普及が急速に進んだことから、これらをソフトウェア開発の道具とするいわゆるＣＡＳＥ（Computer-Aided Software Engineering）の考え方が１９８５年頃から一気に広まってきた。

ＣＡＳＥツールの多くは１９７０年代の技術をＰＣやＷＳに取り込んだものと言えるが、当初プログラムがあたかも自動的に生成されるかのような発表がなされたことや、ツールの開発が一気に進んだことから、その機能、効果等については多くの誤解を生じている。

こうした欧米の動きに対して、我が国においては労働集約的なプログラム作成段階のためのツールの開発がまず行われ、現在はシステム分析、設計段階でのツール化に進みつつある。

従来の「紙」と「エンピツ」によるソフトウェア開発形態からＰＣ、ＷＳをベースとしたＣＡＳＥツールによる開発形態への移行は、生産性あるいは品質の向上にとって極めて重要な要件と考えられるが移行に当たっては、ＣＡＳＥツールのポテンシャルを正しく把握すると共に、手順、教育、環境整備、投資効果等の問題を検討しておく必要がある。

そこで、財団法人日本情報処理開発協会では、学会、メインフレーム・メーカ、ソフトウェア会社等の代表から成る「ＣＡＳＥに関する調査研究委員会」を組織し、こうした問題に取り組むこととした。平成元年度事業では、手始めとして各委員のご協力の基にＣＡＳＥツールの具体例とその中心となっている考え方の調査を行うと共に海外における適用事例の委託調査並びに調査員の派遣等を実施した。本報告書は、それらの内容をとりまとめたものである。

最後に、本調査研究に当って、ご指導ご協力を頂いた、原田委員長を始め、各委員の方々に深甚なる謝意を表すものである。

平成２年３月

「ＣＡＳＥに関する調査委員会」委員名簿

（敬称略）

委員長	原田　実	青山学院大学　理工学部　経営工学科　助教授
委　員	石井　義興	㈱ｿﾌﾄｳｪｱ･ｴｰｼﾞｰ　社長
委　員	岩田　誠司	㈱東芝　ｼｽﾃﾑ･ｿﾌﾄｳｪｱ技術研究所　研究第３部　ｿﾌﾄｳｪｱ第２担当
委　員	海津　辰雄	㈱SRA　CASE営業部　部長
委　員	木村　陽一	㈱CSK総合研究所　XPT事業部　XPT応用部　次長
委　員	桑野　恭二	㈱PRIDE　取締役ﾁｰﾌ･ｼｽﾃﾑ･ｺﾝｻﾙﾀﾝﾄ
委　員	立田　種宏	ｿﾆｰ･ﾃｸﾄﾛﾆｸｽ㈱　ﾛｼﾞｯｸｼｽﾃﾑ部　ｼｽﾃﾑｻﾎﾟｰﾄｾﾝﾀｰ　ｱﾌﾟﾘｹｰｼｮﾝ･ｴﾝｼﾞﾆｱ
委　員	津田　道夫	㈱日立製作所　情報ｼｽﾃﾑ工場　ｼｽﾃﾑ技術統括部　主任技師
委　員	宮下　洋一	日本電気㈱ C&Cｼｽﾃﾑ研究所　応用ｼｽﾃﾑ研究部　研究課長
委　員	藪田　和夫	富士通㈱　SEﾃｸﾆｶﾙｾﾝﾀｰ　ｼｽﾃﾑ開発技術部　ﾌﾟﾛｼﾞｪｸﾄ担当課長
委　員	向山　博	㈶日本情報処理開発協会　開発研究室　専任調査役

目　　次

1. 事業の目的と背景

1．事業の目的と背景

(1) ＣＡＳＥの登場と発展

近年にみられる高性能多機能ＰＣ、ワークステーションの登場や、その低廉化に伴う普及の拡大は、ＣＡＳＥ（ Computer-Aided Software Engineering ）と呼ばれるツールの出現をもたらした。

ＣＡＳＥツールは、当初システム設計段階を支援するツールとして開発され、バックエンドとしてプログラムプロジェネレータと、またフロントエンドとして戦略計画ツールと結合し、開発工程全般をカバーするツールに成長してきていると言われている。

なお、ＣＡＳＥという用語には明確な定義がなく使われていることから、種々のツールがＣＡＳＥツールとして再登場しようとしている。ソフトウェア工学（ Software Engineering ）がソフトウェア開発、保守のための活動をまとめたもの（ discipline ）とするならばそれらの活動を支援するためにコンピュータを用いたツールはすべてＣＡＳＥツールと呼ぶことができ、コンパイラ、デバッカーもＣＡＳＥツールという議論になる。

そこで、当初ＣＡＳＥツールと呼ばれていたシステム設計段階のツールがリポジトリ、あるいはエンサイクロペディアと呼ばれる今迄になかった新しい設計用データベースを備えていることから、この設計用データベースを持ったツールをＣＡＳＥツールとし、他と区別しようとする意見もある。これらは今後の検討を待たなければならない。

ＣＡＳＥツールを伝統的なウォータフォール型の開発手順に照し合せてみると、その種類はおよそ次のように分類できる。

- 上流ＣＡＳＥ
- 下流ＣＡＳＥ
- リバースエンジニアリング（ＲＥ）
- ＩＰＳＥ（ Integrated Project Support Environment ）

しかし、これらの分類は、ウォータフォール型モデルで工程をどのように分けるかに明確な基準がないのと同様に、上流ＣＡＳＥと下流ＣＡＳＥ等が明確な基準によって分けられているわけではない。最近では、戦略計画用のツールを上流として、システム設計用のツールを中流、それ以降を下流とする考え方もある。

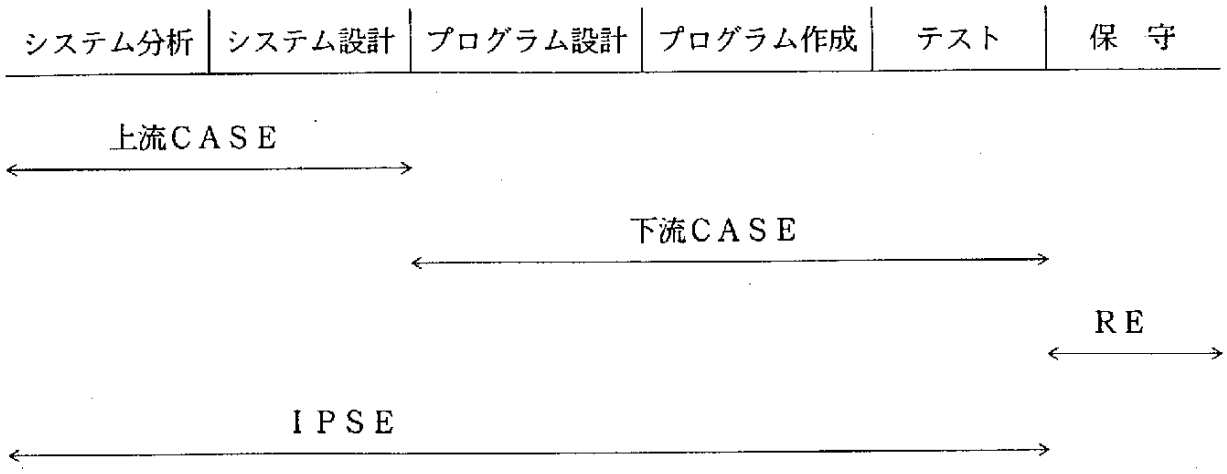


図１－１　ウォータフォール型のソフトウェア開発モデルとＣＡＳＥ

ヨーロッパではＩＰＳＥがしばしばＣＡＳＥの同義語として使用されているし、我が国では、一時似たような概念をソフトウェアＣＡＤと呼んだ時があった。

リバースエンジニアリングあるいはリエンジニアリングと呼ばれるツールは、比較的最近出現し、米国等では非常な関心を浴びてきている。当初のＣＡＳＥツールは新規開発のみを対象としていたのに比べ、このリバースあるいはリエンジニアリングツールは、既存のシステムを対象としている所が大きく異る。即ち、これらのツールは、既存のシステム（プログラム及びデータベース）から仕様を抽出し、その仕様を基に旧システムから高性能ハードウェアや最新のＤＢＭＳへの移向を半自動的に行ったり、新しい機能を付け加えた新システムの構築の支援を行う。

ＣＡＳＥが本格的に広まったのは１９８０年代中頃であり、その歴史は非常に浅いがその発展は次のように分けることができる。

- 第一世代（～８７年） …… 単体ＣＡＳＥ
- 第二世代（８８～８０年代末） …… 統合型ＣＡＳＥ
- 第三世代（９０年代） …… 知的ＣＡＳＥ

この分類からいくと現在は統合型ＣＡＳＥの世代であるといえる。

ＣＡＳＥの統合は、リポジトリと呼ばれる設計データベースを中核として進められており現在、このリポジトリの標準化に向けて多くの活動がなされている。

(2) 事業の目的

代表的なＣＡＳＥツールである Excelerator（ Index Technology 社 ）やＩＥＷ(Knowledge Ware社)のツールは、既に 10,000 セット以上販売されていると言われている。ＣＡＳＥツールは１９８０年代後半に非常な勢いで開発され、またソフトウェア開発の自動化をもたらすものとして多いに宣伝されてきた面があるため、その適用及び効果に関しては混乱があり、期待どおりの効果があったという報告がある反面、導入したものの実際に利用されているのはたったの２０％であるという調査結果もある。また、最近ではソフトウェア開発における自動化、ツール化という面での我が国の立ち遅れも指摘され始めている。

このようにＣＡＳＥツールに関してはさまざまな議論があるが、ソフトウェア生産現場におけるコンピュータ化（自動化）が一番遅れていると言われている現在、ソフトウェア開発にコンピュータを積極的に活用し効率を上げて行くことは極めて重要でかつ緊急な課題である。

そこで、現在欧米等でＣＡＳＥツールと呼ばれているツールの機能、効果、問題点等の実態を解明するとともに我が国のソフトウェア開発環境をいかに向上させ、ソフトウェア開発の生産性及び品質の向上を図っていくかについて調査研究を行うこととした。

(3) 平成元年度事業の概要

平成元年度事業では、ＣＡＳＥに関する調査研究委員会（委員長：原田実　青山学院大学助教授）を設置し、まずＣＡＳＥツールの現状を明らかにすることとした。

また、欧米における代表的なユーザにおける事例調査を英国の調査専門会社であるＯＶＵＭ社に委託し、導入上の問題点、効果等の調査を行った（第３章　海外調査報告参照）。

委員会の開催状況及びその内容は次のとおりである。

第一回（平成元年９月２２日）

① 委員会の進め方について

② 海外事例調査内容の概要

③ ＳＰＡＣＥの機能と効果

第二回（平成元年１１月３０日）

① リアルタイム・システムの構造化分析

② ＸＰＴ－ⅡによるＣＡＳＥの構築

③　海外事例調査結果の概要

第三回（平成２年１月１８日）

①　８０年代のソフトウェア生産技術

②　ＮＡＴＵＲＡＬ２とＰＲＥＤＩＣＴ　ＣＡＳＥ

③　日本市場におけるＣＡＳＥの現状／動向

④　Ｃ－ＮＡＰⅡとＹＰＳ／ＡＰＧ

⑤　ＣＡＳＥツールの試用と問題点

第四回（平成２年２月２８日）

①　ＳＩＳ構築を支える第４世代の統合開発環境

②　ＰＲＩＤＥ－ＩＳＥＭ

なお、これらの内容は第２章にまとめられている。

〔参考文献〕

参考文献は付録として載せてある。

2. ＣＡＳＥツール／方法論の概要と効果

2．　ＣＡＳＥツール／方法論の概要と効果

2．1　ＳＰＡＣＥ

2．1．1　ＳＰＡＣＥの概要

従来から利用していたＴＳＳ端末を通して利用するホスト計算機上で稼働するソフトウェア開発用の計算機支援ツールの多くは、ホストに負荷がかかりすぎる、開発者個々人に専用化された開発環境でない、インタフェースの視覚性が貧弱である、などの限界を持っていた。一方、高速のＣＰＵと高解像度のビットマップディスプレイを持つワークステーションの登場によって図や表を中心としたインタフェースを容易に構築できるようになった。このような背景から、８０年代中期より、このようなワークステーション上で稼働するビジュアルなソフトウェア開発支援ツールが登場しＣＡＳＥ(Computer Aided Software Engineering)ツールと言われるようになった。これらは、要求分析や設計など上流工程におけるそれまでの手書きで済まされていた各種のドキュメント作成を支援するＵｐｐｅｒＣＡＳＥと、ビジュアルなプログラム仕様からプログラムを自動生成するＬｏｗｅｒＣＡＳＥに分けることができる。ここでは、下流工程を支援するＬｏｗｅｒＣＡＳＥの一例として、筆者らが開発し東芝ｴﾝｼﾞﾆｱﾘﾝｸﾞによって実用化されたＳＰＡＣＥについて解説する。

（1）　はじめに

一般に、ソフトウェア開発は、利用者がもつアプリケーションのモデルを、計算機が理解できるコードモデル（すなわち機械語プログラム）に変換することである。

実際、従来事務処理プログラムを開発するには、図２．１－１の上半分に示すように、先ず設計者が業務内容を詳細に定めた設計書（アプリケーションモデル）を作成し、これを基にプログラマがプログラム（コードモデル）を作成し、コンパイラで機械語に変換し、これをコンピュータ上で実行していた。

この変換過程に含まれる作業には、主としてモデルの記述レベル上の変換を行なう合成型作業と、モデルを評価しその特性を求める分析型作業がある。

従来の上流工程を支援するＣＡＳＥツールは仕様書の作図を支援したり分析を支援する色彩が強いが、下流工程を支援するツールではどうしてもこの合成型作業を自動化する必要がある。

このような合成ツールへの入力となるアプリケーションモデルをプログラム仕様と呼ぶと、ＣＯＢＯＬやＦＯＲＴＲＡＮなどのいわゆる第３世代言語で記述したプログラム仕様は、解法を明示的に含んでおり、ユーザモデルからはほど遠い。今後ソフトウェアの生産性を飛躍的に向上させるためには、利用者でも容易に理解でき記述できるプログラム仕様から、機械語やコンパイラが存在する言語によるコードモデルに自動的に変換するいわゆる自動プログラミング的色彩が強いＣＡＳＥツールがぜひとも必要である[1]。

しかし、従来からある事務処理用ＬｏｗｅｒＣＡＳＥ[3]は、プログラムと同内容のものを木構造チャートなどの図で入力することにより処理記述の視覚化をねらったビジュアル言語か、集計や照合や分配などのデータ構造に依存した定型的なプログラムコード群を使いやすい形にマクロ化した言語要素をもち、これらを組み合せてプログラムを簡潔に記述することをねらった第４世代言語[6]かのどちらかであった。

筆者らは、高精細ディスプレイをもつワークステーションが普及するにつれ、ユーザインタフェースのビジュアル化が急速に進むと考え、それ以前から行なってきた設計書の書式と記述法の研究成果を基に、これを計算機処理可能な言語とすることによって望ましいプログラム仕様を実用化できると考えた。

このように筆者らは、まず真に作成しやすいプログラム仕様とは何かを明らかにし、次にこの仕様をプログラムに変換する方式や知識を構築するという手順で、ＬｏｗｅｒＣＡＳＥとしてＳＰＡＣＥ（SPecification Acquisition and Compiling Engine）を開発した[2],[4],[5]。

ＳＰＡＣＥは、図２．１－１の下半分に示すように、ディスプレイ上での設計書作成作業を支援するとともに、作成された設計書からＣＯＢＯＬプログラムを生成する。

したがってＳＰＡＣＥは、従来の第４世代言語（４ＧＬ）とビジュアル言語の両方を統合した、次世代のビジュアル４ＧＬといえるものであり、ソフトウェア開発の中下流工程を一貫して支援する統合型ＣＡＳＥである。この結果、プログラム仕様の視覚化と抽象化の両者が統合的に達成され、処理内容の記述と入力方法を大幅に改善できると期待されている。

ＳＰＡＣＥのようなＬｏｗｅｒＣＡＳＥを開発するに当って最も重要なことは、ソフトウェアの構造や要素間の関係を表わすのに十分視覚的であり、かつ複雑で大規模なモジュールの機能を正確に表わすのに十分形式的で簡潔な計算モデルを開発することである。筆者らは機能表現の視覚化のために決定表に改良を加えたロジックテーブルを用いる。またモジュール機能を抽象化して簡潔に表現するために、モジュールの実行状態を条件式という関数とその値で宣言

的に指定できるようにした。この2点、ロジックテーブルと条件式が、ＳＰＡＣＥを他のＬｏｗｅｒＣＡＳＥに対して独創的なものにしている。

以下では、ＳＰＡＣＥのこれらの特徴に関して、プログラム仕様の視覚化を（2）で、抽象化を（3）で、両者がうまく統合されたプログラム仕様の事例を（4）で、デバッグ支援機能を（5）で、プログラム生成の仕組みを（6）で、それぞれ解説する。

（2）　プログラム仕様の視覚化

コンピュータに業務処理を行わせるには、業務の多種多様な特性をコンピュータに入力する必要がある。従来はこれらすべてをプログラムやＪＣＬ（ジョブ制御言語）として記述し、コンピュータに入力していた。これらは1次元のテキスト形式言語であり、図や表のような2次元の記述方法に比べて仕様要素を関係付ける手段に乏しく、その結果理解しづらいものであった。

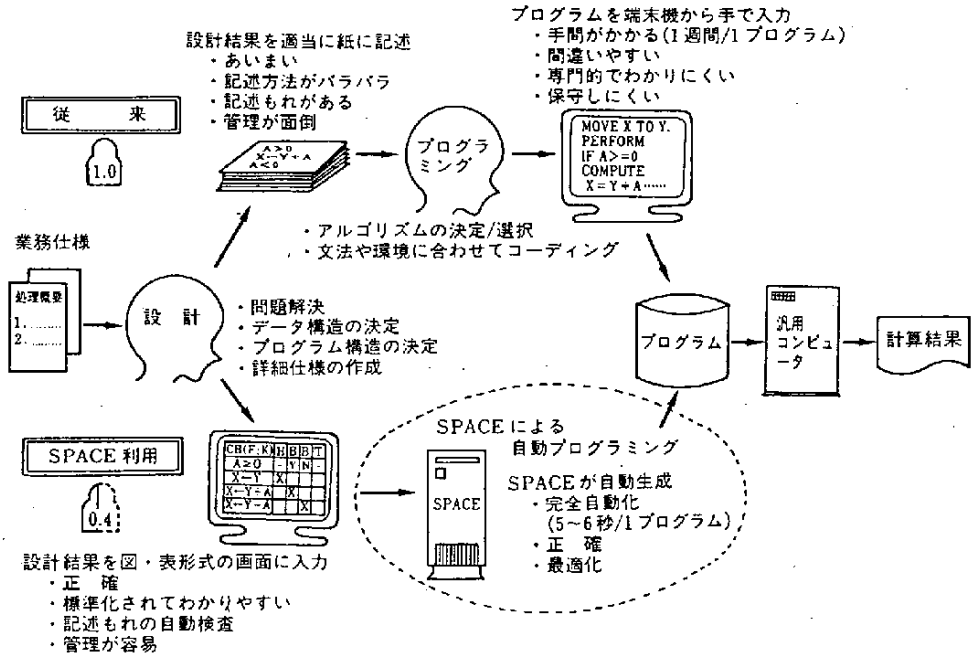


図2．1－1　ＳＰＡＣＥによるソフトウェア開発形態の刷新

これに対しＳＰＡＣＥでは、従来からある設計書に近い形式をもつ、2つの図形式画面と5つの表形式画面（後出の図4参照）が用意されている。利用者は、これらの設計画面に業務処理を構成するジョブ、ファイル、プロセス（＝プログラム）、およびモジュールに関する様々

な情報を、それぞれに最も適した記述方法で指定する。

例えば、ジョブを構成するプログラムとファイルの関係はプロセスフロー図に、ファイルの諸属性はファイルテーブルに、ファイルごとのキー定義とデータ構造はデータ構造テーブルに、プロセスのモジュールへの分割は階層構造図に、モジュールの機能はロジックテーブルに、それぞれに適した視覚的な方法で表わす。各画面への設計内容の記述は、あらかじめ表記法と意味が定められた図形や文からなる仕様要素－設計パターンと呼ぶ－から、適切なものを選んで記入するといった"はめ込み方式"で行なう。

これらの設計画面は図／表形式であるため、これに記入される図形や文はその記入場所や接続の種類によって明確な意味の基で関係付けられる。この結果仕様が視覚化され、その内容を正確かつ簡潔に表現・理解できるようになる。

1)　ロジックテーブル

一般に、コンピュータで計算を行う最大の利点は、同一形式の多数のデータに対して同様の計算を繰り返し行う時、図2．1－2の上半分に示すように、この繰り返し1回分の計算方式と各繰り返しのためにデータを準備する処理を指定するだけで、仕様を記述できることである。

ＳＰＡＣＥでは、モジュールが次節で述べる処理パターンのいずれかをもつようにして処理記述の抽象化を図るために、各モジュールはこのような繰り返しを、その中に高々1つ（同一範囲の多重ループは1つの繰り返しと考える）しか含まないものとする。このようなモジュール分割基準の基で個々のモジュールの機能を、図2．1－2の下半分に示すように、決定表に初期処理欄と前処理欄を追加したロジックテーブルと呼ぶ設計画面に記述する。したがって、モジュール機能の構成要素である条件C_iと処理A_jとその対応関係R_kを独立でかつ整理された形で記述できる。このように視覚化されたプログラム仕様はエンドユーザにも理解しやすく、したがって設計の確認を得られやすく、また後々の変更も行いやすい。

(3)　プログラム仕様の抽象化

プログラム仕様がどんなに視覚化されても、木構造チャートのようにその内容がプログラムと同程度に複雑なものであれば、良い仕様と言えない。処理内容をプログラムよりも抽象度の高いレベルで簡潔に記述できる仕組みが必要である。

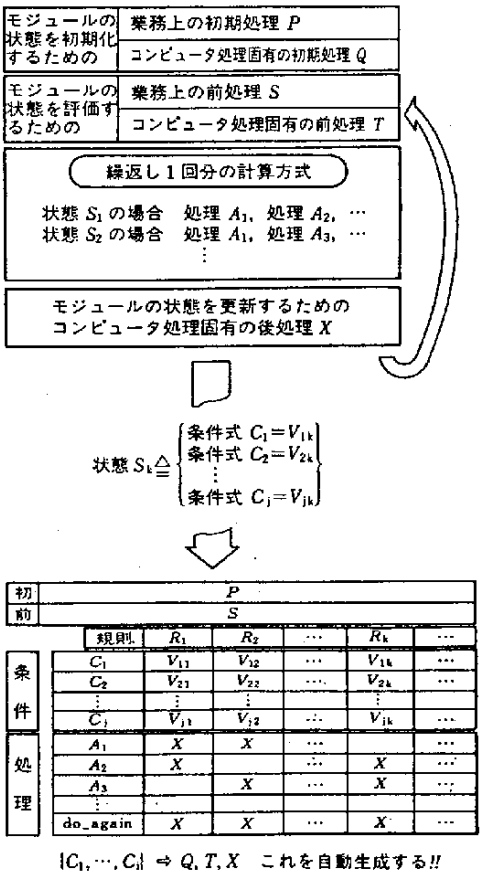


図2．1－2　ロジックテーブルと条件式を用いたモジュールの機能記述

1）　処理パターン

一般に、事務処理プログラムは、本来手順的な事務業務を機械化するものであるから、プログラムの出力応答がファイルや入力から得られる手順や構造を詳細に与えないとその仕様を記述できない、－いわゆる『たちの悪い(wicked)』－、問題である。しかしこの手順の内、照合や集計ロジックなどのデータ構造に依存したレコード入力制御ロジックやオンライン画面制御ロジックは、かなり定型化できることは、事務プログラム作成上の知識として広く知られている[6],[8]。実際このような制御ロジックは、以下の６つに分類され、処理パターンと呼ぶ*1。

① 複数の順ファイルからレコードを入力し、各ファイルに照合キーで指定されるレコードがあるかないかに応じて特定の処理を行う照合処理。

脚注*1:処理パターン①と②ではファイルはキー項目の値の昇順でソートされているとする。

② 順ファイルからレコードをキー項目の値順に入力し、キー項目が同一値をもつレコード群ごとに特定の処理を行うグループ処理。

③ 索引ファイルからキー項目値に関する与えられた論理条件を満たすレコードを直接検索し、その成否に応じて特定の処理を行なう検索処理。

④ オンライン画面から１レコード分のデータとファンクションキーなどを読み込み、その状態に応じて特定の処理を行なうオンライン処理。

⑤ データベースから与えられた論理条件を満たすレコードを検索し、その成否に応じて特定の処理を行なうＤＢ処理。

⑥ 作業領域中のデータの値を用いた論理条件による、レコード入力を伴わない計算処理。

このような処理パターンをもつモジュールの機能をロジックテーブルを用いて記述する時は、図２．１－２の上半分に示すように、①モジュールの状態を初期化するために呼び出された時に１度だけ実行される初期処理、②繰り返し内の処理に先立ってモジュールの実行状態を評価するための前処理、③繰り返しにおいて『どんな状態S_iの時どの処理 $\{A_{i1},\cdots\}$ を行うか』を指定する本体処理、および④次の繰り返しのためにモジュールの状態を更新する後処理の４つを指定すれば良いことは容易に分る。

この内、設計者が指定すべきものは、業務に固有な処理である本体処理$\{S_i \rightarrow A_{i1},\cdots\}$と、集計エリアの初期化などの初期処理Ｐと、検査モジュールの呼び出しなどの前処理Ｓの３つである。一方、レコードの初期読みなどの初期処理Ｑ、照合キー値の設定などの前処理Ｔ、およびレコードの先読みなどの後処理Ｘは、データ構造と処理パターンの型名を指定すれば、一意的に定まり自動生成可能である。

２） 条件式

条件式は、上で述べた処理パターンをもつモジュールの実行状態Ｓを簡潔に表すために予め定義された関数$C_i : S \rightarrow \{v_{ij} \mid j=1,\cdots\}$（$v_{ij}$は関数$C_i$ごとに定められた相異なる取り得る値）である。これを用いれば、モジュールの個々の実行状態S_jは関数とその値の組み合せ〔関数C_1＝値v_{1j},…〕で表すことができる。

ＳＰＡＣＥは、条件式ごとにその値を計算するロジック以外に、その付帯的意味として（マニュアルに）定められた処理Ｑ、Ｔ、Ｘをロジックテーブルの実行ループの適切なところに自

動的に展開する。したがって利用者は、モジュールの処理パターンを表現するように条件式を選び、ロジックテーブルの条件部に併置するだけで、モジュールの機能を簡潔に記述できる。

現在、ＳＰＡＣＥには処理パターン①～③と⑥に関する９つの条件式がある。この内各処理パターンごとに代表的なもの各々１つを表２．１－１に示した。

表２．１－１　ファイル処理記述用の条件式（一部）

名前	書　き　か　た	条件値	読　み　か　た
照合式	MC(F;K)	Y, N, E, *	ファイルFから入力したレコードのグループキーKと，現在の処理対象を表す照合キーKの値が等しいときYes，等しくないときNo. ＊はNまたはEの意味.
制御切れ式	CB(F;K)	H, B, T, E, *	当該モジュールが初期化された直後の状態のとき，あるいはファイルFから入力したレコードによって，グループキーKが同一値の新しいレコードグループに入ったときHead，グループ内での各レコード処理状態のときBody，現在のグループから抜け出ようとする終了状態のときTail. ＊はTまたはEの意味.
検索式	SC(R;K;r;C1,...,Cn) ただし，Ci≙項目Di：式Si	Y, N	グループキーKを参照キーとするアクセスパス上で論理条件(Di r Si, ...)を満足するレコードがファイルR中に存在するときはYes，存在しないときはNo.
論理式	COBOLの論理式C	Y, N	論理式Cの評価結果が真のときYes，そうでないときNo.

Fは順/動アクセスファイル，Rは乱/動アクセスファイル，Kはグループキー.
MCとCBに対する条件値Eは入力したレコードがEof (End of file) レコードであるとき.
SCにおいて，グループキーKに対応する集団項目（または基本項目）は項目D1,...,Dnから構成され，またrはCOBOLで許される比較演算子である.

例えば、ファイルFと他のファイルの照合処理を行なうモジュールの状態は、MC(F;K)を用いて表せる。すなわち、ファイルFから入力したレコードがキーKで表される現在照合中の対象*2である状態はMC(F;K)=Yで、そうでない状態は=Nで表す。また，ファイルFのキーKによるグループ処理はCB(F;K)を用いて表せる。すなわち、ファイルFから入力したレコードによって、キーKが一定の新しい集団に入った初期状態はCB(F;K)=H(ead)で、集団中の各レコード（先頭や最後のレコードも同様に含んで）処理状態は=B(ody)で、現在の集団を終ろうとする状態は=T(ai

脚注*2:現在照合対象になっているレコードは、照合キーMKと同じ値をキー項目Kにおいてもつことによって識別される。なお、照合キーMKの値は、各ファイルからキー順に読み込まれている現在レコードのキー項目Kの値の最小のものである。

I)で表す。

これらを用いれば、例えば複数のファイルF,…に対する照合処理の各状態はMC(F;K)をファイルの数だけ、複数のキーK,…による多段のグループ処理の各状態はCB(F;K)をキーの数だけ、それぞれ条件部に併記するだけで、その制御ロジックの詳細に立ちいらずに簡潔に表現できる。

なお現在、オンライン処理パターン④は、前処理欄に、アプリケーション画面の固定部の表示処理と可変データの読み込み処理を直接ＣＯＢＯＬで記述し、また条件部に、読み込まれたデータの値、終了コード、ファンクションキーの値などに関する論理式を記述することによって表現している。同様にＤＢ処理パターン⑤に関しても、レコードの検索処理を前処理欄に直接ＣＯＢＯＬで記述している。このようなＣＯＢＯＬコード群はホスト計算機のオンラインモニタやＤＢモニタごとに異なる拡張ＣＯＢＯＬ言語を使って記述する必要がある。しかしこれらのコード群もアルゴリズムの大筋はモニタによらず共通なので、他の４ＧＬと同様に可変部をパラメタ化することで定型化できるはずで、これらを暗黙の付帯処理にもつ条件式として近い内に登録する予定である。

3）　処理文

処理文は、モジュールの処理要素を記述するためにロジックテーブルの処理部に記入する設計パターンである。処理文の多くは個々の計算やレコード出力を表すＣＯＢＯＬ命令そのものである。これ以外に、ＳＰＡＣＥ特有のものとして、子モジュールを呼び出すためのEXEC文や、条件式に伴う暗黙の後処理Ｘを行った後次の繰り返しに戻るためのDO-AGAIN文などがある。

また、一連の処理をマクロ化したものとして、指定された式の集計を行うSUM文や、処理記述の一部のみがモジュールの実行状態によって異なる処理を記述するのに便利なWITH文などがある。

（4）　ＳＰＡＣＥによるプログラム仕様の事例

本章では、事例を用いて、ＳＰＡＣＥにおけるプログラム仕様がいかに視覚的に理解しやすく、機能記述の抽象度が高く、記述しやすいものであるかを具体的に説明する。用いる問題はファイル処理の代表例である図２．１－３に示すような在庫管理問題である。この問題は、『各製品ごとに、入出庫レコードがあれば入出庫量合計を求め在庫レコードを更新し、さらに

新在庫量が最低在庫量を下回れば発注レコードを出力し、また削除レコードがあれば在庫レコードを削除する。これらの各処理において所与の条件に反する時は、エラーメッセージを出力する。』ものである。

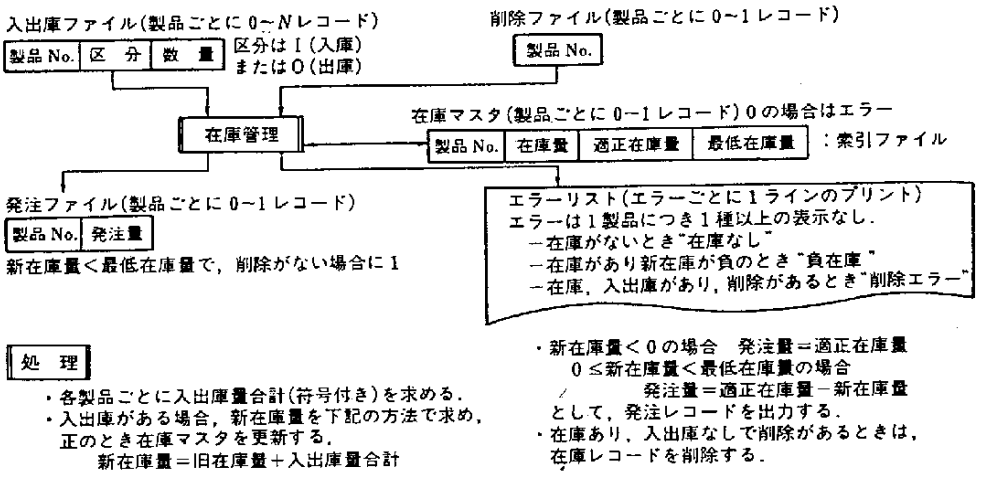


図２．１－３　ファイル処理の代表例としての在庫管理問題

利用者は、次の５つの設計ステップにしたがって、各設計画面を完成していくことでプログラム仕様を作成する。これらの記述例を図２．１－４に示した。

[Step1:プロセス分割]

ジョブを一連のファイル処理を行なうプロセス（＝ジョブステップ）に分割し、結果をプロセスフロー図①に書き表す。この例は、『ジョブが在庫管理プロセスのみからなり、これは３つのファイルを入力し２つのファイルと１つのリストを出力する。』ことを表している。

[Step2:ファイル属性定義]

このジョブで使われるファイルの諸属性を、ファイルテーブル②に記入する。この例は、『入出庫ファイルのデータは、Q5006.PDM.DATA(NYUSHU)に格納されており、そのレコードの定義はコピーライブラリのNYUSHUにある。このファイルは順編成ファイル(PS)で、１レコードは８０バイト固定長(F)で、１ブロックには２０レコード入っている。』ことなどを指定している。

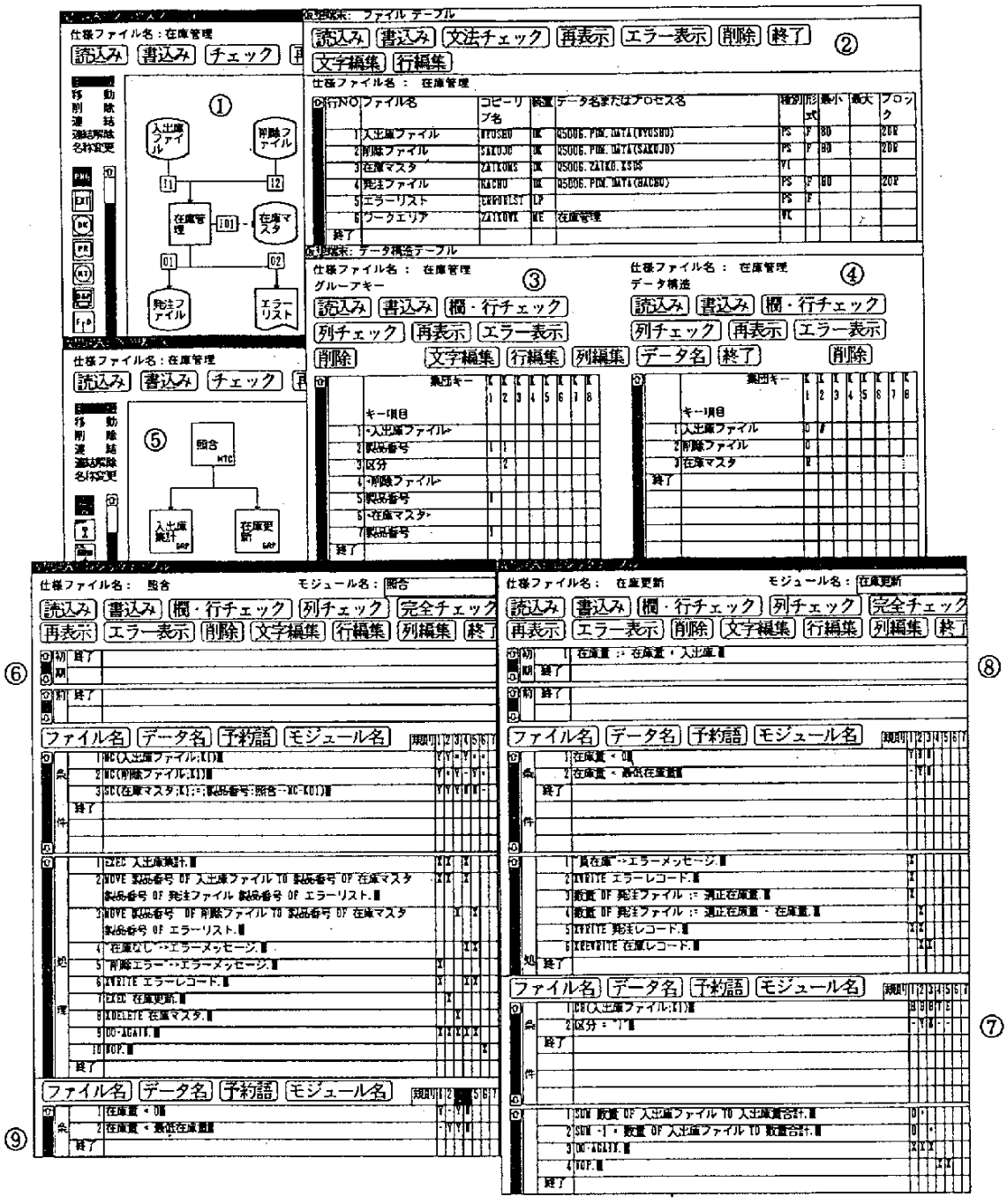


図２．１－４　在庫管理問題をＳＰＡＣＥを用いて記述したプログラム仕様

[Step3:データ構造定義]

事務処理プログラムでは複数のデータ項目を繋げて１つのキーとして扱うことが多い。例えば製品番号と区分をプログラムの中で一組のものとして扱いたい場合には、例えばＫ２という

名前を与えて、③のようにグループキー定義テーブルに記入する（ただし、実際にはＫ２はこの例題の以後の記述では引用しない）。

一方、④のデータ構造テーブルは、各ファイルがどのようなレコードグループ（グループキーが同じ値をもつレコード群）の繰り返しから構成されているかを表す。例えば、入出庫ファイルは（製品番号=K1が一定の）製品レコード群の繰り返しからなり（これをK1列にＯを記入することで指定する）、各製品レコード群は同一製品の(K2が一定の)入庫レコード群と出庫レコード群から成る（最下位グループが複数レコードを含む時はＯの替りに＃を指定する）。

[Step4:モジュール分割]

プロセスフロー図中の各プロセスを、その機能を分担して実現する１群のモジュールに分割する。この時各モジュールは（３）の１）で論じた６つの処理パターンのいづれかをもつようにする。この結果を階層構造図⑤に書き表す。この例は、『在庫管理プロセスが、照合パターン（□の右下隅のMTCで表わす）をもつ照合モジュールと、それから呼び出されるグループパターン（同じく、GRPで表わす）をもつ入出庫集計モジュールと、計算パターン（同じく、CMPで表わす）をもつ在庫更新モジュールによって構成されている。』ことを示している。

[Step5:モジュール機能定義]

モジュールの機能をロジックテーブルに記入する*3。

照合モジュールの機能は、⑥に示したようなロジックテーブルで表わせる。このモジュールは、各製品ごとに入出庫ファイル、削除ファイル、在庫マスタに当該製品のレコードがあるかないかに応じて様々な処理を行う。したがって、各ファイル内に照合すべき（キーK1で識別される）製品のレコードがあるかどうかを知るために、順編成の入出庫ファイルと削除ファイルに対してはＭＣ式を、索引編成の在庫マスタに対してはＳＣ式（ここで照合--MC-K01はキーK1に対応する照合キーMK1の値の格納領域名）を条件部に記入する。

次に、このモジュールで行う具体的な処理、－すなわち、入出庫集計モジュールの呼び出し(A1)、固定データの転送(A2、A3)、エラーメッセージの出力(A4～A6)、在庫レコードの更新(A7)と削除(A8)、再実行処理(A9)、無処理(A10)－、を処理部に記入する。

脚注*3：以下でｉ番目の条件、処理、規則をCi,Ai,Riで表わす。

最後に、題意にしたがって条件と処理の対応を取る。例えば、３ファイル全てにレコードがある時（C1～C3=YのR1)は、削除エラー処理をするためにA1、A2、A5、A6、A9に実行指示を示す"X"を記入する。削除ファイルにのみレコードがない時(C1、C3=YでC2=*のR2)は、更新処理をするためにA1、A2、A7、A9に実行指示を示す"X"を記入する*4。この様な処理は、最終的に入出庫ファイルと削除ファイルが共に存在しなくなる(C1=C2=*のR6)まで、DO-AGAIN命令によって繰り返される。

一方、入出庫集計モジュールは、⑦に示すように、当該製品の入出庫レコードを順に読み込んでこの製品の入出庫量合計を求める。このために、製品ごとのグループ処理の各状態を表らわすためにC1にＣＢ式を、さらに区分の値によって処理を分けるためにC2に論理式を記入した。この結果、このモジュールが呼ばれるとまず集団ごとの初期状態になるので、この時(CB=HのR1)、SUM命令に対して0指示を行ない、入出庫量合計を０に設定する。また、レコード処理状態の時(CB=BのR2,R3)には、区分が"I"かどうかに応じてSUM命令を用いて、数量(A1)または-1*数量(A2)を入出庫量合計に加算する。当該製品レコードがなくなると(R4,R5)、このモジュールを呼び出した照合モジュールに戻る。

最後に、在庫更新モジュールは、⑧に示すように、初期処理として在庫量を更新し、これとゼロ（０）および最低在庫量との大小関係によって、エラーメッセージの出力(A1,A2)や、発注レコードの出力(A3～A5)や、在庫レコードの更新(A6)などを行う。なおこのモジュールは内部で繰り返しを行わず、処理が終わると照合モジュールに戻る。

このようにＳＰＡＣＥでは、レコードの入力やキーの比較などの制御ロジックの詳細を記述せずに、条件式とその関数値という抽象度の高いレベルでモジュールの状態を指定でき、この各状態に個々の計算処理を対応づけることでモジュールの機能を記述できる。

（５）　仕様検査、コード生成、デバッグ

仕様作成が終ったならば、各画面において欄・行チェックボタンや列チェックボタンを押して、表現方法が正しいかを検査する。誤りがあれば、その行あるいは列の番号が網掛けされ、エラー表示ボタンを押せばエラーの内容が表示される。さらに、モジュールの機能を定義するロジックテーブルに対しては、記述した仕様が完全であるか（記述に漏れや矛盾がないか）な

脚注*4：－(don't care dash)は各条件式に対して、その任意の条件値を表わす。

どを完全チェックボタンを押すことで検査できる。例えば、⑨に示すように、規則R1とR2を指定した段階でこの検査を行なうと、矛盾（＝重複）している規則R3の列番号が斜線で網掛けされ、漏れている規則R4の列番号が点々で網掛けで表示される*5。

この後、コード生成画面において、コード生成に使う仕様ファイル（各設計画面の内容が格納されたファイル）群を指定しコード生成ボタンを押すと、この事例で約670行のＣＯＢＯＬプログラムが生成される。同様にしてＪＣＬも生成される。生成されたプログラムのコンパイル・実行はホスト計算機に転送してから行なう。

実行結果が、利用者の期待に反している場合は、その誤りは入力したプログラム仕様にある。この誤りの検出のためにロジックテーブルの実行時トレースを出力できる。このトレースを見れば、いつどのロジックテーブルが呼ばれ、その時のレコードの状態はどうであったか、どの規則が選ばれ、どの処理が実行されたかを一目で理解できる。

設計者は実行中の正しいモジュール間の遷移状態を認識していることが多く、実行時トレースを見れば、意図した様にプログラムが実行されなかった原因を容易に見つけることができる。

（６）　ＳＰＡＣＥにおけるプログラム生成方式

ＳＰＡＣＥが生成するプログラムは、図２．１－５の左側に示すように、データ宣言、主ルーチン、入力手続き、ロジックテーブルを実行するループ群からなる基本構造（プログラムフレームと言う）をもつ。ＳＰＡＣＥは与えられた仕様ファイル群を解析し、記入された設計パターンごとにその名前や型や他のパターンとの関係をシンボルテーブルに登録する。全ての仕様ファイルを解析した後、シンボルテーブルに登録された各設計パターンごとにそれが表す定型処理を記述したプログラムコード群（これをプログラムパターンと言う）をプログラムフレームの適切なところに編集して埋め込む。

具体的には、データ宣言部は、ファイルテーブルとプロセスフロー図の情報を基にしたファ

脚注*5:決定表が完全であるとは、記述した条件式で分類されるありとあらゆる場合が、記述された規則で漏れなく重複することなく表現されていることである。ＳＰＡＣＥは、記述された規則をそれと同等な基本規則（全ての条件値が"*"や"-"以外の基本値である規則）に展開し、これらが可能な全ての基本規則列中に出現する回数が0(未指定)、1(正常)、2以上(重複)かを調べることによって（完全性判定アルゴリズムと呼ぶ）、モジュール仕様の完全性を判定する。

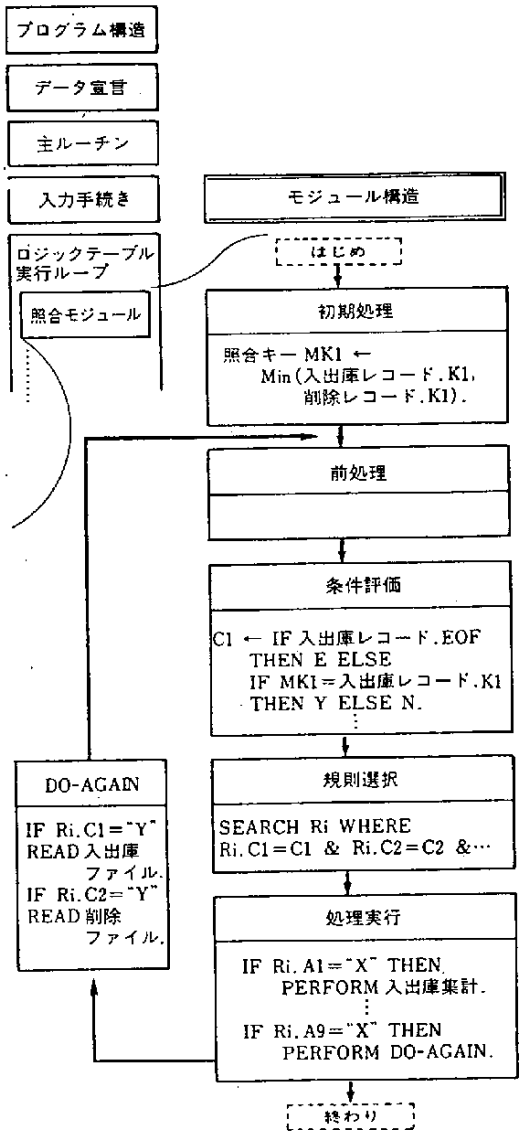


図2．1－5　生成プログラムの構造と条件式に対するコード展開例

イルとそのレコードの宣言、ロジックテーブルの規則列{Ri}の配列宣言、ロジックテーブル中に用いられた条件式に関連する制御変数の宣言などからなる。

主ルーチンは、各ファイルのオープンと初期読み込みの後、階層構造図上の最上位のモジュールを呼び出し、最後に各ファイルのクローズを行なう手続きである。

ロジックテーブルを実行するループは、図2．1－5の右側に示したような構造をもち、各ブロックには条件式や処理文に対応するＣＯＢＯＬ命令が埋め込まれる。例えば、例題の照合モジュールに対しては、初期処理部でＭＣ式（ＳＣ式の説明は省く）の暗黙の初期処理として入出庫ファイルと削除ファイルの現在レコードのキーK1の小さい方を照合キー(照合--MC-K01)

にセットし、条件評価部で照合キーと両レコードのキーK1を比較してＭＣ式の値を計算し、この値をもつ規則Riを規則選択部で求め、処理実行部で規則Riに実行指示記号Xがある時に各処理文に対応する処理を行うＣＯＢＯＬ命令を実行する。特にDO-AGAIN文に対しては、単なる繰り返しの先頭へのGO TO命令だけでなく、その前にＭＣ式の暗黙の後処理として、直前に実行された繰り返し処理において各ファイルFからMC(F;K)=Yならば１レコードを読み込む命令も生成する。

このように条件式は、プログラム生成時に（３）の２）で述べた暗黙の初期処理や前処理やDO-AGAIN文に伴う後処理などのコンピュータ処理固有の制御ロジックを伴った関数に変換される。したがって、利用者はモジュールの機能を、その制御ロジックの詳細に立ち入ることなく簡潔に記述できる。

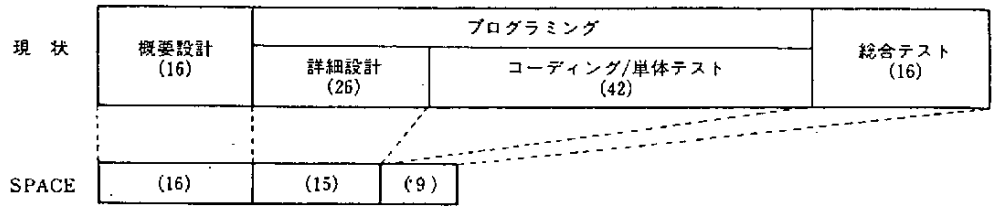
２．１．２　ＳＰＡＣＥの効果

ＳＰＡＣＥでは、業務の諸特性を適切に表現する図・表形式の設計画面に、処理ロジックを抽象的に表現できる条件式などの設計パターンを記入するという方式でプログラム仕様を作成することで、プログラム仕様の視覚化と抽象化を両立することができた。

これによって、①従来の手書きの設計書作成に比べ、詳細設計作業の信頼性と生産性を大幅に向上できる、②プログラミング作業を自動化し、開発期間を大幅に短縮できる、③設計書のみを保守すればよいので保守作業を大幅に軽減でき、また資源管理も容易になる、などの効果を期待できる。

いくつかのプログラムに適用した結果、各工程の効率化率は、それぞれ詳細設計が約４０％減、コーディング／単体テストが１００％減、総合テストが約４０％減であった。この効率化率をB.W.Boehmによる新規プログラム開発における工程別プログラム開発費配分表に適用すると、図２．１－６に示すように、新規プログラムの総開発コストの約６０％を削減できることが期待できる。また生成されたプログラムは手作りプログラムに比べて、ＣＰＵ時間は１～２倍程度かかるものの、実行時間では全く同じという結果を得ている。

このように、ＳＰＡＣＥを用いることによって、大幅に開発期間を短縮できる。同時にプログラムやドキュメントの信頼性や品質も向上し、保守効率も向上できる。



ただし，( )内の数字は工数割合を示す．現状の割合は B. W. Boehm による小さなプログラム(2KDSI)の組織だった開発形態における工数比率による．なお，要求分析は，サンプルデータにばらつきが多く，除かれている．

図２．１－６　SPACEによって総開発コストの60%を削減できる

２．１．３　ＳＰＡＣＥの適用上の注意と今後の計画

ＳＰＡＣＥは、現在サンマイクロ社のＳＵＮウインドゥ版とΣウインドゥ版（仕様コンパイラと言う）の２種類が存在する。特に仕様コンパイラでは、データ項目名やファイル名などの日本語記述が可能になり（実際、図２．１－４は仕様コンパイラの設計画面である）一段と仕様の理解性が向上する。

今後は、

①Ｘウインドゥ版を開発し多種類のワークステーション上で稼働するようにする、

②オンライン／ＤＢ処理に関する条件式を追加する、

③意味的に等価な範囲内で最も高速なプログラムを生成する最適化機能の追加する、

④上流工程のＣＡＳＥと連係をとる、

⑤ＳＰＡＣＥでのモジュール設計を自動化するシステムを開発する

などを予定している。

参考文献

[1] R.Balzer:"A 15 Year Perspective on Automatic Programming",IEEE Trans. Softw. Eng., Vol.SE-11,No.11, pp.1257-1268(1985).

[2] 原田実:"仕様ｺﾝﾊﾟｲﾗSPACE",情報処理学会第33回全国大会論文集,PP.609-610(1986).

[3] 原田実:"事務処理分野における自動ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ",情報処理, Vol.28, No.10, pp.1378-1398 (1987).

[4] 原田実:"COBOLﾌﾟﾛｸﾞﾗﾑ自動生成ｼｽﾃﾑSPACEにおける仕様の視覚化と抽象化",電子情報通信学会論文誌,VOL.J71-D, No.12, pp.2555-2562(1988).

[5] 原田実:"プログラム仕様の視覚化と抽象化－ビジュアル４ＧＬ：ＳＰＡＣＥを事例として－",自動プログラミングハンドブック,オーム社,pp.427-436,1989.12.

[6] 河野,鵜飼,中尾田,岸:"ﾃﾞｰﾀに着目した仕様を入力とするCOBOLﾌﾟﾛｸﾞﾗﾑ生成ｼｽﾃﾑDSLの開発",日立評論,Vol.65, No.7, pp.53-58(1983).

[7] J.Martin(安部,藤田訳):"ﾌﾟﾛｸﾞﾗﾏｰなしのｱﾌﾟﾘｹｰｼｮﾝ開発",ｱｼｽﾄ出版会(1984).

[8] J.D.Warnier(鈴木訳):"ﾜｰﾆｴ･ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ法則集",日本能率協会(1975).

2.2 SEWB

2.2.1 SEWB概要

ソフトウエア開発の生産性向上，信頼性向上を目的として各種の図形を利用した設計技法が開発され、定着化してきた。SEWBはワークステーションによるソフトウェアの分散開発環境で、システム開発ライフサイクルの一連の工程に対して図形を用いたビジュアルな開発を実現している。

SEWBでは、対話管理，ライブラリ管理を統括したCASEプラットフォームである「SEWB基本部」を核とし、ツール間のデータ授受，操作の統一を実現した。

(1) SEWBの開発方針

日立製作所ではホストコンピュータの会話処理で稼働するEAGLE2（Effective Approach to Achieving High Level Software Productivity2）を既に開発しソフトウェア開発の効率をあげてきた。

近年、高性能多機能ワークステーションが普及し、ワークステーションを利用した開発への期待が高まっている。日立ではワークステーション2050（OSはUNIX）を利用したソフトウェア分散開発環境としてSEWB（Software Engineering Workbench）を開発した。

SEWB開発の基本方針は、コンピュータ処理に都合のよい従来のコンピュータ寄りのマンマシンインタフェースではなく、利用する人の立場に立ち、人間の思考を助けるためのオブジェクト指向マンマシンインタフェースを確立することである。

① 統合ソフトウェア開発環境の実現

個別に開発したツールの寄せ集めではなく、統合化したソフトウェアの開発環境を実現する。SEWBでは支援ツールの統合化方式として集中制御型統合方式を採用している（図2.2-1）。従来のツールボックス型統合方式ではファイル間でデータの受け渡しをするだけの粗結合であるため、操作の不統一や共通的な機能に差が出た。「SEWB基本部」を核とする集中制御型統合方式は、対話管理やデータ管理をツールから分離した密結合である。

これにより、ユーザインタフェースの統一，仕様とプログラムのソフトウェア情報の一元管理，支援ツール間の重複機能の統一を可能にしている。

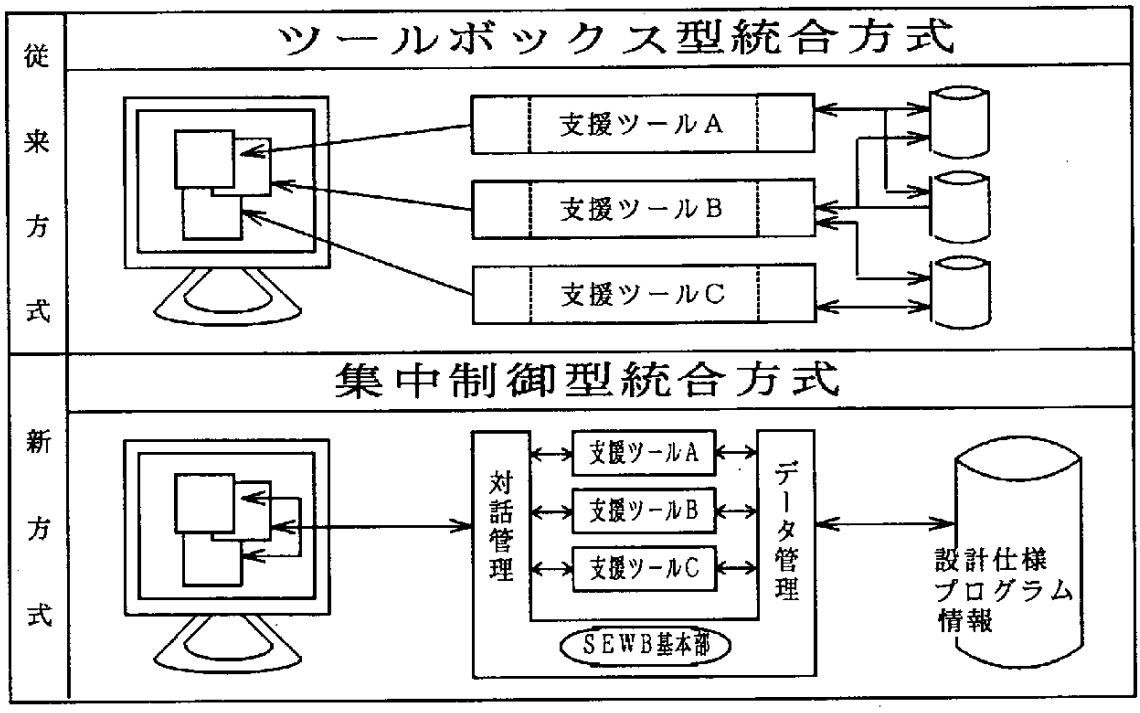


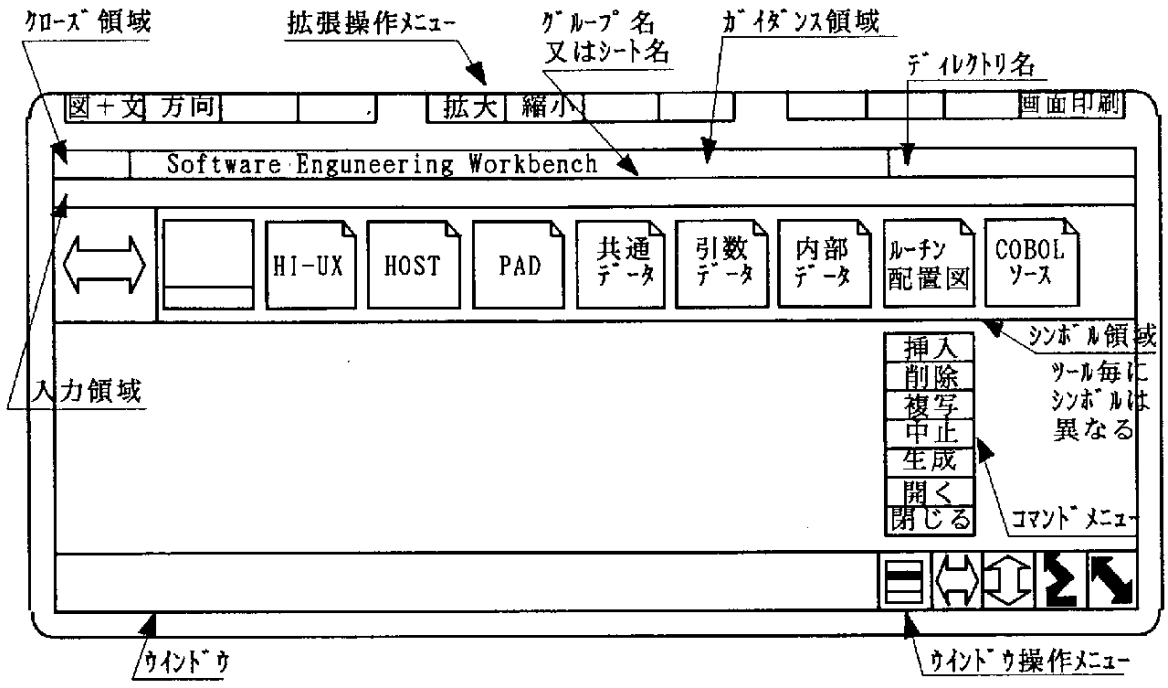
図２．２-１　統合ソフトウェア開発環境

② ユーザインタフェースの統一

ＳＥＷＢ上に搭載されている各種のツールのユーザインタフェースは統一的に定められており、これによりツール毎に操作法がまちまちで操作法を習得するのが大変といった従来の問題を解決した。

図２．２-２はＳＥＷＢ画面レイアウトの標準規格である。コマンドの入力領域，ガイダンス領域、シンボルの表示領域の位置、サイズ等が各ツール間で統一されている。ＳＥＷＢのコマンドは、ポップアップメニュー上で選択する方式をとっている。このコマンドはツール間で統一されており、メニューに表示されるコマンドの内容は、操作状態により必要なものが動的に表示される。

ＳＥＷＢでは仕様書／プログラムをシートと呼び、関連したシートを集めてグループとして管理している。シンボルメニューの▭はグループ，▭はシートを表している。



注）グループ：シートを分類するための入れ物。シート及び他のグループの情報を格納している。
シート　：ソフトウェア情報を構成するファイルの集合でＳＥＷＢにおける最小単位。

図２．２-２　ＳＥＷＢのユーザインタフェース

③　図形を用いたシステム設計／プログラム開発の一貫支援

ＳＥＷＢでは「標準的な図形記法」を用いて、業務処理設計，プログラム設計，テストまでを一貫して支援している。

これらの図形は「構造化分析技法」,「構造化プログラミング技法」等に基づき、国際的にも広く普及している標準記法である。例として、データフロー図に相当する業務処理仕様図ＳＤＦ（Ｓｔｒｕｃｔｕｒｅｄ　Ｄａｔａ　Ｆｌｏｗ　Ｄｉａｇｒａｍ），プログラムフロー図，ＰＡＤ（Ｐｒｏｂｌｅｍ　Ａｎａｌｙｓｉｓ　Ｄｉａｇｒａｍ）などの図式がある。

ＳＤＦはＤｅＭａｒｃｏのＳＡに基づいている。またＰＡＤはプログラムの構造を木構造チャートで表現する記法である。これらの技法をコンピュータで手軽に扱えるようにし、従来、一般的に「読めない」「わかりずらい」といわれた仕様書の標準化を図ることができた。

（2） ＳＥＷＢの種類

ＳＥＷＢではソフトウェア開発のライフサイクルと対応させ、図２．２－３に示す４種類のＳＥＷＢがあり、現在、システム設計者用ＳＥＷＢとプログラマ用ＳＥＷＢを製品化している。

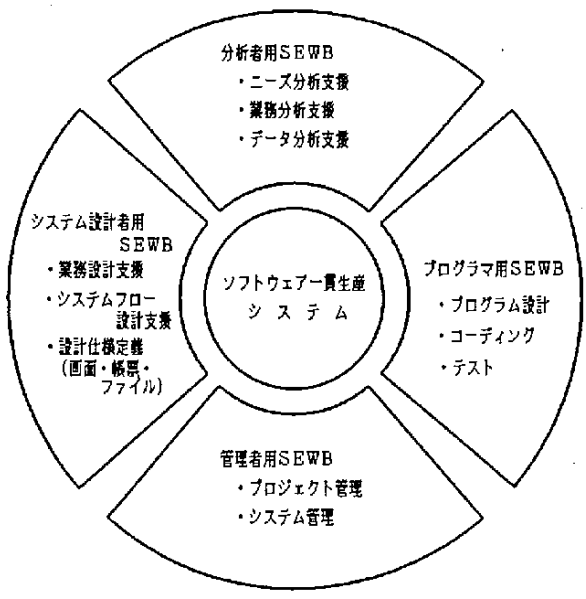


図２．２－３　ＥＷＢの種類

① 分析者用ＳＥＷＢ

業務モデルの作成と分析，データ分析による正規化作業とデータ項目の標準化，データディクショナリへの登録を支援する。

② 設計者用ＳＥＷＢ

システム設計における業務処理仕様の作成，画面や帳票などの仕様決定，オンラインプロトタイプによる業務運用方式の設定等の作業を支援する。

③ プログラマ用ＳＥＷＢ

ソースまたはＰＡＤの様な木構造チャートによるプログラム作成と単体テストを支援する。

④ 管理者用ＳＥＷＢ

工程管理や品質管理などプロジェクト管理者の作業を支援する。

（３） ＳＥＷＢの適用範囲

日立では、ユーザニーズの把握から業務システムの開発・保守までの一貫した作業を標準化し、ＨＩＰＡＣＥとして体系化している。

（ＨＩＰＡＣＥ＝ＨＩｔａｃｈｉ Ｐｈａｓｅｄ Ａｐｐｒｏｃｈ ｆｏｒ Ｈｉｇｈ Ｐｒｏｄｕｃｔｉｖｅ Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍ’ｓ Ｅｎｇｉｎｅｅｒｉｎｇ）

ＨＩＰＡＣＥの工程（フェーズと呼ぶ）では、ＳＥＷＢはシステム計画フェーズからプログラム作成フェーズまでの範囲をサポートしている。図２．２-４にＳＥＷＢの適用範囲を示す。

ホストコンピュータ用業務システムの開発では、

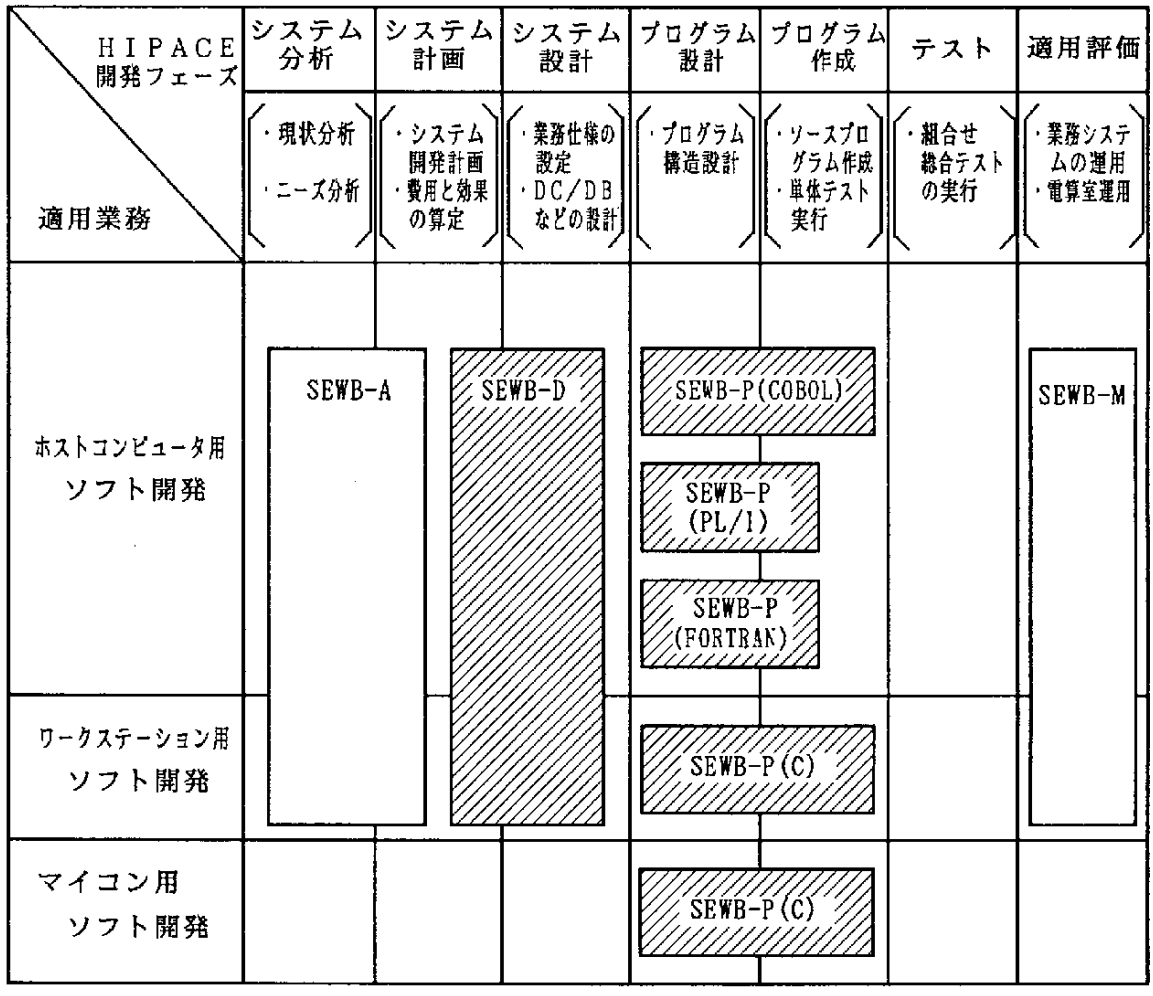
・設計者用ＳＥＷＢ

・プログラマ用ＳＥＷＢ（ＣＯＢＯＬ，ＰＬ／Ｉ，ＦＯＲＴＲＡＮ）

ワークステーション及びマイコン用ソフトウェア開発では

・プログラマ用ＳＥＷＢ（Ｃ）

が利用できる。



▨：製品の対象範囲

SEWB-A：分析者用ＳＥＷＢ

SEWB-D：システム設計者用ＳＥＷＢ

SEWB-P：プログラマ用ＳＥＷＢ

SEWB-M：管理者用ＳＥＷＢ

（　　）内：プログラミング言語

図２．２-４　ＳＥＷＢの適用範囲

２．２．２　ＳＥＷＢを利用したソフトウェア開発

ＳＥＷＢを利用した開発の流れと支援ツールの関連を図２．２－５に示す。

設計者用ＳＥＷＢで定義された各種の設計情報は、ホストコンピュータの開発支援ＥＡＧＬＥで管理され、それらを基にプログラムの自動生成を行っている。

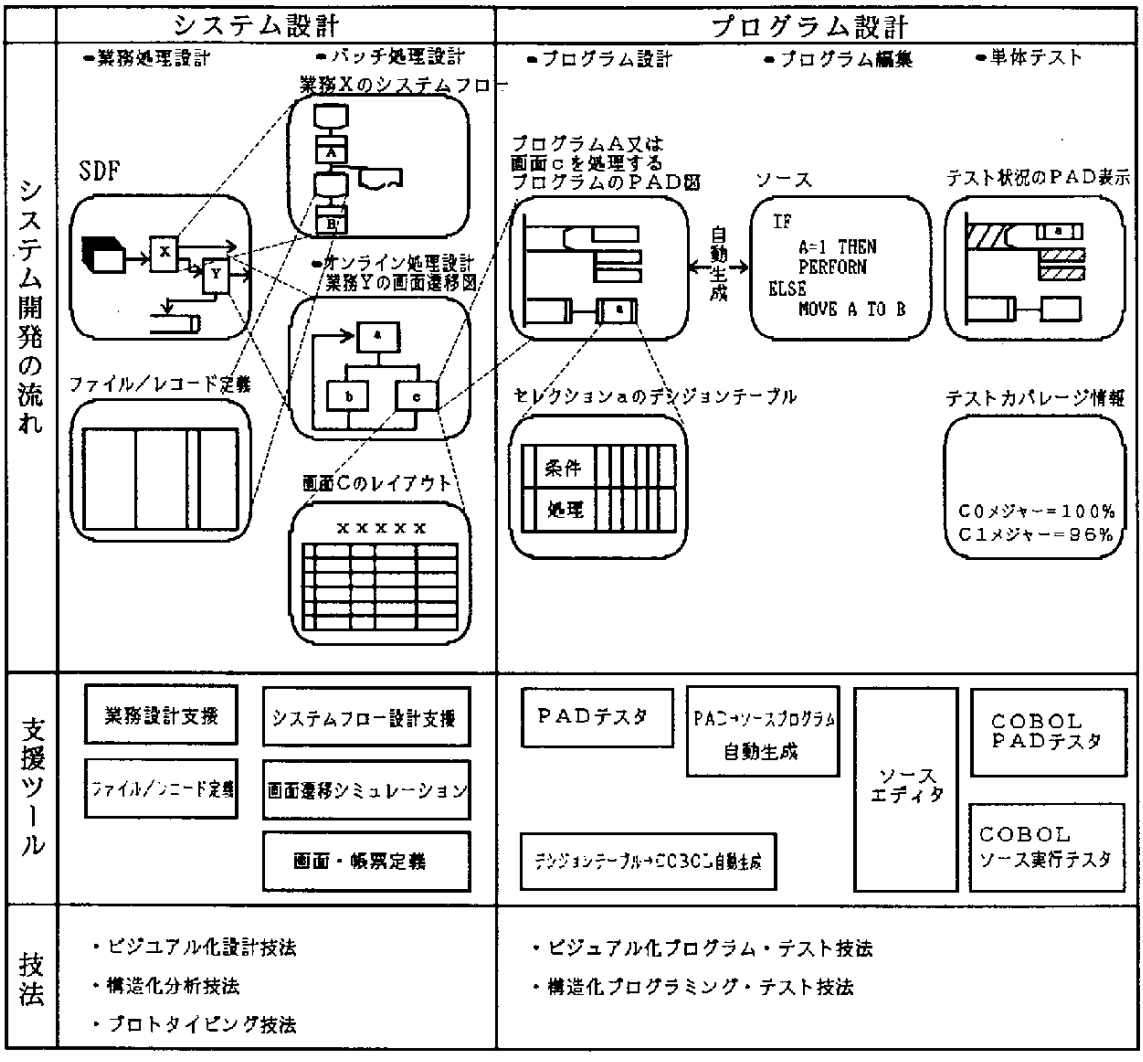


図２．２－５　ＳＥＷＢを利用したソフトウェア開発

（1） 設計者用ＳＥＷＢ

① 業務設計支援

システム設計技法のひとつに業務処理仕様図ＳＤＦがある。ＳＤＦとは構造化分析に基づき、データの流れに着目して業務処理仕様を記述する技法であり、次の様な特長がある。

・「もの」（データ，ファイル）と「処理」に区分したモデル表現

・トップタウンによる階層毎の分析

ＳＤＦを利用してシステム分析・設計を行う効果としては、

・エンドユーザ，システムエンジニア向のコミュニケーションの向上

・システム分析品質の向上（業務機能のレビューと設計不良の早期発見による）

があげられる。

ＳＥＷＢではＳＤＦの作成･編集を支援し、アイコン･マウスを利用し簡単に描画できる。

ＳＤＦの編集例を図２．２－６に示す。

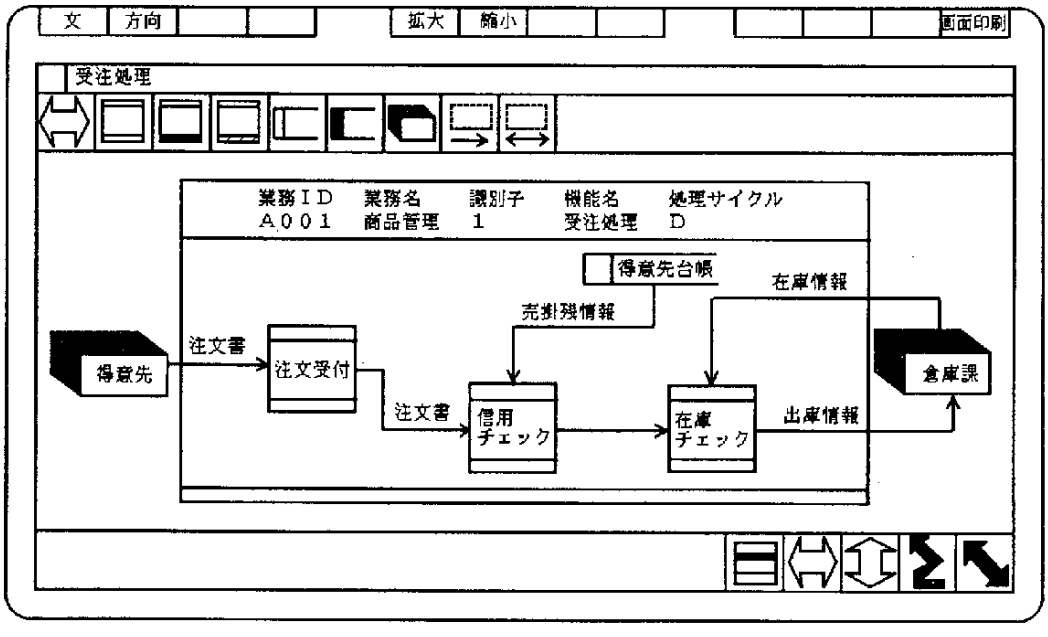
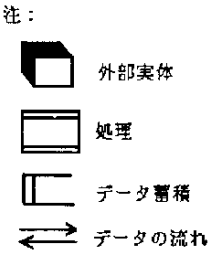


図２．２－６　ＳＤＦの編集例



② システムフロー設計支援

バッチ処理設計を支援する図式にプログラム処理フローを示すシステムフローがある。システムフローは対象業務のプログラム間の関連，プログラムとファイルとの関連と処理手順を表したものである。ＳＥＷＢではＳＤＦで記述される最下位のレベルの機械化対象の処理が、システムフローに展開される。

ＳＤＦで記述された業務処理の内容を解析し、あらかじめ登録してあるシステムフロー部品を用いてシステムフローを自動生成している。

③ 画面帳票定義

オンラインシステムの端末用画面，帳票レイアウトを対話形式で定義する。定義された画面や帳票はテスト表示ができ、エンドユーザと仕様確認を行うことができる。

レイアウトはまずラフスケッチを行い、その結果から各フィールドの詳細情報を入力し、エンドユーザの要求に応じて編集する。

④ 画面シミュレータ

画面・帳票定義で作成した画面・帳票をユーザの設定した遷移図に従って遷移させ、オンラインシステムの実際の動きをワークステーションでシミュレートすることができる。

画面遷移図をエディタで作成した後、各画面間の遷移条件を記述する。遷移条件は、簡単な記法を用い、ＰＦキー操作やデータ入力条件等を定義する。

⑤ ファイル／レコード定義

ファイル属性やレコード仕様を定義する。

作成したレコード定義は、ホストコンピュータに転送しＣＯＢＯＬプログラムで再利用できる。

（2） プログラマ用ＳＥＷＢ

① ＰＡＤエディタ

ＰＡＤは構造化プログラム技法に最適な図式であり、ＳＥＷＢでは連接，選択，反復の三つの基本ボックスを含め、プログラミングに必要な１０種のボックスを提供している。

また、日本語ＰＡＤエディタ（図２．２-７）では、構文を使った入力方式を支援している。構文はユーザが定義することができ、ＰＡＤ編集時に画面に一覧表を表示する。ユーザは、使用したい構文を選択し、構文中の仮記号に対して数値，文字を代入し、ＰＡＤを完成する。

日本語ＰＡＤからプログラム変更を行うと自動的に、構文に従ってＣＯＢＯＬソースプログラムが生成される。

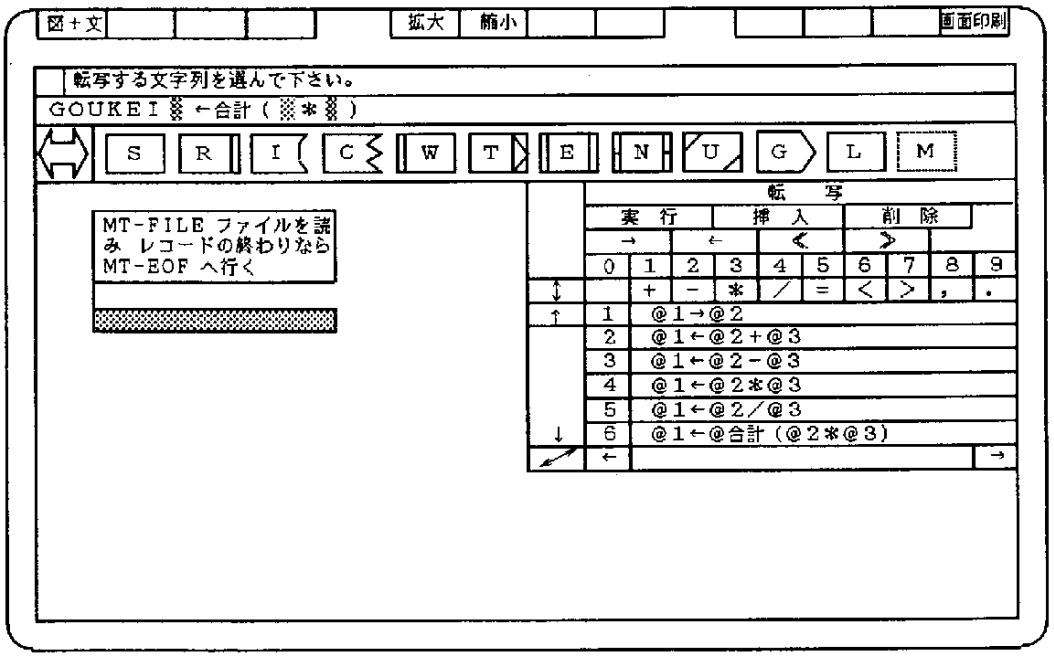


図２．２-７ 日本語ＰＡＤの編集例

② ＰＡＤテスタ

ＰＡＤによるテストを支援するもので実行したボックスの色を変えたり、変数の内容を実行に合わせて表示して、テスト状況を確認できる。

ボックス毎の実行・数の表示やテストカバレージ情報の表示など豊富な機能を持つ。

ＰＡＤテスタはＣＯＢＯＬとＣ言語用を製品化している。

その他のプログラマ用ＳＥＷＢのツール一覧を表２．２－１に示す。

表２．２－１　プログラマ用ＳＥＷＢ

日　本　語　名	略　　名	機　　能　　概　　要
ＰＡＤ⟷ＣＯＢＯＬ自動生成	ＳＥＷＢ／ＣＢＬ	ＰＡＤからホスト用ＣＯＢＯＬソースを自動生成。ソースからＰＡＤを自動生成。
デシジョンテーブル ＣＯＢＯＬ自動生成	ＳＥＷＢ／ＤＴＣＢＬ	デシジョンテーブルの作成，ＣＯＢＯＬソースの自動生成。
ＣＯＢＯＬソース⟶実行テスタ	ＳＥＷＢ／ＣＵＰＳ	ホスト用ＣＯＢＯＬソースを解釈し実行。ソース上に実行状況表示。
ＰＡＤ⟷ＦＯＲＴＲＡＮ自動生成	ＳＥＷＢ／ＦＯＲ	ＰＡＤからホスト用のＦＯＲＴＲＡＮソースを自動生成。ソースからＰＡＤを自動生成。

２．２．３　ＳＥＷＢの効果

システム開発ライフサイクルでのシステム開発効率（システム設計からテストまで）は、支援ツールの適用率，部品の登録数と再利用率，プロジェクト体制（管理方式や要員の質）に影響を受ける。従来に比べ２倍から３倍以上も生産性が上がったプロジェクトがある反面、期待通りの効果が出なかった例もある。社内複数プロジェクトの実績では、設計者用ＳＥＷＢの実績では３０～５０％の効果が上がっている。

生産性向上の理由として次の事があげられる。

・ホストの開発支援ツールに比べ、環境作りが容易。

・操作生が良い。新人でも短期間で習得できる。

・応答が早い。モデルケースではホストコンピュータの会話処理に比べて、１／３～１／６０の応答時間が短縮した。

２．２．４　今後のソフトウェア生産技術

業務システムにおけるソフトウェアの規模は今後も増大すると予測される。ソフトウェアの生産性と品質を向上するため、今後もＣＡＳＥの機能拡充を図る必要がある。特にソフトウェア開発サイクルの全工程を支援するＩ－ＣＡＳＥ（Ｉｎｔｅｇｒａｔｅｄ－ＣＡＳＥ）が重要になる。

Ｉ－ＣＡＳＥを実現するには、次のような技術開発が必要になるであろう。

・オープンアーキテクチャを実現したＣＡＳＥプラットフォーム

・リポジトリによる情報資源の統合管理

・戦略情報システム計画・立案の支援を含む上流工程支援の充実

・地域分散開発に対応した開発ネットワーク

・チームによる共同開発を支援するグループウェア技術

・ＡＩを応用した設計支援

・開発済ソフトウェアを再活用するためのリエンジニアリング技術

・プログラム開発作業を大幅に削減するプログラム自動化技術

2.3　NATURAL上での統合型CASE

ソフトウェア・エージーは、COBOL等をベースとした第3世代のアプリケーション開発に対して、第4世代開発ツールNATURALを中心とした第4世代テクノロジを提唱してきた。NATURALは、次のような概念に基づいており、その基本はCASEの考え方と一致している。

- 高度会話型開発環境の提供
  - 会話型画面／レポート定義
  - 会話型プログラミング
  - 会話型テスト
- プロトタイピング・アプローチの採用
- ディクショナリとの統合
  - ディクショナリ主導型システム開発
  - 自動ドキュメンテーション
- ハードウェア、OS、TPなど環境からの独立
  - 環境に依存しない共通のアプリケーション開発

CASEが目指すところの「開発環境の支援・自動化」「高い生産性」「ユーザ満足度の高いシステム構築」のためには、バッチ処理しかなかった時代の産物であるCOBOLをベースにしたCASEツールでは不十分であり、上記のような基本概念に基づいた言語および開発ツールがまずベースとして必要である。

このNATURALを中心として、要求分析・設計段階を中心として開発過程全般を支援するCASEとして、PREDICT　CASE、下流CASEとしてアプリケーション・ジェネレータ　NATURAL　CONSTRUCTを提供する。CASEツールを支えるCASEリポジトリとしてオブジェクト指向E－Rデータベースを使用する。当社ではCASEリポジトリを開発データベース（DDB）と称し、リレーショナル型DBMSを拡張したE／RモデルDBMS　ADABAS　ENTIREがその維持管理を行う。

これらプロダクトに加え、プログラム作成および本番環境を支援する統合ディクショナリPREDICTと共に、統合型CASE（I－CASE）環境を提供する（図2.3－1）。

以降に各プロダクトの概要を記述する。

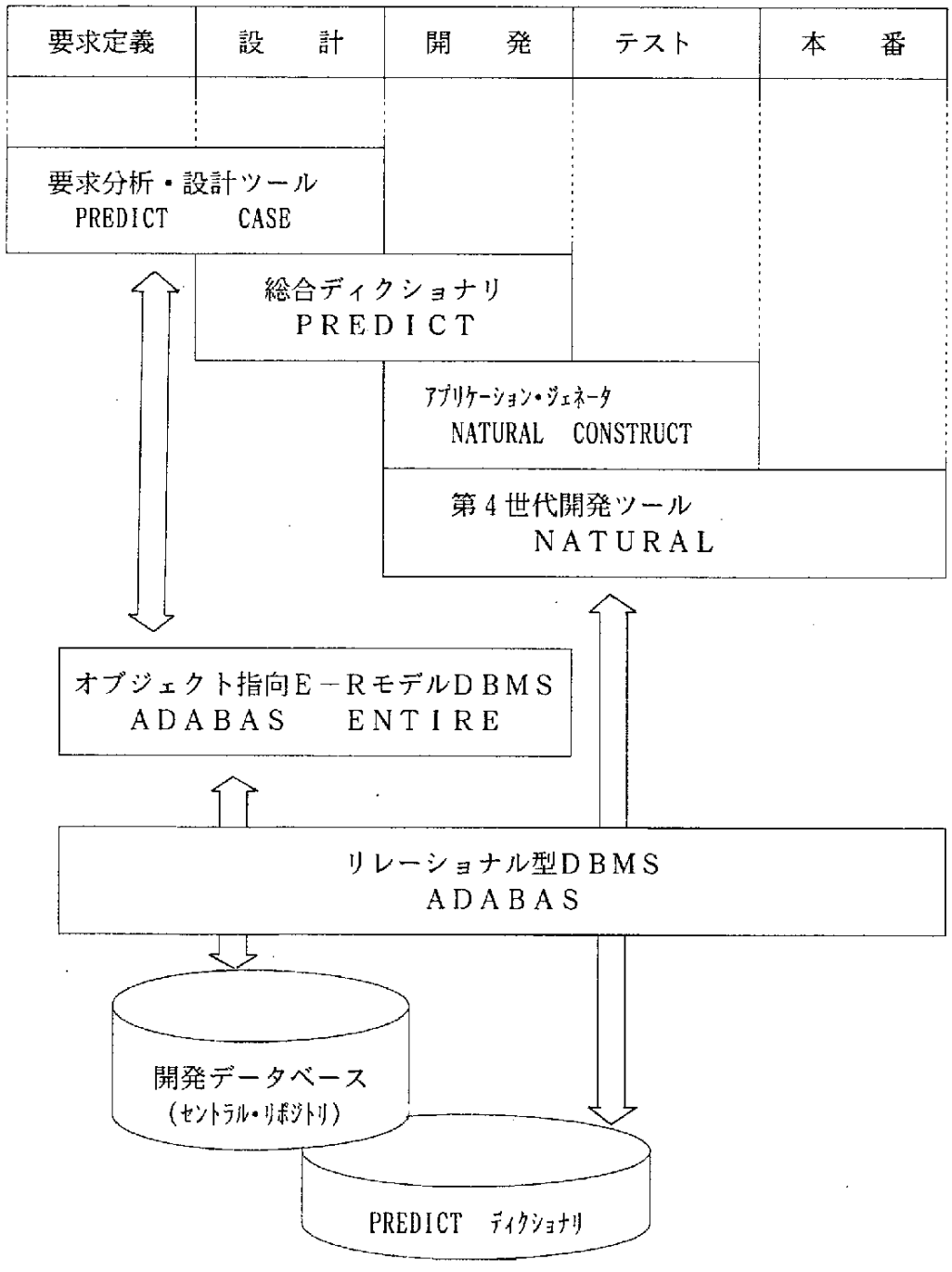


図2.3－1　ＣＡＳＥとその関連プロダクト

2.3.1　ＮＡＴＵＲＡＬ

当社ＣＡＳＥツールのベースであるＮＡＴＵＲＡＬについて簡単に説明する。

ＮＡＴＵＲＡＬはこれまで「第四世代言語」として紹介されてきた。このため、単なるプログラミング言語の一つであると解釈されがちであるが、実際はアプリケーション開発に必要な

ツールが一つの統合化されたシステムとしてまとめられた「第四世代アプリケーション開発システム」と位置付けられる。

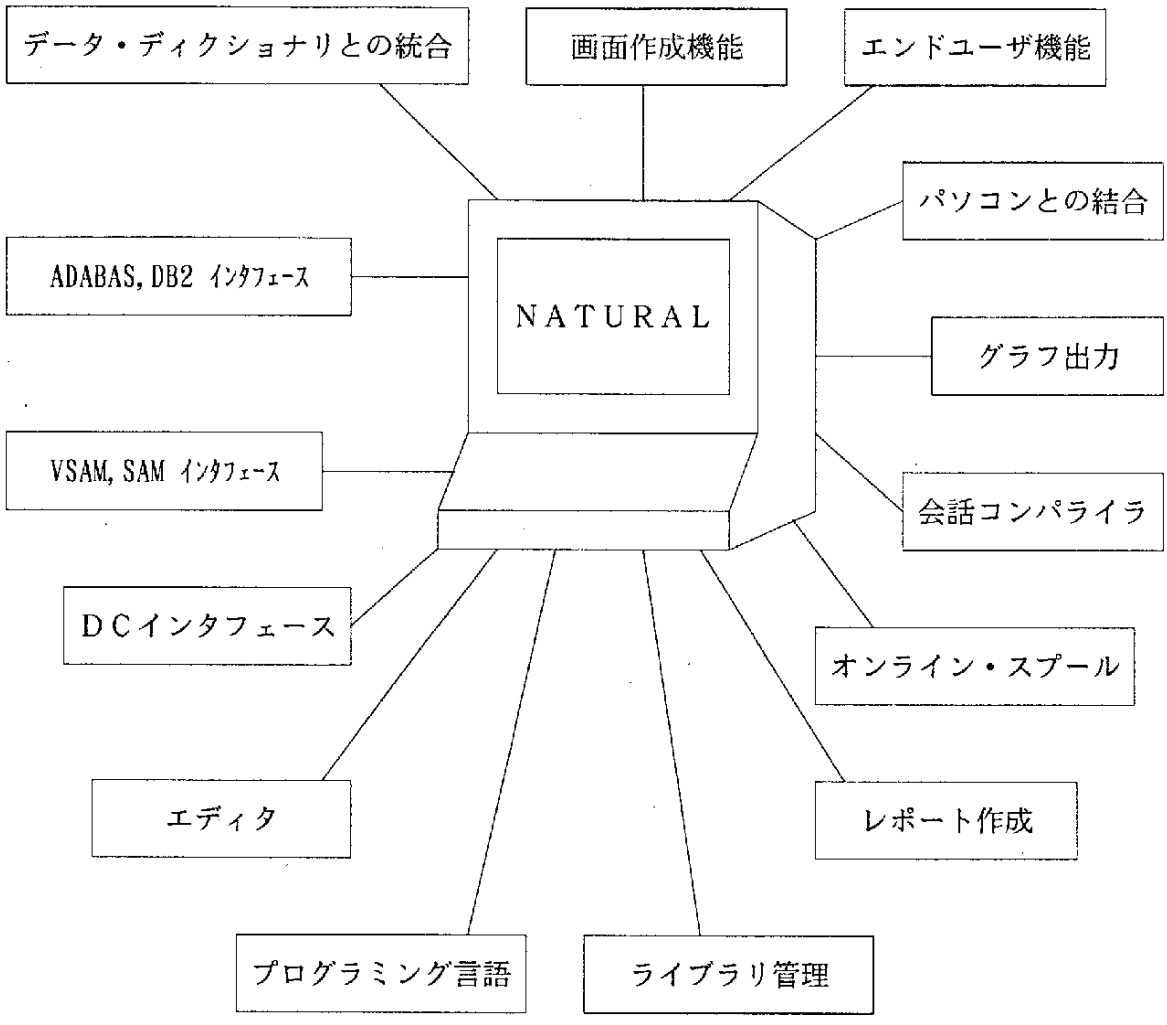


図2.3－2　統合化システム－NATURAL

NATURALは、アドホックな問い合せから高度なアプリケーション・システム開発まで広範な分野をカバーする。図2.3－2に示すように、エディタ、プログラミング言語、画面作成、コンパイラ、データベース・インタフェース、DCインタフェース等アプリケーション開発に必要な機能をすべて含んでいる。対象ユーザもエンドユーザからシステム部門のSE、プログラマまで様々な知識レベルにまたがる。各部門の人が各業務に、NATURALという共通の手段を使うことにより、プロトタイプの作成から本番への移行まで一つのツールで統一的に行える。

このため、従来のようにバラバラに提供される様々なツールを使用するために生じる負荷は

大幅に削減される。つまり、統一性や互換性のないツールをいかに使うかに煩わされることなく、アプリケーション開発そのものに専念できる。

従来の方式では、プログラムのリンクエディットや画面定義などバッチ方式で行わなければならず、アプリケーション開発の途中に必ず「待ち」があり、思考が中断されてしまう。このような状況では、開発の各段階でエンドユーザが介入することも困難であった。

ＮＡＴＵＲＡＬを使用すればすべて作業を会話型で継続して行え、直ちに結果を目で見て確認できる。ユーザが各段階で参画でき生産性も上がる。

さらに、ＮＡＴＵＲＡＬプログラムでは動作環境を全く意識する必要はない。したがって、例えばＴＳＳ下でプロトタイプの作成、試行、変更を繰り返し、本番は各種オンライン環境(COM-PLETE、IMS/DC、CICS、AIM/DC、ADM/DC、TMS 、DCCM等) 下で何の変更手続きもなしに運用することが可能である。この「可能性」という点も大きな特長である。

また、ディクショナリと深く結びついており、データ項目定義やチェック・ルールなどはディクショナリから自動的にプログラムに取り込まれる。プログラムと使用ファイル／データ項目、画面マップ、サブルーチン等のクロスリファレンス情報もディクショナリに自動ドキュメントされる。

2.3.2　要求分析・設計ツール

(1)　ＰＲＥＤＩＣＴ　ＣＡＳＥとは

ＰＲＥＤＩＣＴ　ＣＡＳＥは業種を問わず広範囲のアプリケーション開発プロジェクトに適用できるＣＡＳＥツールであり、大勢の人が同時に開発する大規模なアプリケーションに対応するため、内部的に各登録情報のステータスを保持し、一貫性を保っている。

情報システム部門だけでなく担当部門をアプリケーション・システムの開発に参画させることができ、アプリケーション開発過程における担当者と開発者との考え方の相違からくるリスクを最小限にすることができる。

ＰＲＥＤＩＣＴ　ＣＡＳＥは、ＡＤＡＢＡＳ、ＡＤＡＢＡＳＥＮＴＩＲＥ、ＮＡＴＵＲＡＬおよびＰＲＥＤＩＣＴをベースに作られている。

ＰＲＥＤＩＣＴ　ＣＡＳＥ自身もＮＡＴＵＲＡＬアプリケーションであり、データはすべてＡＤＡＢＡＳに格納される。ＰＲＥＤＩＣＴ　ＣＡＳＥのデータ構造はエンティティ・リレーションシップ・モデル（Ｅ・Ｒモデル）がベースであり、ＡＤＡＢＡＳＥＮＴＩＲＥを使用し

ている。

データに関しては、ＰＲＥＤＩＣＴ　ＣＡＳＥのスキーマ・ジェネレータ機能により、ＰＲＥＤＩＣＴ　ＣＡＳＥとＰＲＥＤＩＣＴが関連付けられる。スキーマ・ジェネレータは、ＰＲＥＤＩＣＴ　ＣＡＳＥのアプリケーション・データ・モデルから論理データベース・システム（ＡＤＡＢＡＳ定義）を生成する。

アプリケーション・システムに関しては、開発データがＰＲＥＤＩＣＴ　ＣＡＳＥに格納され、本番データがＰＲＥＤＩＣＴに格納される。

同様な関係が機能（ファンクション）についても存在し、ＰＲＥＤＩＣＴ　ＣＡＳＥに定義された機能構造を用いてアプリケーション・システムのＮＡＴＵＲＡＬオブジェクトがコンパイルされる。したがって、マップ、プログラム、コピーコード、ルールはすべて機能（ファンクション）として定義される。

⑵　ＰＲＥＤＩＣＴ　ＣＡＳＥの考え方

①　開発過程

ＰＲＥＤＩＣＴ　ＣＡＳＥにおいて開発過程とはアプリケーション・システムのライフサイクル全体と把える。すなわち、要求が起こってから、設計、開発、テスト、運用、および保守に至る、システムが消滅するまでを扱う。

ＰＲＥＤＩＣＴ　ＣＡＳＥを利用することにより、現場担当とシステム開発者が開発過程において共通の道具となるため、コミュニケーションや理解の向上を計ることができる。

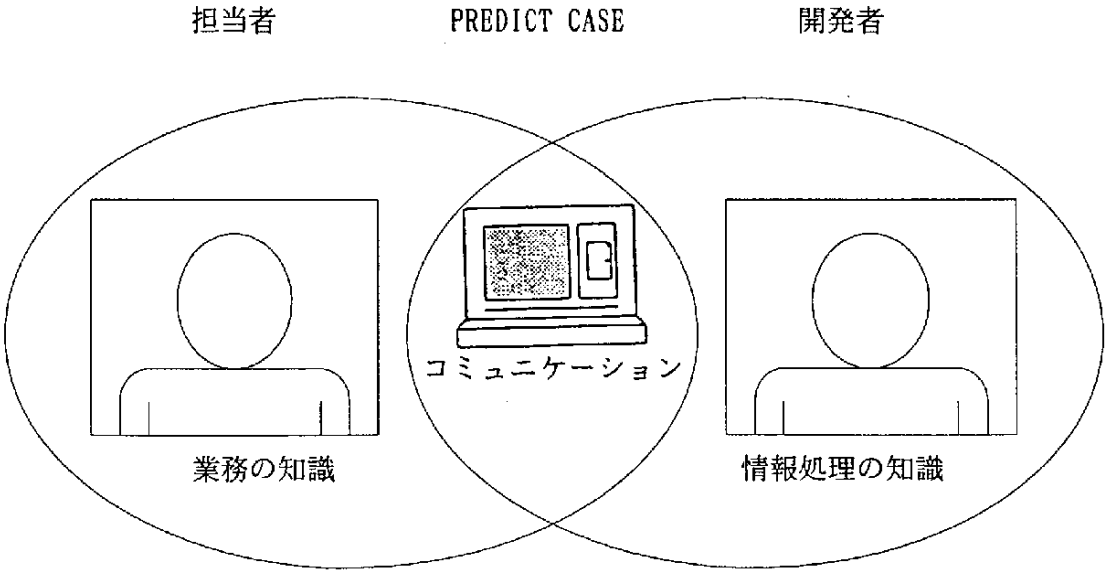


図2.3－3　PREDICT CASEによるコミュニケーションの向上

② アプリケーション・システムの品質

アプリケーション・システムの品質は、要求定義、設計（モデル）、プログラム自身およびドキュメンテーションの品質に依存する。設計（モデル）やプログラムについては柔軟性や使い易さ、効率、機能の満足度、保守のし易さ、可搬性など重要てある。ドキュメンテーションについては、変更し易いこと、現実を反映していること（アップ・ツー・デート）、あいまいさがないこと、などが重要となる。

ＰＲＥＤＩＣＴ　ＣＡＳＥは開発過程全般のドキュメントの標準化を計り、またＰＲＥＤＩＣＴ　ＣＡＳＥに登録された情報間の過不足や矛盾を自動チェックする。

⑶ ＰＲＥＤＩＣＴ　ＣＡＳＥの手法

ＰＲＥＤＩＣＴ　ＣＡＳＥは、ＩＳＯＴＥＣ（Integrated Software Technology）手法とＮＡＴＵＲＡＬ環境とを組み合わせて開発されており、プロジェクト定義からプログラム作成までの統合化された解決策を提供する。

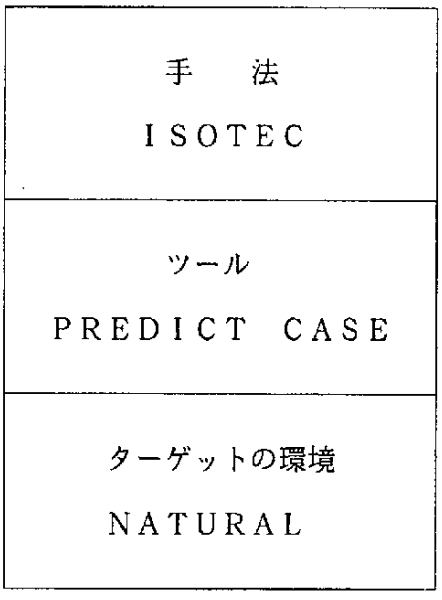


図2.３－４　統合された解決策

ＩＳＯＴＥＣは、西独のコンサルテーション会社、ＥＤＶ　Studio Ploenzce のシステム・デザイン手法であり、特に開発の早期段階に有効があり、情報システム部門にもエンドユーザ部門にもわかりやすい手法である。

当手法はソフトウェア開発過程全体を支援する。

ＰＲＥＤＩＣＴ　ＣＡＳＥによる手法とツールの統合化により、高度なユーザ・フレンドリ性と有効性を可能とする。以下のような利点をあげることができる。

・品質保証の自動化

・開発データと本番データ間の一貫性

・用語の統一

① ＰＲＥＤＩＣＴ　ＣＡＳＥの構成要素

ＰＲＥＤＩＣＴ　ＣＡＳＥは、次の要素から成っている。

・手　法：ライフサイクルの考え方が適用される。

・ツール：中央の開発データベース内の開発データを保持し、使用する。

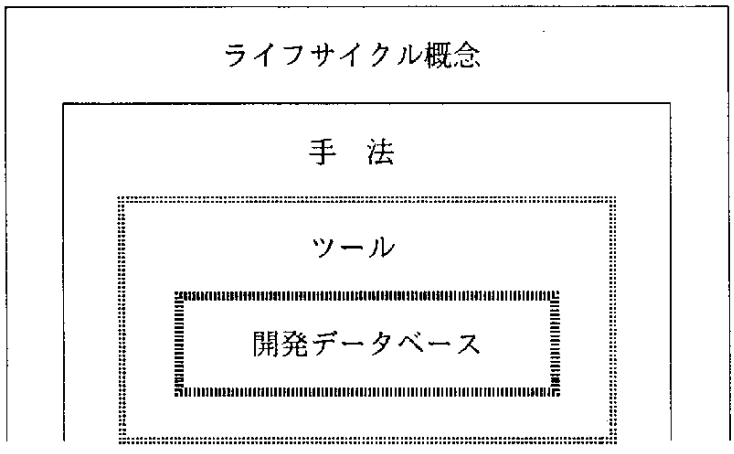


図2.3－5　ＰＲＥＤＩＣＴ　ＣＡＳＥの構成要素

② 手　　法（ＩＳＯＴＥＣ）

アプリケーション・システムは、データ、機能、システム機能のモデリングを組合せて、以下の手法を用いながら設計する。

・情報構造分析（ＩＳＡ：Informatin Structure Analysis ）

データ・モデルは「情報構造」と呼ぶ。情報構造では、ビジネスの世界での関連する対象、その属性および関係（リレーションシップ）を表現する。情報構造は手法「情報構造分析」の結果であり、当手法はＰ．チェンのエンティティ・リレーションシップ・アプローチにもとづいており、エンティティの汎化やインテグリティ・ルール等

の拡張機能も含んでいる。

・機能構造分析（ＦＳＡ：Function Structure Analysis ）

機能構造分析の目的は、ビジネス機能モデルを作成することである。ＦＳＡは、ビジネス機能をサブ機能に分け、機能間および外部とのやりとりを明確化する。

・システム機能の構築と仕様（ＳＳＦ：Structuring and Specification of System Functions）

機能構造分析で決定したビジネス機能をＤＰシステムが支援する方法を記述する。

③ 開発データベース（ＣＡＳＥリポジトリ）

ＰＲＥＤＩＣＴ　ＣＡＳＥは中央にある開発データベース（ＤＤＢ）にもとづいている。すべての開発過程の結果はＰＲＥＤＩＣＴ　ＣＡＳＥにより自動的にドキュメントされ、開発データベース内にオブジェクトとして格納される。

図2.3－6にＰＲＥＤＩＣＴ　ＣＡＳＥメタ構造、すなわちＰＲＥＤＩＣＴ　ＣＡＳＥの全オブジェクト・タイプと関連を示す。

オブジェクト・タイプ間の関連名は次のように読み取る。

・左から右へ

・２つのオブジェクト・タイプが上下にあるときは、上から下へ

・輪になった再帰関連は右まわりに

関連のタイプは矢印で示している。

――――▷	オプションの関連	・ゼロまたは１。
――――▷▷	オプションの関連	・ゼロまたは１以上。
――――▶	必須の関連	１オブジェクトが必ず存在。
――――▶▶	必須の関連	少なくとも１オブジェクトは存在。

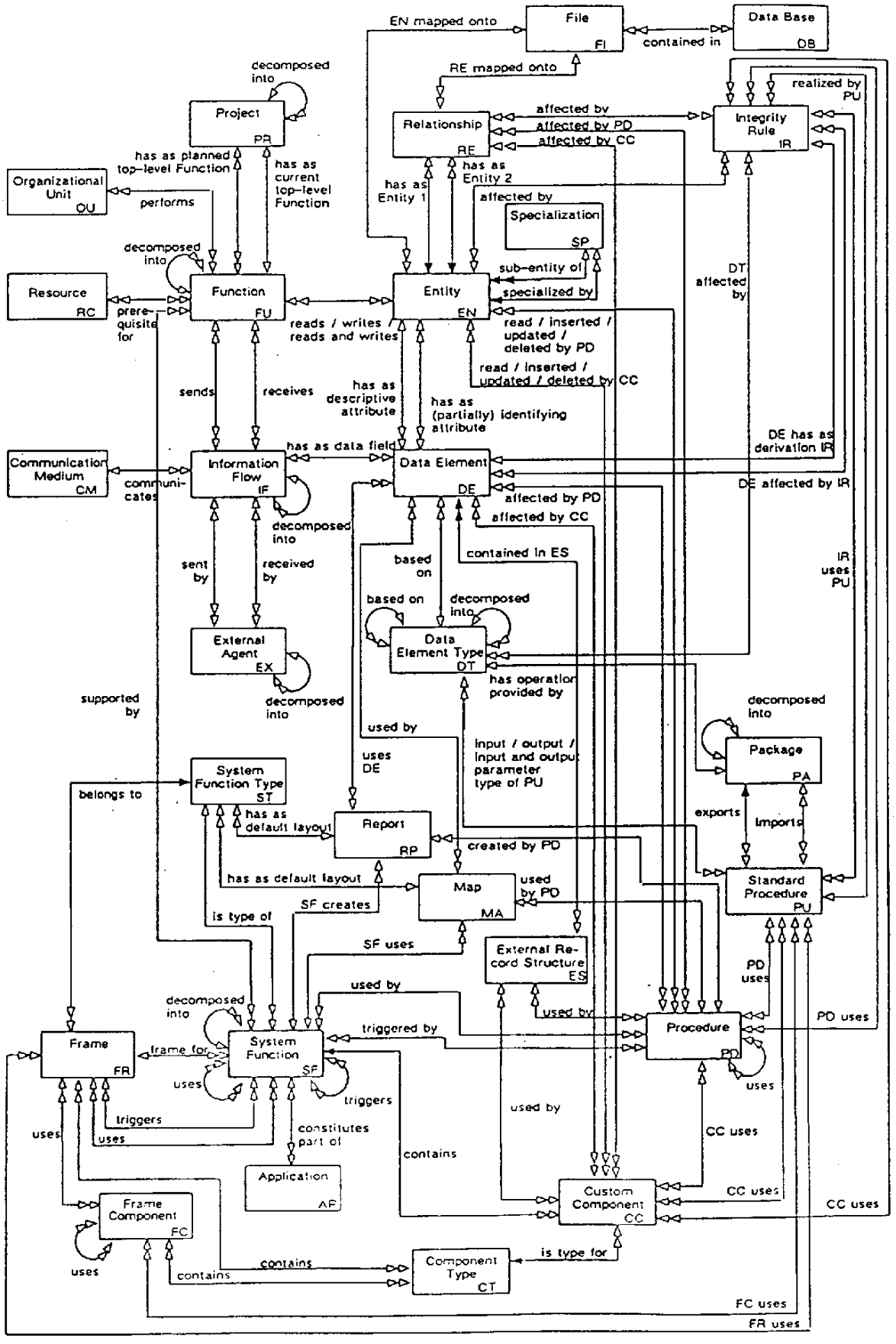


図2.3－6　PREDICT　CASEメタ構造、（DDB:CASEリポジトリの構造）

④ ツール

ＰＲＥＤＩＣＴ　ＣＡＳＥは開発結果を保持し、評価するために、さまざまなタイプの情報を格納・管理するツールを各種提供している。

オブジェクト指向の作業を可能とするためのＰＲＥＤＩＣＴ　ＣＡＳＥワークステーション機能を提供しており、開発データを図形で描くことができる。

ＰＲＥＤＩＣＴ　ＣＡＳＥホスト機能は、開発結果の維持管理、検索、品質保証を行うユーザフレンドリなインタフェースを提供する。評価レポートの他、プログラム構築、スキーマ生成等の機能も提供する。これらは、開発データベース内に格納された開発データから本番データを作成するものである。

(4) ソフトウェアの本番環境

ＰＲＥＤＩＣＴ　ＣＡＳＥは、ＮＡＴＵＲＡＬおよびＰＲＥＤＩＣＴと共に統合されたソフトウェア本番環境を実現する。アプリケーション・システムの開発はすべて完全に当ツールの制御下で行われる。

- ＰＲＥＤＩＣＴ　ＣＡＳＥは開発データと本番データ間の一貫性を保証する。
- 開発環境における機能はすべてＰＲＥＤＩＣＴ　ＣＡＳＥと共に使用され、ユーザはその環境内で開発作業を行う。

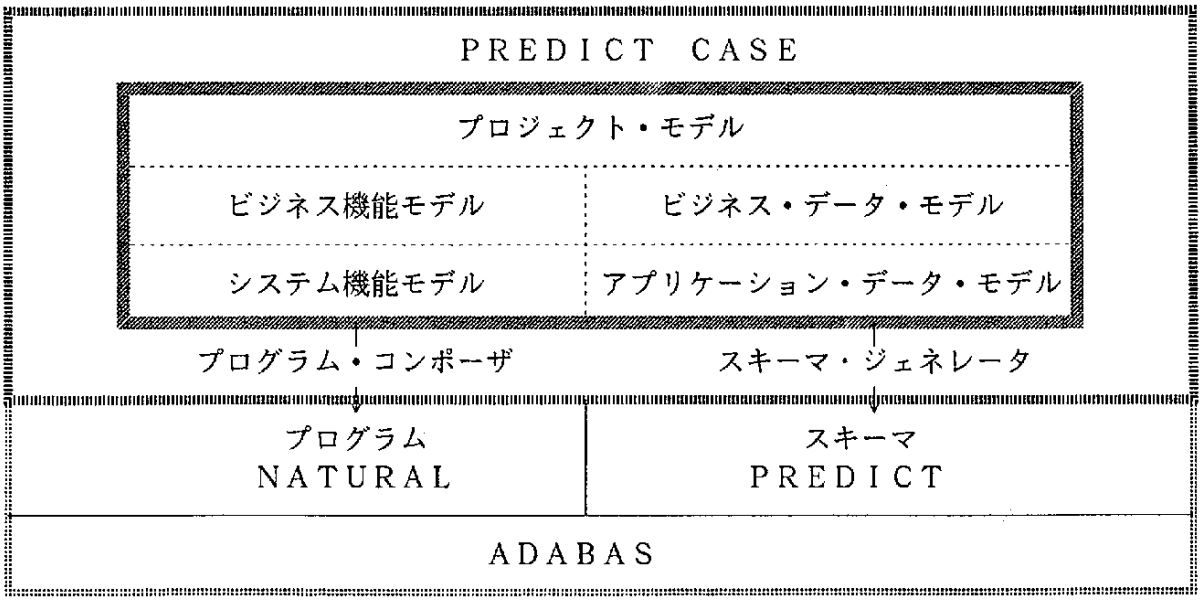


図2.3－7　ソフトウェアの本番環境

① 開発データおよび本番データ

機能モデルおよびデータベース・モデルは開発データベース内に格納される。ビジネス・モデルを支援するシステム機能からＮＡＴＵＲＡＬプログラムを作成する。システム機能はビジネス・データ・モデルの一部、つまりアプリケーション・データ・モデルのデータのみを扱う。ＮＡＴＵＲＡＬプログラムはアプリケーション・データ・モデルから生成されＰＲＥＤＩＣＴに格納されているデータベース・スキーマをアクセスする。

データベース・スキーマおよび実行可能プログラムを本番データと呼ぶ。ＰＲＥＤＩＣＴ ＣＡＳＥから生成された本番データと同様に、開発データも互いの関連付けがされＰＲＥＤＩＣＴ ＣＡＳＥが制御する。アプリケーション・システム内の変更は開発データの変更と本番データの再生成により行う。本番データを直接変更することはできず、直接変更しようとすると品質保証機能が自動的にチェックを行う。

② ＮＡＴＵＲＡＬ環境における機能性

ＮＡＴＵＲＡＬの機能はＰＲＥＤＩＣＴ ＣＡＳＥと完全に統合化されている。

- システム機能を構成するビジネス論理単位の仕様はすべて言語ＮＡＴＵＲＡＬを用いて表す。
- ＮＡＴＵＲＡＬエディタを用いて画面マップ、レポートおよびデータ・エリアの指定を行う。
- ＰＲＥＤＩＣＴ ＣＡＳＥオブジェクトの機能は、プログラム・コンポーザにより集められる。
- プログラム組立て時に構文エラーがみつかると、そのＰＲＥＤＩＣＴ ＣＡＳＥオブジェクト内にエラーの旨が示される。できあがったプログラムはＰＲＥＤＩＣＴ ＣＡＳＥから直接テストすることができる。

(5) ＰＲＥＤＩＣＴ ＣＡＳＥの機能

ＰＲＥＤＩＣＴ ＣＡＳＥの機能は以下のサブシステムに分かれている。

- データ入力および更新機能

  オブジェクトを入力、変更、削除および（一部）コピーする。

- 選択機能

  品質保証やレポート機能に受け渡す集合を作成する。

- 品質保証

開発データベース内の矛盾や手法内のルールに対する違反をみつける。

ＰＲＥＤＩＣＴ　ＣＡＳＥの主要な機能要素は次のとおりである。

・ＰＲＥＤＩＣＴ　ＣＡＳＥスキーマ・ジェネレータ

ＰＲＥＤＩＣＴ　ＣＡＳＥ内にドキュメントされたアプリケーション・データ・モデルをＰＲＥＤＩＣＴのファイル／フィールド記述に変更する。

・ＰＲＥＤＩＣＴ　ＣＡＳＥナビゲータ

データ更新のためのナビゲーションを行うエディタ。

・ＰＲＥＤＩＣＴ　ＣＡＳＥプログラム・コンポーザ

ＰＲＥＤＩＣＴ　ＣＡＳＥに指定されたシステム機能から実行可能なコードを生成する。

・ＰＲＥＤＩＣＴ　ＣＡＳＥドキュメンテーション・ジェネレータ

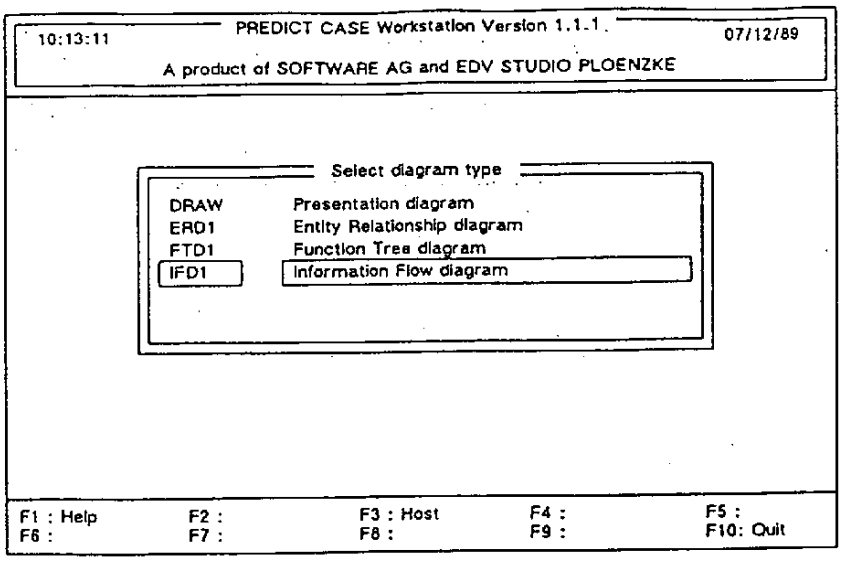
レポート機能であり、典型的なドキュメンテーション・モデルを提供し、簡単に一貫性のあるドキュメントを作成できる。

・ＰＲＥＤＩＣＴ　ＣＡＳＥワークステーション

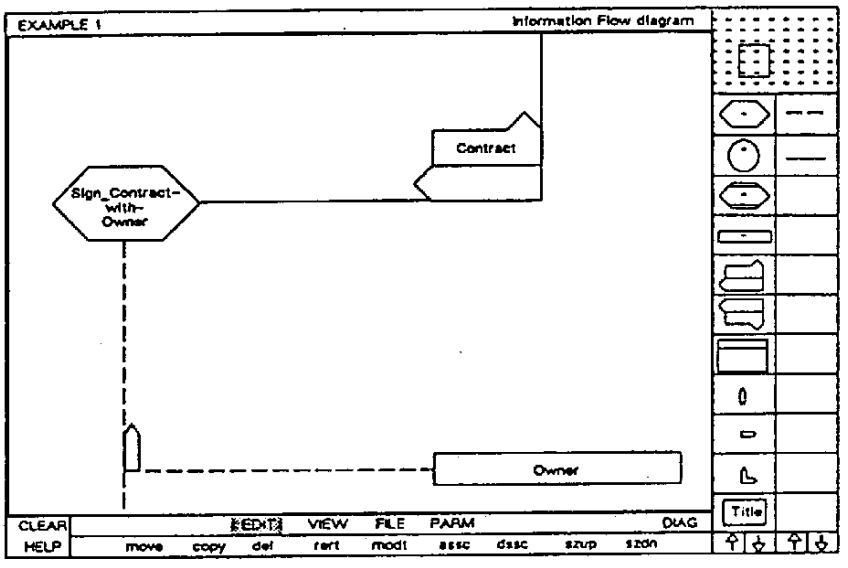
ＰＲＥＤＩＣＴ　ＣＡＳＥ開発データベース内にドキュメントされたオブジェクト間の関連を視覚化し、維持するためのＰＣツール。

データをホスト・コンピュータからダウンロードし、図内に情報を転送することができる。ＰＣ上で図を作成したり、編集し、ホスト・コンピュータにアップロードできる。

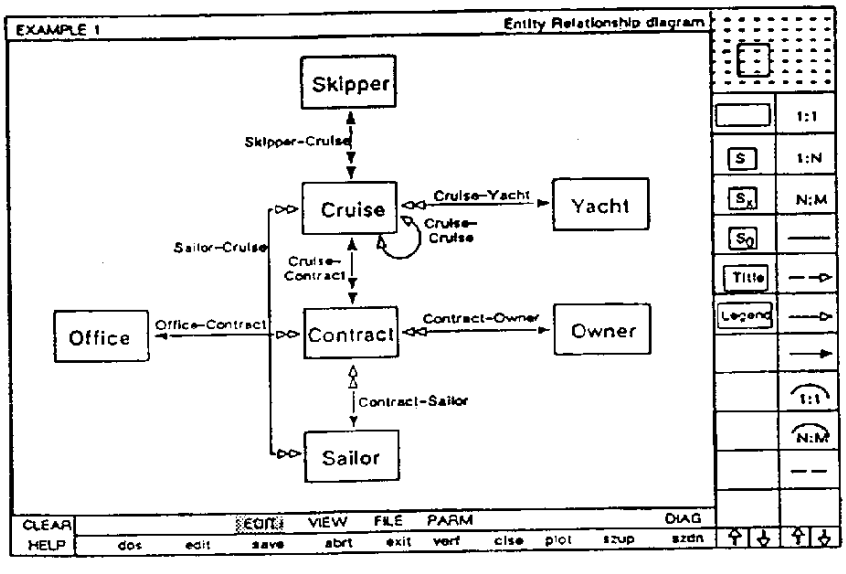
図2.3－8にワークステーションの画面例を示す。



メニュー画面



IFDの例



ERDの例

図2.3－8　PREDICT　CASE　ワークステーションの画面例

2.3.3　アプリケーション・ジェネレータ

ＮＡＴＵＲＡＬ　ＣＯＮＳＴＲＵＣＴは、ＮＡＴＵＲＡＬをさらに拡張したアプリケーション構築のためのツールであり、ＮＡＴＵＲＡＬプログラムを自動生成する。

標準の機能モデルを提供しており、パラメータを指定することにより、これらを組合せてアプリケーション・プログラムを即座に生成することができる。各種のユーザ出口を提供している他、モデル自身もカマタマイズすることができ、ユーザ特有の要求にあわせたアプリケーションを構築できる。

ＰＲＥＤＩＣＴディクショナリとも完全に、統合化されており、ビュー定義、項目毎のチェック・ルール等が自動適用される。

ＮＡＴＵＲＡＬ　ＣＯＮＳＴＲＵＣＴの利用により、プログラムの共通化・部品化を有効にすすめることができ、生産性の向上に加えて、標準化の推進を実現できる。

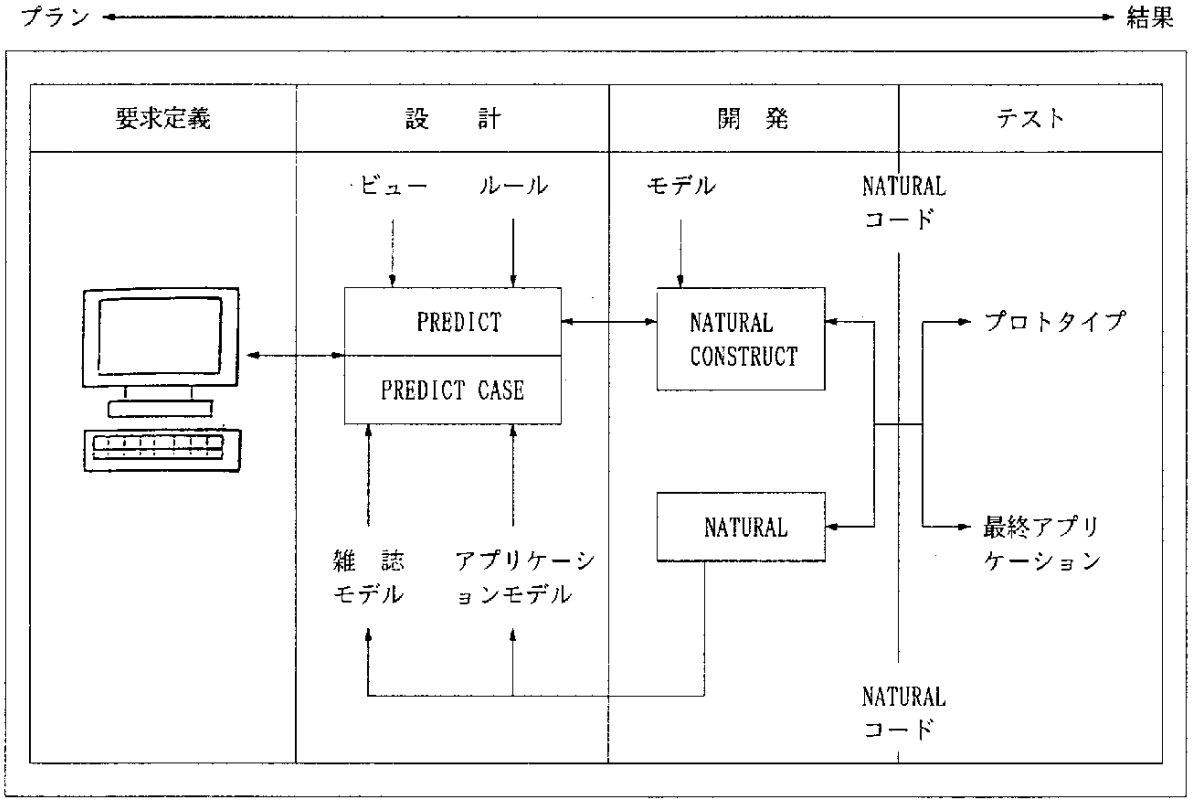


図2.3－9　ＮＡＴＵＲＡＬ　ＣＯＮＳＴＲＵＣＴ

2.3.4　まとめ

ＮＡＴＵＲＡＬを中心としたソフトウェア・エージのＣＡＳＥプロダクト群は、これまで述べたように、ハードウェア／ソフトウェア環境に依存しない統一したアプリケーション開発環境を提供し、アプリケーション開発過程全般を支援する統合型ＣＡＳＥ（Ｉ－ＣＡＳＥ）環境を実現する。

エンドユーザ主導型のアプリケーション開発が、今後ますます重要視されると思われるが、情報システム開発のスタッフとしてエンドユーザが参画するための道具として、これらプロダクトの果す役割は大きい。

2.4　ＳｔＰ

2.4.1　ＳｔＰの概要

ＳｔＰ（Software through Pictures TM）は、ソフトウェアの生産性／信頼性の向上を目的に開発された統合型ＣＡＳＥツールである。その特長は、構造化分析(SA)／構造化設計(SD)の各種方法論のサポートは勿論のこと、リアルタイムＳＡも拡張機能を含めサポートされていることにある。また特に、開発元であるINTERACTIVE DEVELOPMENT ENVIRONMENT INC.(IDE社）の社長であるDr A.WASSERMANらの開発したプロトタイピング用ツール(USER SOFTWARE ENGINEERING METHODOLOGY）も付け加えられている。現在、稼働環境として、分散開発環境下（ネットワーク化されたワークステーション上でのマルチユーザ開発環境）での効果的な利用を可能にしており、大規模なProject Teamによるソフトウェア開発や複雑なリアルタイムシステムの開発に適している。

ＳｔＰでは、オープンアーキテクチャーの思想をとり、従来のツール（オブジェクトコード提供）にはない幅広いカスタマイズ性を備えている。例えば、他社のツールとの結合、ＳｔＰ自身の環境設定変更等が行える。

すでに、日本でも、数種のＣＡＳＥツールが販売されているが、ほとんどがＳｔＰも含めて米国製のものであり、上流工程をサポートする(Upper CASE)ツールが主流を占めている。Upper ＣＡＳＥの場合、当然ＳＡ／ＳＤの方法論をバックグランドに持っているため、ツールサイドの理解度の早さに比べ、方法論の修得度の不足があげられており、日本に於けるＳＡ／ＳＤの方法論の普及／促進の努力が日本に於けるＣＡＳＥツールの普及速度を左右すると考えられる。

しかしながら、現状のソフトウェア開発支援、特に上流工程支援に関しては、これといった自動化されたツールがない現状では、ＣＡＳＥがその期待を受けるのも妥当な選択であろう。

以下に、ＳｔＰの具体的な機能／特長を述べる。

(1)　サポートしている方法論

①構造化分析支援

a. DFE(DATAFLOW DIAGRAM EDITOR)/(DeMARCO/Yourdon, Gane/Sarson)

b. DSE(DATA STRUCTURE EDITOR)/(Jackson)

C. ERE(ENTITY RELATIONSHIP EDITOR)/(Chen)

②リアルタイム構造化分析支援

a. CSE(CONTROL SPECIFICATION EDITOR)/(Hatley/Pirbhai)

b. STE(STATE TRANSITION EDITIR)/(Hatley/Pirbhai)

③構造化設計支援

a. SCE(STRUCTURE CHART EDITOR)/(Constantine/Yourdon)

④プロトタイピング支援

a. TDE(TRANSITION DIAGRAM EDITOR)/(Wasserman)

User Software Engineering

⑤汎用図形エディター（方法論に依存しない、様々な図形生成用）

a. PCT(PicTure)/(Wasserman etc.)

(2) データディクショナリ機能

ＳＡ／ＳＤ方法論より作られたチャートの情報より自動的にデータ辞書を作成し、様々な分析／チェックを行う。

(3) プロジェクト　データベース

データ辞書の構築を含め、情報の一元管理を行う為のＩＤＥ社オリジナルの“Troll/USE”というＲＤＢＭＳが組み込まれており、ＳｔＰの情報管理の中核的な役割を持ってする。

(4) その他マルチユーザ、ネットワーク環境に対応した、各種ロック機能、バージョン管理機能等の強力なユーザ支援環境をサポートしている。

(5) オープンアーキテクチャー(Visible Connections TM)

ツールのインターフェース、ファイル形式、データベースのスキーマ等を公開している。また、ＰＤＬ、Ｐ－ｓｐｅｃ、スキーマ等の情報のジェネレーションを可能にしている。これらにより、ユーザ環境に合せた数々のカスタマイズが可能になっている。

ＳｔＰの“動き”を一部紹介する。図2.4－1は、ＳｔＰのメインメニューである。アイコン化されているのが各技法を支援しているエディタ群で、メインメニューからマウス&クリックのオペレーションで全ての操作を行う事が可能となっている。

図2.4－2は、支援している技法の一つのDataflow Diagramを書く為のエディターＤＦＥである。ＳｔＰでは、統一されたユーザフレンドリなユーザインターフェースを提供している。例えば、この図のポップアップメニューは他の各エディターでも共通のものであり、同じオペレーションで描画を行う事が可能となっている。また、プロセスの詳細化を行う時には、〔下へ〕をクリック後にプロセスを表す丸をクリックするが、データ構造の定義でも同じ様に〔下

へ) をクリックし、データストアやデータフローのオブジェクトをクリックする事によりデータ構造を定義する為のエディター(ERE or DSE)が起動されるようになっている。

各エディターでダイヤグラムを描画した後には、データ辞書生成を行う。ＳｔＰでは、ダイヤグラムを基にデータ辞書を自動生成する。図2.4－3は、データ辞書生成を実行した結果である。ＢＮＦ記法などを直接テキスト編集する必要は無く、ＳｔＰがこの様な形式に自動的に置き換えてくれる。また、生成した辞書から各エディター毎に特有なチェックを行う機能を持っている。これにより、信頼性の高い分析／設計作業が行える。

ＳｔＰの大きな特徴の１つにオープンアーキテクチャー思想がある。ＳｔＰで定義された情報は、データ辞書としてＤ／Ｂ化されて保持されるが、そのＤ／Ｂのスキーマ情報の公開、並びにＤ／Ｂへのアクセス・ライブラリーの提供を行っている。

また、図2.4－1のメインメニューは、ＡＳＣＩＩ　ＴＥＸＴ形式で定義されており、ユーザにより自由に定義／変更することが可能になっている。例えば、ユーザが独自の言語ジェネレータを持っていた場合、ユーザ・カスタマイズにより、ＳｔＰのＤ／Ｂに蓄えられた分析／設計情報からジェネレータに必要な情報を引き出し、ジェネレータに起動をかけると言ったような定義を行い、メインメニューから実行することも可能である。その他にも図2.4－2の様なエディタのポップアップメニューにユーザ定義コマンドを追加することも可能になっている。その他、代表的なエディタは図2.4－4～図2.4－9に示す通りである。

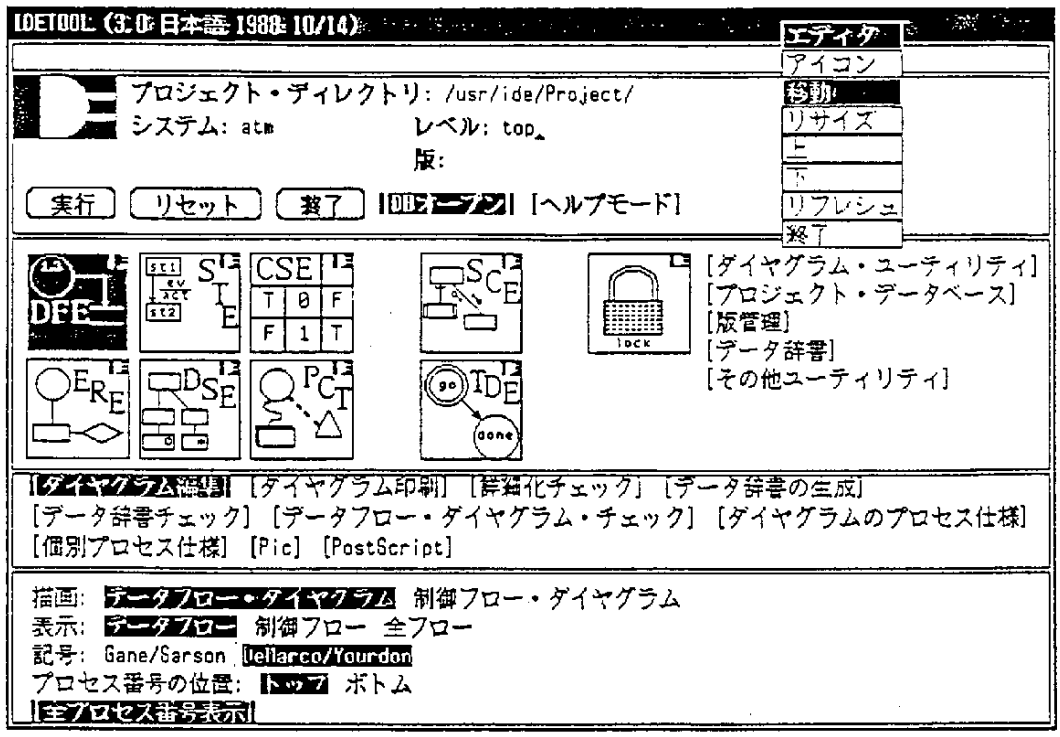
(6)　ＳｔＰ稼働環境

・稼働ワークステーション：日本語版ＳｔＰ：ＳＯＮＹ　ＮＥＷＳ（ＮＥＷＳ・ＯＳ）
：ＳＵＮ／３シリーズ(SUN・OS4.0,JLE 1.0以上)
英語版ＳｔＰ：ＳＵＮ／３シリーズ、ＳＵＮ／４ｼﾘｰｽﾞ(SPARC)
ＤＥＣ／ＶＡＸ　Ｓｔａｔｉｏｎ
その他

・メインメモリー：８ＭＢ以上

・ディスクエリア：25ＭＢ以上（ユーザエリア）

・周辺機器　　：ビットマップディスプレイ
レーザービームプリンタ
等



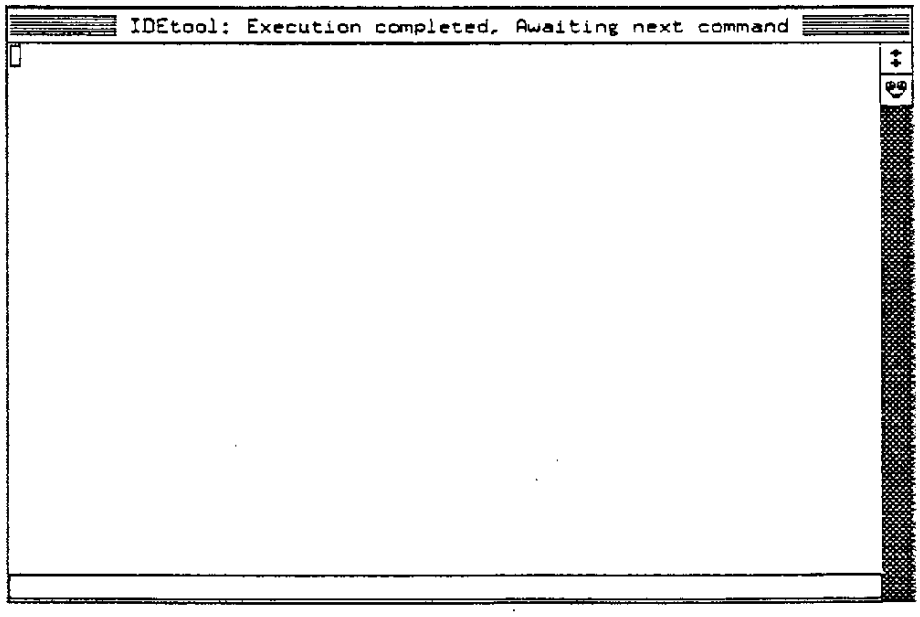


図2.4－1　ＳｔＰの初期画面

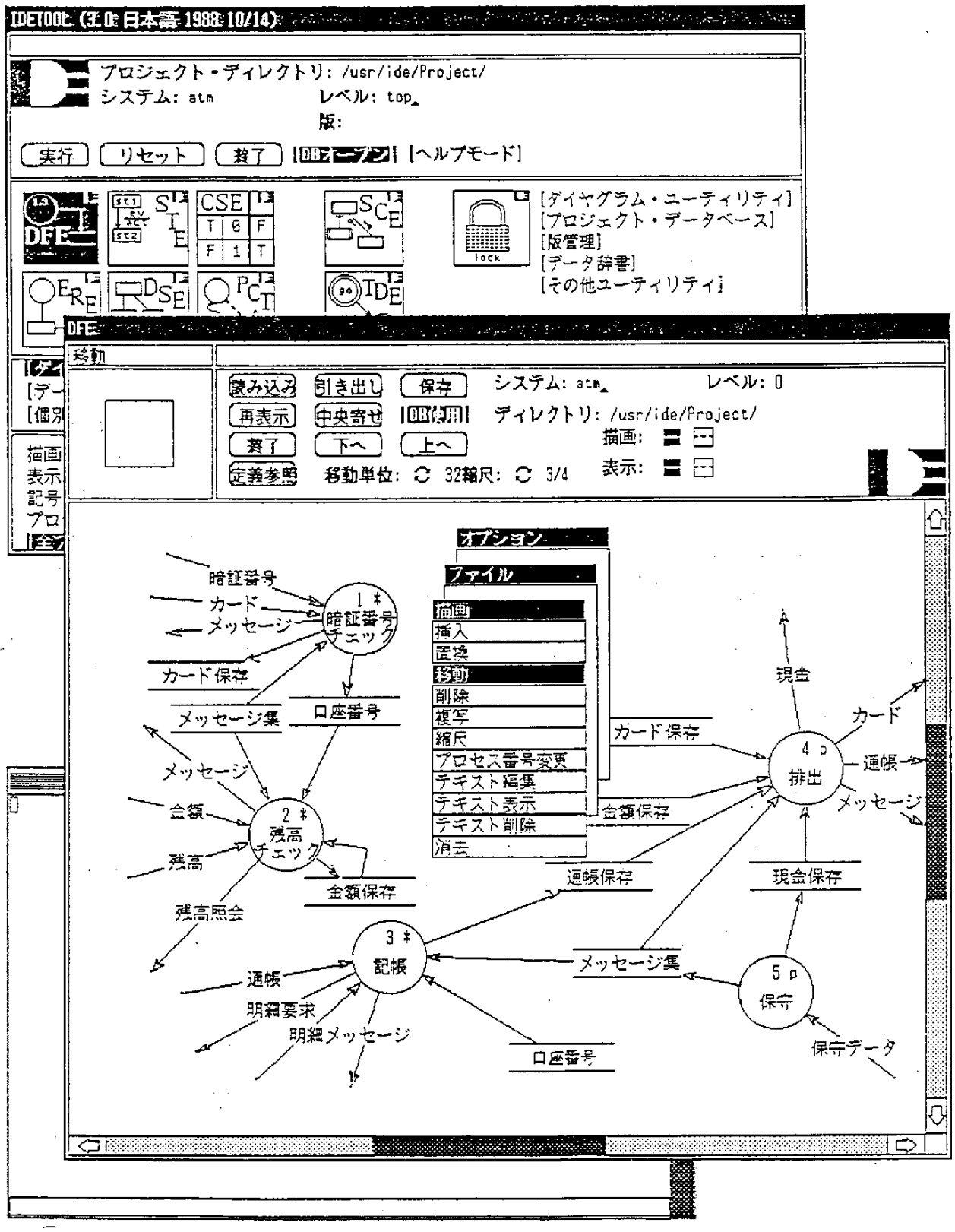


図2.4－2　データフロー図

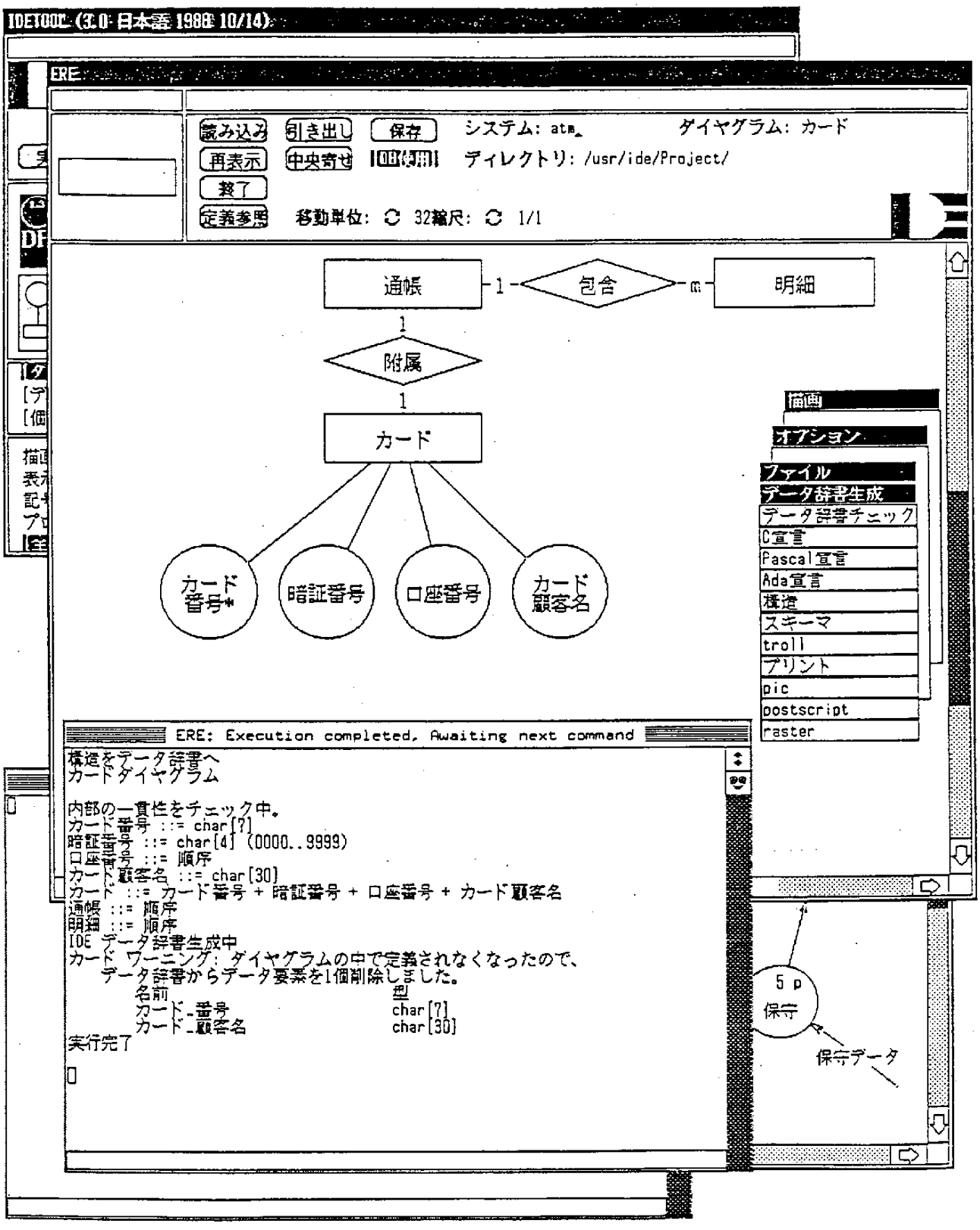

IDETOOL (3.0 日本語 1988 10/14)
ERE
読み込み
引き出し
保存
システム: atm
ダイヤグラム: カード
再表示
中央寄せ
ディレクトリ: /usr/ide/Project/
終了
定義参照
移動単位:
32
縮尺:
1/1
通帳
1
包含
m
明細
1
附属
1
カード
カード番号*
暗証番号
口座番号
カード顧客名
描画
オプション
ファイル
データ辞書生成
データ辞書チェック
C宣言
Pascal宣言
Ada宣言
構造
スキーマ
troll
プリント
pic
postscript
raster
ERE: Execution completed, Awaiting next command
構造をデータ辞書へ
カードダイヤグラム
内部の一貫性をチェック中。
カード番号 ::= char[7]
暗証番号 ::= char[4] (0000..9999)
口座番号 ::= 順序
カード顧客名 ::= char[30]
カード ::= カード番号 + 暗証番号 + 口座番号 + カード顧客名
通帳 ::= 順序
明細 ::= 順序
IDE データ辞書生成中
カード ワーニング: ダイヤグラムの中で定義されなくなったので、
データ辞書からデータ要素を1個削除しました。
名前 型
カード_番号 char[7]
カード_顧客名 char[30]
実行完了
5 p
保守
保守データ

図2.4－3　ER図

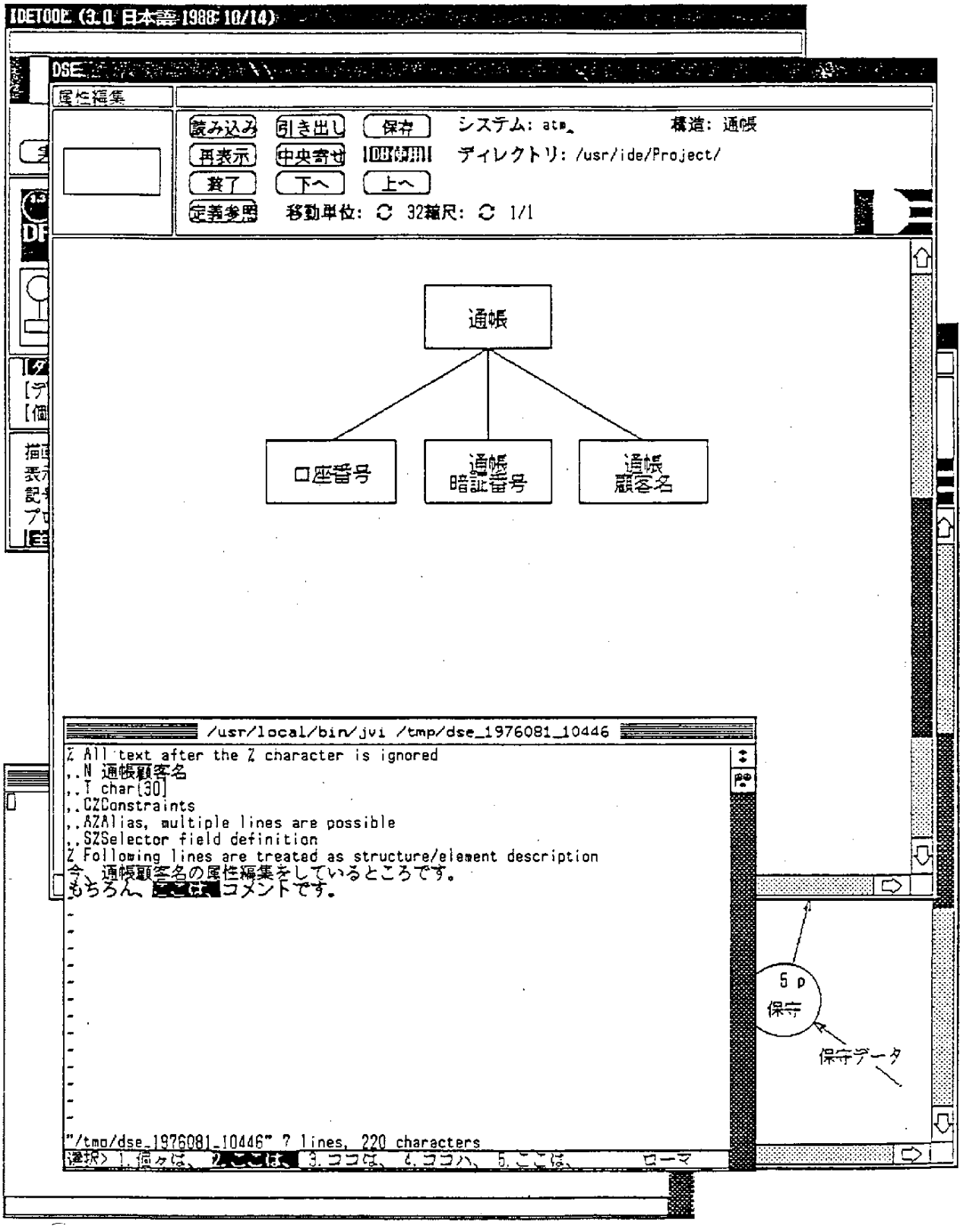

IDETOOL (3.0 日本語 1988 10/14)
DSE
属性編集
読み込み
引き出し
保存
システム: atm
構造: 通帳
再表示
中央寄せ
ディレクトリ: /usr/ide/Project/
終了
下へ
上へ
定義参照
移動単位: 32
縮尺: 1/1
通帳
口座番号
通帳
暗証番号
通帳
顧客名
/usr/local/bin/jvi /tmp/dse_1976081_10446
Z All text after the Z character is ignored
,.N 通帳顧客名
,.T char[30]
,.CZConstraints
,.AZAlias, multiple lines are possible
,.SZSelector field definition
Z Following lines are treated as structure/element description
今、通帳顧客名の属性編集をしているところです。
もちろん、ここは コメントです。
"/tmp/dse_1976081_10446" 7 lines, 220 characters
5 p
保守
保守データ

図2.4－4　データ構造図

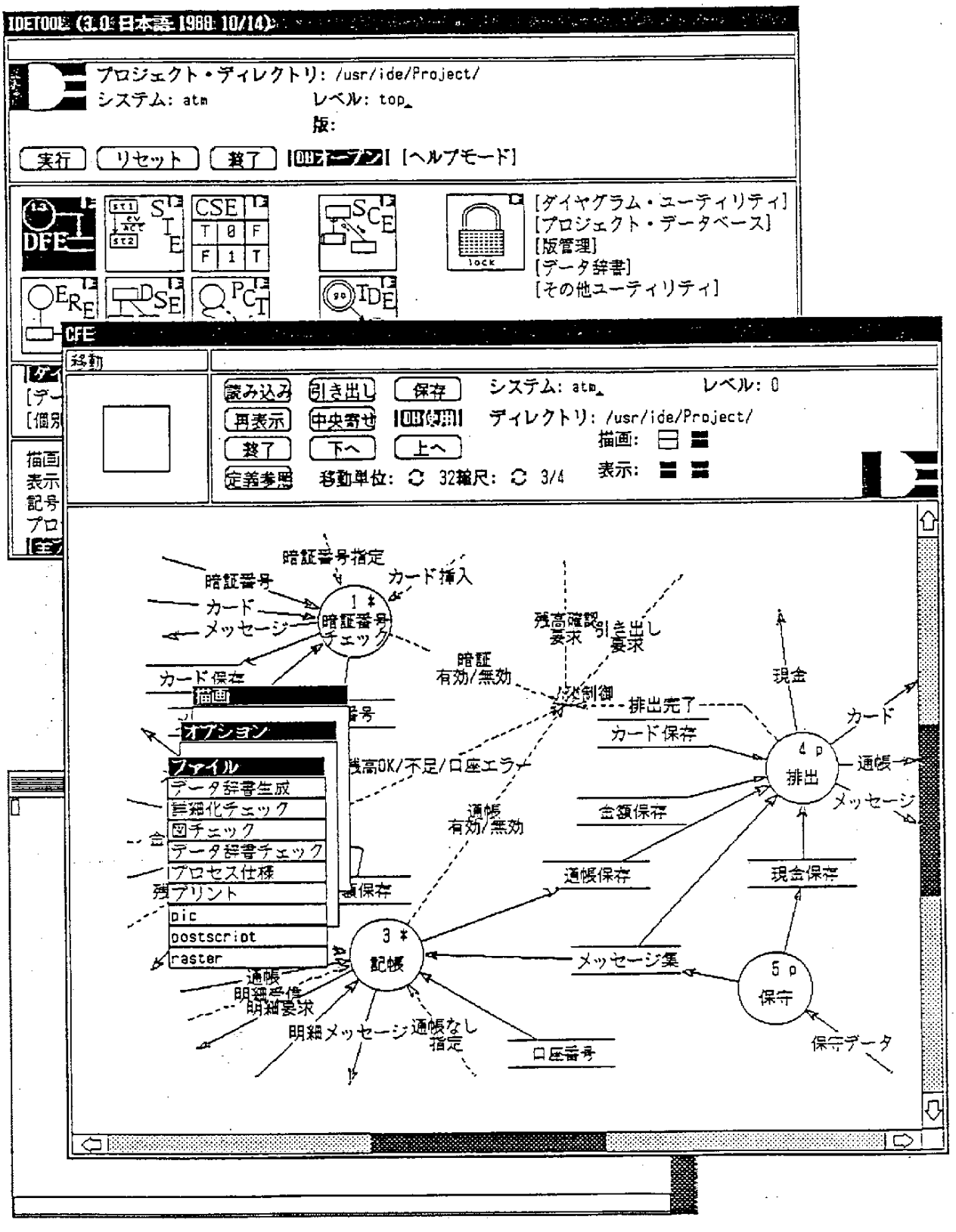

IDETOOL (3.0 日本語 1988 10/14)
プロジェクト・ディレクトリ: /usr/ide/Project/
システム: atm
レベル: top
版:
実行
リセット
終了
[ヘルプモード]
[ダイヤグラム・ユーティリティ]
[プロジェクト・データベース]
[版管理]
[データ辞書]
[その他ユーティリティ]
CFE
移動
読み込み
引き出し
保存
再表示
中央寄せ
終了
下へ
上へ
定義参照
移動単位:
32縮尺:
3/4
システム: atm
レベル: 0
ディレクトリ: /usr/ide/Project/
描画:
表示:
描画
表示
記号
プロ
暗証番号指定
暗証番号
カード挿入
カード
メッセージ
1 *
暗証番号チェック
カード保存
暗証有効/無効
残高確認要求
引き出し要求
制御
排出完了
カード保存
現金
カード
4 p
排出
通帳
メッセージ
金額保存
通帳有効/無効
通帳保存
現金保存
残高OK/不足/口座エラー
描画
オプション
ファイル
データ辞書生成
詳細化チェック
図チェック
データ辞書チェック
プロセス仕様
残プリント
dic
postscript
raster
3 *
記帳
メッセージ集
5 p
保守
保守データ
通帳
明細要求
明細メッセージ
通帳なし指定
口座番号

図2.4－5　制御フロー図

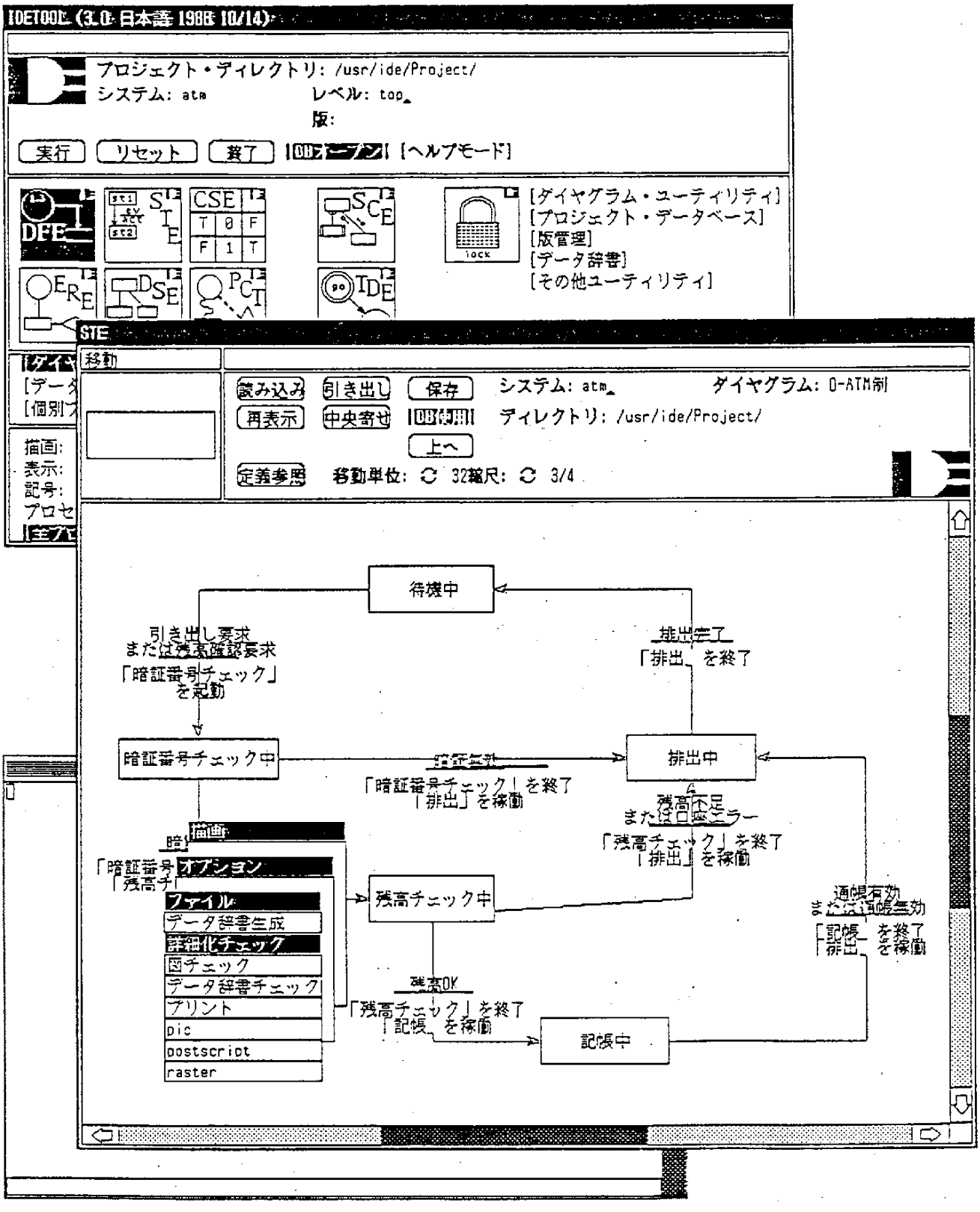

IDETOOL (3.0 日本語 1988 10/14)
プロジェクト・ディレクトリ: /usr/ide/Project/
システム: atm
レベル: top
版:
実行
リセット
終了
[ヘルプモード]
[ダイヤグラム・ユーティリティ]
[プロジェクト・データベース]
[版管理]
[データ辞書]
[その他ユーティリティ]
STE
移動
読み込み
引き出し
保存
システム: atm
ダイヤグラム: 0-ATM制
再表示
中央寄せ
ディレクトリ: /usr/ide/Project/
上へ
定義参照
移動単位:
32
縮尺:
3/4
待機中
引き出し要求
または残高確認要求
「暗証番号チェック」
を起動
暗証番号チェック中
暗証無効
「暗証番号チェック」を終了
「排出」を稼働
排出中
排出完了
「排出」を終了
残高不足
または口座エラー
「残高チェック」を終了
「排出」を稼働
残高チェック中
残高OK
「残高チェック」を終了
「記帳」を稼働
記帳中
通帳有効
または通帳無効
「記帳」を終了
「排出」を稼働
描画
オプション
ファイル
データ辞書生成
詳細化チェック
図チェック
データ辞書チェック
プリント
dic
postscript
raster

図2.4－6　状態遷移図

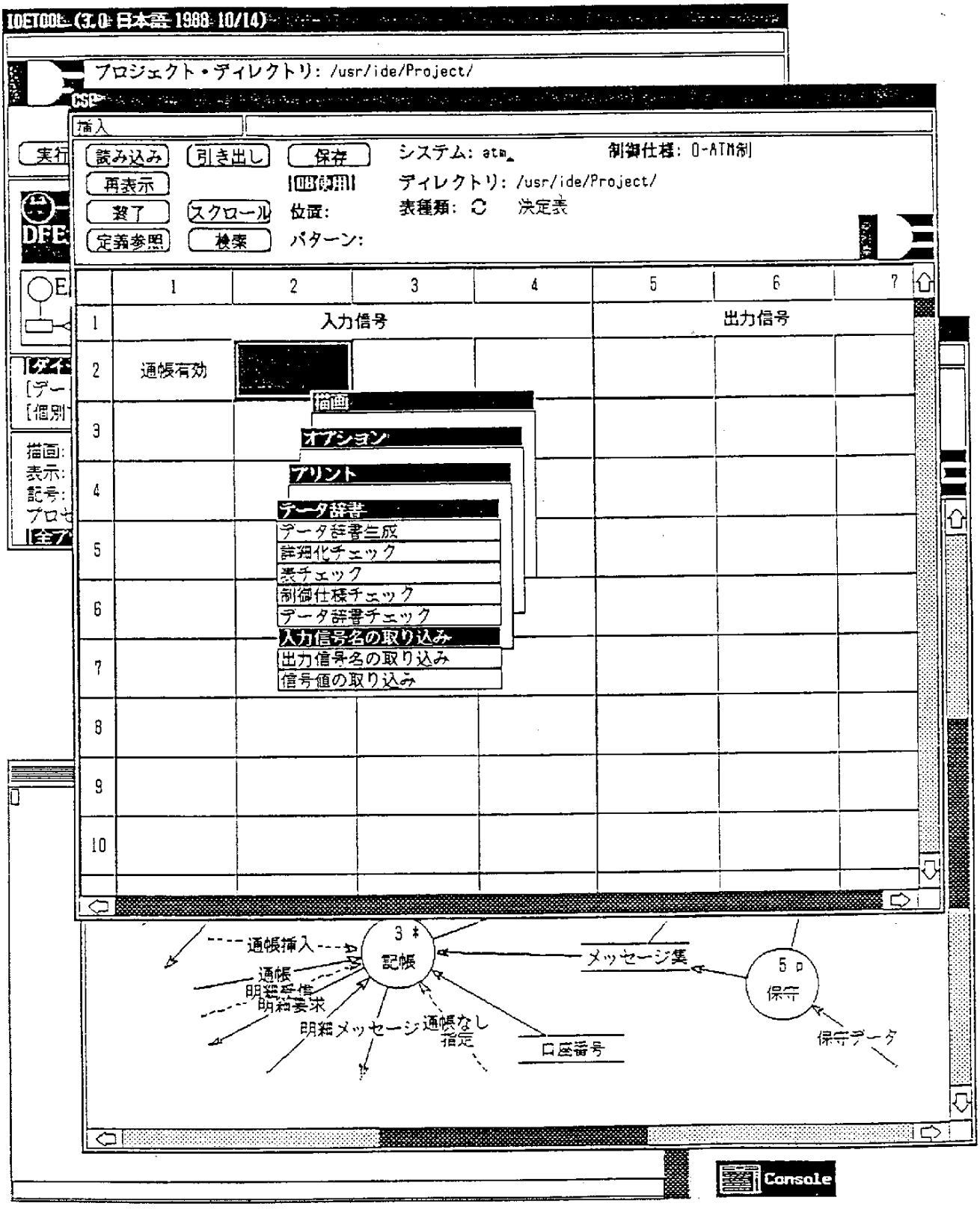

プロジェクト・ディレクトリ: /usr/ide/Project/
挿入
実行
読み込み
引き出し
保存
システム: atm
制御仕様: 0-ATM制
再表示
ディレクトリ: /usr/ide/Project/
終了
スクロール
位置:
表種類:
決定表
定義参照
検索
パターン:
入力信号
出力信号
通帳有効
描画
オプション
プリント
データ辞書
データ辞書生成
詳細化チェック
表チェック
制御仕様チェック
データ辞書チェック
入力信号名の取り込み
出力信号名の取り込み
信号値の取り込み
通帳挿入
通帳
明細受信
明細要求
明細メッセージ
通帳なし指定
3 *
記帳
メッセージ集
5 p
保守
保守データ
口座番号
Console

図2.4－7　決定表

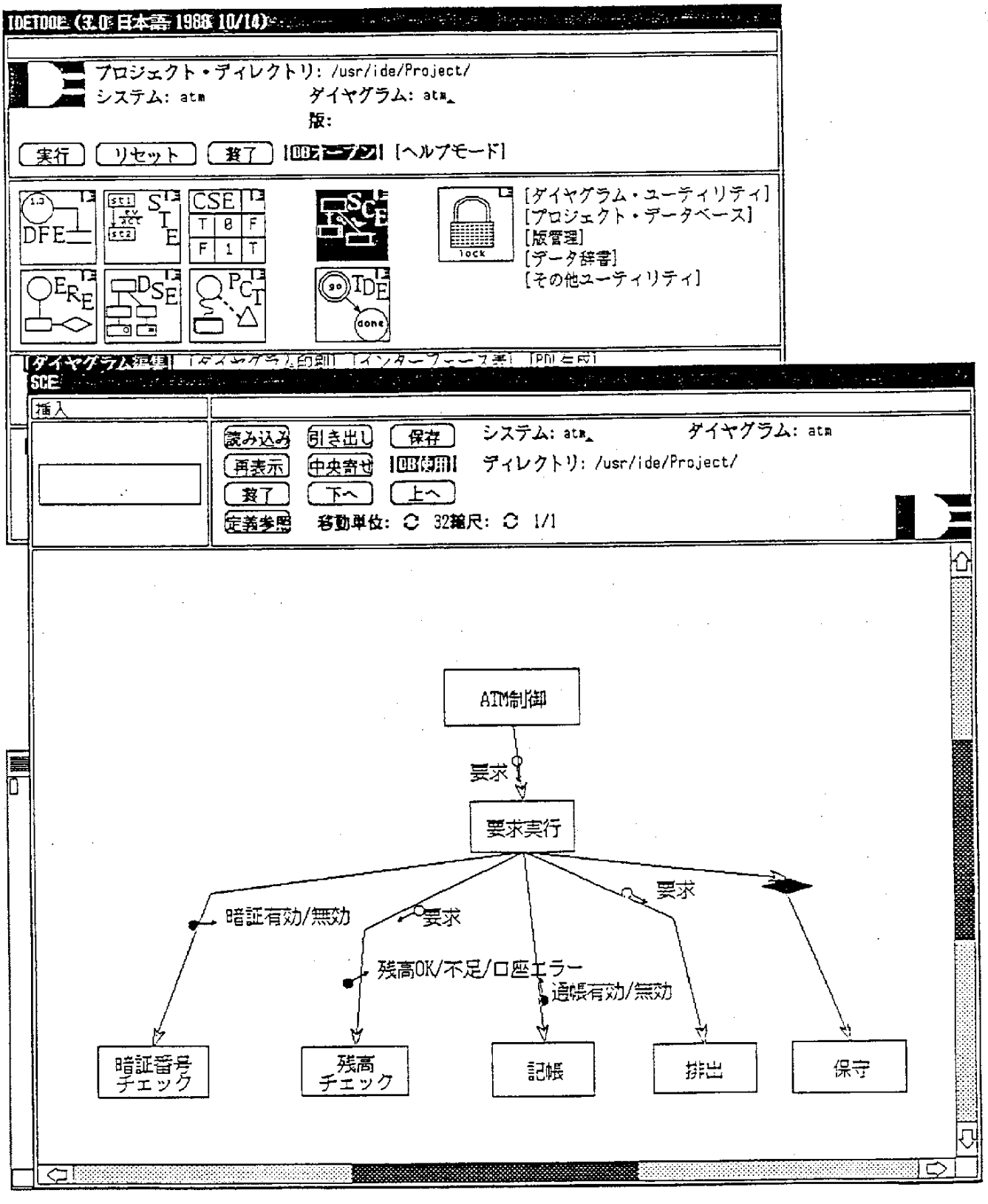


図2.4－8　構造化チャート

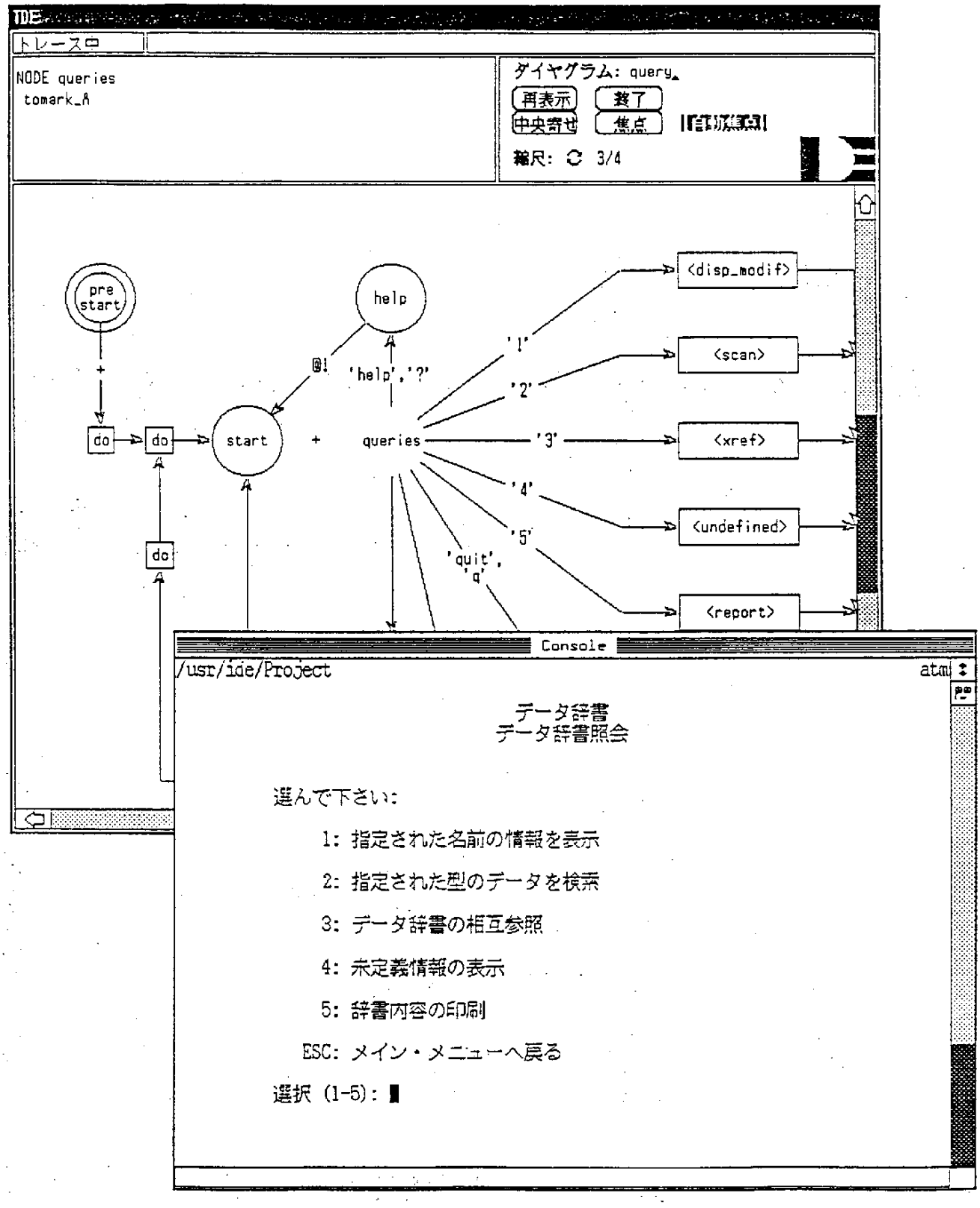


図2.4－9　プロトタイピング用遷移図

2.4.2　ＳｔＰの効果

ＣＡＳＥツールに関しての効果については、ＳｔＰも含め特に日本に於いてはまだ、ＣＡＳＥツールが上陸してきて間もないこともあり、具体的な効果に関するデータがあまりとられていないのが現状である。この問題に関しては昨年秋に米国で開催されたCASExpoFall'89に参加して資料や意見等を見聞きしたが、米国に於てもＣＡＳＥの現状調査している団体等でも、ＣＡＳＥツールは急激に普及してはいるが、その効果の具体的なデータが取られておらず取る必要性が協調されていた。一般的には効果が数倍から数十倍という声が上がっているが、それは、従来の手法との比較なのか、どの工程でのものなのか、実際使っている設計者のレベルは高いのか低いのか等様々な観点でのデータの取りようによって大きく差が出てしまうのでなかなか困難を要する。

現在、当社では社内でのＳｔＰ利用のデータ等を取り始めようと努力している。また、ＳｔＰユーザの数も増えてきており、近い将来にユーザグループの発足を計画中である。できればその機会に使用効果の事例等を収集し、今後のＣＡＳＥの普及に寄与できるよう考えている。具体的な適用に関しては、当社がある顧客から受託した作業で実際に使用した例での結果について報告する。

作業内容：制御システム

開発規模：約１万Ｓｔｅｐ（Ｃ：言語）

ＳｔＰ適用工程：システム要求定義／設計工程

特に、エンドユーザ仕様の確定／レビュー作業

開発環境：開発場所／当社、ハードウェア環境／ＳＯＮＹ－ＮＥＷＳ

適用内容：システム要求定義段階（エンドユーザとの仕様の取り決め作業）で、具体的なＳｔＰで作成したシステム外観図及びデータフロー図を使用

具体的な効果としては次のものがあった。

① ＳｔＰで作成した外観図等を利用した各種チェック作業で効果（数値的なデータはとれていない）があった。

② エンドユーザ側と当社設計側の使用の詰め（確認／レビュー）が、文書、会話等、従来の表現方式での作業とＳｔＰを使用した場合とでは、ＳｔＰ利用の方が両者の確認の際の洩れや具体的な仕様の追加の是非が短期間で解消された。

③　新しいアイデアがさらに色々と発生し逆に作業内容としては密度の濃い作業内容となった。

④　仕様の確定作業に於て、ワークステーションで対話形式で実際にその場でエンドユーザに要求システムのイメージを見せることによって、より具体的な内容の確認が可能になった。

なお、留意点としては次のものがある。

①　ユーザへのＳｔＰ仕様の是非の了承をとる必要がある。

②　設計者がＳＡ／ＳＤ方法論及び、ＳｔＰを熟知していること。

③　②の逆で熟知していない場合は、従来の方法より時間が多くかかる事が想定される。

④　開発環境として、ワークステーションが導入されていること。

⑤　ＳｔＰを開発工程のどの段階で使用するかが明確になっていること。

ＳｔＰの効果についてまとめると次のようになる。

①　紙とエンピツそして“消しゴム”がいらない！

マウス、アイコン等簡単な操作でドキュメント作成が可能になり、且つ修正（メンテナンス）作業が機械化される。

②　従来の言葉、文章、各種チャートによる“曖昧”からの脱皮（設計者のレベルや個人によって、設計書等のドキュメントが異なりエンドユーザや、共同設計者、ＳＥ、プログラマに誤解を招きやすい、もしくは、説明に多大な時間／コストが費やされる場合が多い。）

これは、結果的には標準化の不備であり、小規模なシステムであれば許される場合もあるが、大規模システムや複雑なシステムでは、許されない問題である。ＳｔＰを使う効果の一つは、標準化の手助けになることがいえる。

③　ＳｔＰのＣＡＳＥツールとしての最大の効果は、要求定義／分析／設計の各工程での作業に於けるＳｔＰの論理的チェック機能が挙げられる。これにより、各工程の作業の不具合、ミス、エラー等を自動的にチェックし、後工程でのエラーの発生頻度を極端に減少でき、前工程への戻りを防ぐ効果がある。

④　ＳｔＰを使って作られた各種設計図やデータは、自動的にデータディクショナリに保持される。このことは、類似システム開発等での再利用に活かされ重複開発を防ぐ事が可能になる。

⑤　複数の人がチームとして開発するときに、ＵＮＩＸ、ワークステーションのネットワークをフルに活用でき、情報の伝達、データの共有化、打ち合わせ等コミュニケーションの手段としても、非常に効果的である。

2.4.3　ＳｔＰ適用上の注意

ＳｔＰに限らず、ＣＡＳＥツール全般（Ｕｐｐｅｒ－ＣＡＳＥ）に対する適用上の注意としては以下のものが想定される。

(1)　適用範囲の決定

①　ＳｔＰを利用しようとしているターゲットシステムは？

ターケットシステムの開発規模は？大（？）中（？）小（？）

どんなシステム？

事務処理（？）リアルタイム処理（？）ｅｔｃ.

社内システム（？）、エンドユーザシステム（？）ｅｔｃ.

②　開発工程のどの範囲で利用するのか？

要求分析（？）、要求定義（？）、基本設計（？）、詳細設計（？）等。

現状の問題点を調査し、その問題点の解決を目的に利用範囲を決定することが大事である。

ターゲットシステムの性格／内容／開発期間等も適用範囲の決定に考慮する必要がある。

(2)　方法論の決定

上記①、②の内容に応じ構造化分析（ＳＡ）／構造化設計（ＳＤ）／リアルタイム構造化分析の各種方法論のどれをメインに使用するか等を検討する。

ターゲットシステムの内容によっては、全ての方法論を使う場合と特定の方法論を使うだけで済む場合がある。

(3)　環境のカスタマイズ方針決定

開発組織（部、課、プロジェクト等）の開発環境（ハードウェア環境、ソフトウェア環境、開発手法、標準化等）に於て、従来の環境とＣＡＳＥを導入するに当たっての新しい環境の移行／改善等を整理する。これは、開発組織の単位（レベル）でカスタマイズの大小は異なると思われる。カスタマイズ方針にも、管理面と技術面が必要となる。

(4)　ツールのカスタマイズ方針決定

ＳｔＰのオープンアーキテクチャーの利点を活かす所であり、上記の検討結果によりＳｔＰ自身の環境設定をターゲットシステム及び部門／プロジェクトの開発環境にあわせたカスタマイズ（変更）を必要に応じて行う。

例として、ＳｔＰの初期画面（メインメニュー）にプロジェクト特有のメニューを追加し他のツールをＳｔＰのメインメニューから起動かけられるようにする等。

(5) 方法論の講習

ＣＡＳＥ（ＳｔＰ）ツールを使う上で最も重要な点である。基本的な、構造化分析（ＳＡ）／構造化設計（ＳＤ）方法論の修得そして、リアルタイム構造化手法の修得が必須である。修得方法としては、各種文献（方法論に関する）による勉強、当社セミナー及び関連セミナー等の受講がある。

(6) 環境のカスタマイズ

(3)の方針の具体化を行う。

(7) ツールのカスタマイズ

(4)の方針の具体化を行う。

(8) ツールの講習

ＳｔＰのオペレーションの修得を行う。ツール自身の操作に関しては通常半日～数日（利用者レベルによる）で修得可能である。

まとめとして、上記導入に当たっての手段に関しては、導入時期及びＳｔＰ利用／適用度合いに応じて導入手段の順番／内容をより現実の状況に促したかたちで優先順位を付けて実行するための検討が必要である。

2.4.4 ＳｔＰの今後の計画

ＩＤＥ社の考えているＣＡＳＥ環境を図2.4－10に示す。現状におけるＳｔＰのアーキテクチャ（リリース4.2）を図2.4－11に、また将来計画（リリース５）を図2.4－12に示す。なお、統合化されたＣＡＳＥ環境の構築及び、ＳｔＰユーザの利用環境の向上を目指して次のことも計画している。

(1) オープンアーキテクチャーの発展

各種戦略パートナーとのＪｏｉｎｔ（米国において）

① ドキュメント支援

・Ｆｒａｍｅ Ｍａｋｅｒ

・Ｉｎｔｅｒｌｅａｆ

・AT&T ｄｉｔｒｏｆｆ

② データベース

・Ｓｙｂａｓｅ

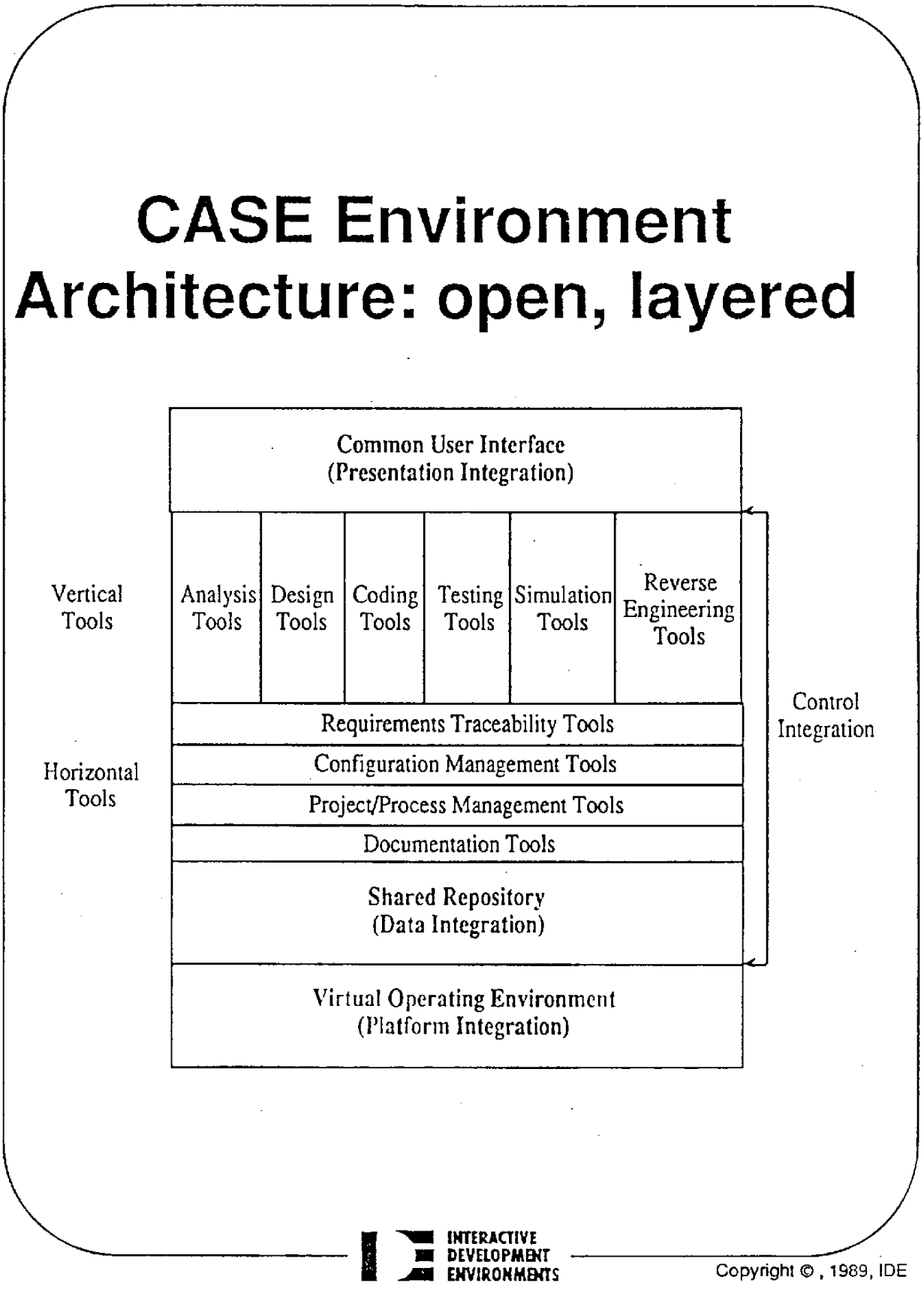


図2.4－10　ＣＡＳＥの基本構造

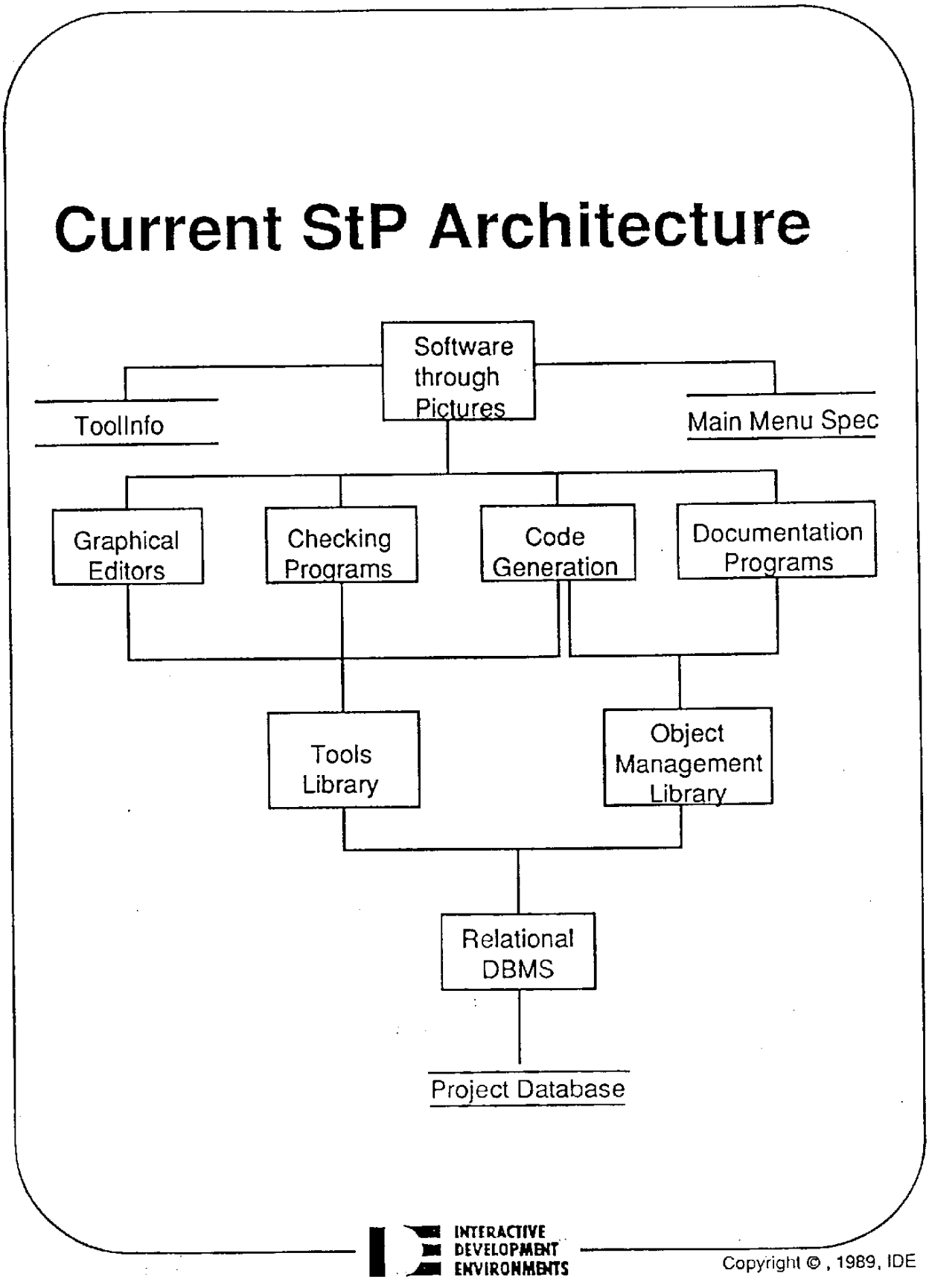


図2.4－11　現状におけるＳｔＰのアーキテクチャ

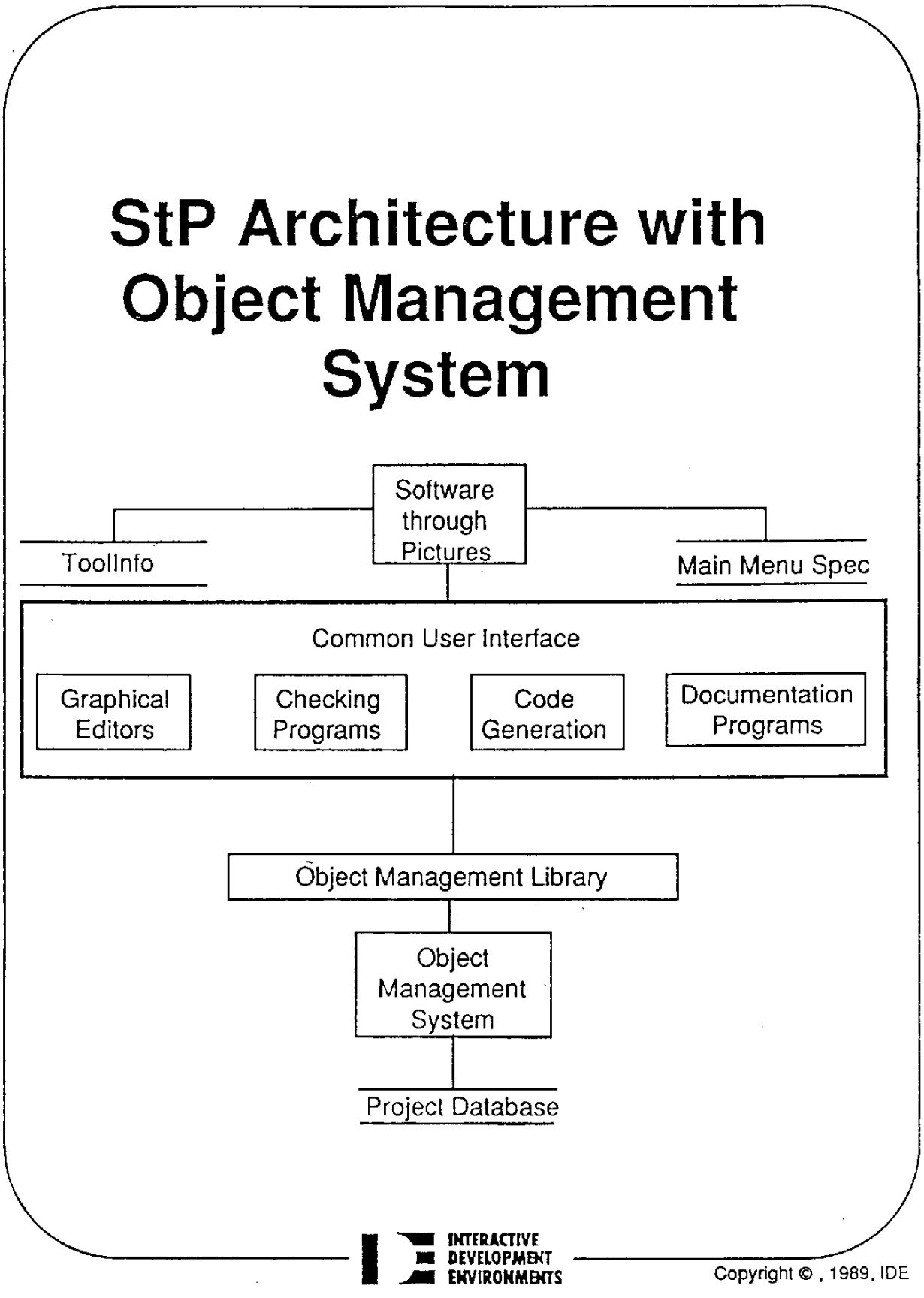


図2.4－12　将来のＳｔＰアーキテクチャ

・ｅｔｃ.

③ 各種ジェネレータの接続

Ｉ－ＣＡＳＥへの具体化

(2) ヘテロジニアスネットワーク上での利用環境の実現

ＴＭ：“Software through Pictures”，“Visible Connections”は、米国 INTERACTIVE DEVELOPMENT ENVIRONMENTS Inc.（ＩＤＥ社）の商標である。

2.4.5 その他

(1) ＣＡＳＥ Ｔｏｏｌの定義？

ＣＡＳＥ(Computer Aided Software Engineering）の名の通りソフトウェアの開発支援ツールは全て、ＣＡＳＥと呼んでもおかしくないが、狭義には、（専門家達は）ソフトウェア開発のライフサイクルの中でも、上流工程（要求分析／定義、設計）を支援するための方法論を持った支援環境（ツール）と考えていることが大勢を占めている。上流工程は、従来のソフトウェア開発ライフサイクルの中で最もコンピュータによる支援が遅れていた工程である。

近年になって、ハードウェアの急激な進歩（高性能ワークステーション等の登場）及びＵＮＩＸに代表されるオープンアーキテクチャー思想のオペレーティングシステムや各種ネットワーク機能の充実により、いままで”重たい”と言われていて、高価なミニコンピュータなどでしか稼働できなかったソフトウェアツールが安価な高性能ワークステーション上で稼働出来るようになったことも大きく影響している。現在のＣＡＳＥ（特にＵｐｐｅｒＣＡＳＥ）ツールも、その機能の豊富さからツール自身のボリュームも大きく“重たい”ソフトウェアに属する。しかし、上記で述べた通り高性能ワークステーションであれば、問題なく稼働出来る環境が整ってきた。

(2) 方法論とは？

上流工程支援（Ｕｐｐｅｒ）では

構造化分析技法(Structured Analysis)

構造化設計(Structured Design)

が中心の方法論である。

これらの方法論の提唱者は、最も有名な人として

Tom DeMarco (Structured Analysis/Design)

Ed Yourdon （Structured Analysis/Design）

Larry Constantine（Structured Design etc.）

Micheal Jackson （Jackson Methods（Module Struct.））

Chris Gane(Structured System Analysis)

Trish Sarson(　　〃　　)

等が挙げられる。他にも、多くの研究者等が関わっていると思われるが、ここでは省く。

また、近年特にリアルタイム処理系の為に、構造化分析を基にリアルタイムシステム処理用の構造化分析（仕様書作成手法）を

Derek Hatley

Imtiza Pirbhai

の２人が中心となって具体化／実践したものが有名である。

現時点で、ＣＡＳＥツールの主流を占めている方法論は上記に挙げた３種類のものである。当然、ツール化される場合は、これら方法論の関連付けやデータディクショナリーの持ち方等が異なってくる。また、各種ユーザーインターフェースの具体化もそれぞれのツールによって異なってくる。

最近では“Information Engineering”“Re Engineering”“Prottyping”等の手法？が提唱されている。これもＣＡＳＥの原点であるソフトウェア開発支援環境を目的とした一連の流れであると思われる。そして、ソフトウェア開発のトータルライフサイクルをカバーする為のＩ－ＣＡＳＥ(Integrated CASE：統合化されたＣＡＳＥ環境）の構築が望まれ始めている。しかし、上流工程と下流工程を統合化するには、まだまだ技術的な課題が多分に残されている。

(3)　ＣＡＳＥの現状

日本での現状を考えてみると、ＣＡＳＥという言葉が世の中で騒がれ始めたのは、１９８８年前後と思われる（一般的に）。

しかし、ＣＡＳＥという名前が何を意味しているか、どんな目的に適用できるのか等に関してはその本質を摑んでいる人は残念ながら少ないのが現状であると推測される。

米国での急速なＣＡＳＥの普及が、日本へ多大な影響を与えていることは事実であるが、米国の現状もその伝わってきている評判を調べてみると様々な問題（積極的な）を抱えているようである。それらに対しては、既に調査団体が具体的にデータをとり始め、問題解決の為に様々な活動を行っている。

米国でのこれらの状況は、今後日本での普及の手本となることが多々あると思われる。また、日本に於ても普及のための様々な活動を地道にする必要がある。（特に、日本は米国との習慣の違いやソフトに対する価値観の差、カスタマイズ性等日本独自の考え方も必要であると思われる。）

(4) 具体的な問題点及び課題

① 日本に於けるＳＡ／ＳＤ方法論の普及の不足

② 日本語の遅れ（この１年で実用化されてはいるが）

③ ワークステーション（ＵＮＩＸ）の本格的利用の遅れ

（飛躍的に普及してはいるが？）

④ 日本特有の伝統的な開発手法（保守的な）からの脱皮の問題

⑤ 方法論を中心とした各種情報の不足

⑥ ＣＡＳＥの使用実績結果の不足

⑦ 各種事例の不足

⑧ 教育関連の場の不足

⑨ ＣＡＳＥに対する過度な期待（夢物語）を持ち過ぎる。

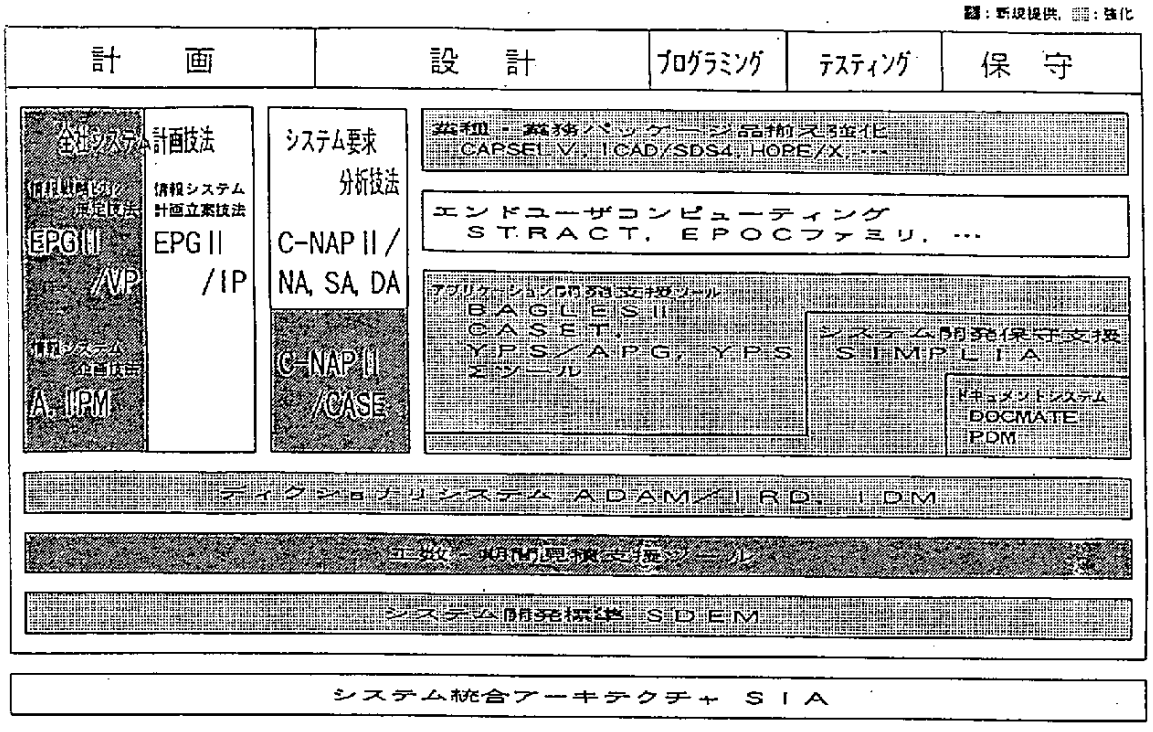
等

2．5　SDAS

2．5．1　SDASの概要

SDAS（Systems Development Architecture and Support facilities）は富士通の総合開発システムであり，多数の技法，ツール，パッケージの総称である。ここでは，この内いわゆるCASE，特に計画・設計工程を支援するツールを中心に説明する。

SDASは図2.5-1 に示すような体系を持っている。



EPG II:Executive Planning Guide II , VP:Vision Planning, IP:Information system Planning
A. IPM:ALLCHAT® Information Planning Method™,

C－NAP II:Customer-Needs and systems Analysis Procedures II, NA:Needs Analysis SA:Systems Analysis , DA:Data Analysis
C－NAP II／CASE:Customer-Needs and systems Analysis Procedures II /Computer Aided Software Engineering

CAPSELV:Customer's APplication SErvice Library V
ICAD／SDS4:Integrated Computer Aided Design and manufacturing system/Standard Design System 4
HOPE／X:HOspital administration system PakagE/eXtended,

STRACT:STRategic interACTive information system

BAGLES II:Business Application Generator's Library for Extensive Support II
CASET:Computer Aided Software Engineering Tools
YPS／APG:YPS based Application Program Generation system,
YPS:YACII Programming System , YACII:Yet Another Control chart II

SIMPLIA:SIMple development & maintenance support Program Libraries for Application system
DOCMATE:DOCument MATE
PDM:Program Dictionary Manager,

ADAM／IRD:Application Development And Management facilities/Information Resource Dictionary
IDM:Item Dictionary Manager

SDEM:System Development Engineering Methodology

SIA:Systems Integration Architecture

図2.5-1 SDASの体系（90年1月）

このＳＤＡＳは三つの狙いを持っている。

⑴計画設計品質の向上による生産性の向上

企業の経営目標を実現する整合性のとれた情報システムを構築するためには要求定義を的確に行うことが重要である。このため全社システム計画技法ＥＰＧⅡ，情報システム企画技法Ａ．ＩＰＭ，システム要求分析技法Ｃ－ＮＡＰⅡ，Ｃ－ＮＡＰⅡ支援ツールＣ－ＮＡＰⅡ／ＣＡＳＥを提供し問題の解決を図る。

⑵中間工程省略による生産性の向上

新規開発や保守の効率化を実現するためにソフトウェア開発の中間工程を省略する三つの方法を提唱している。

・業種，業務パッケージの活用

新規開発を最小限におさえ，早期稼働を図る。

・エンドユーザコンピューティング

エンドユーザ自身が容易に利用できるツールを提供することにより，情報を利用する現場でタイムリーに検索・加工することを図る。

・上流工程からの機械化，自動化

アプリケーション開発支援ツール群による機械化，自動化や部品再利用によ，開発と保守の効率化を図る。

⑶生産性向上のための基盤の整備

・プロジェクト管理の重視

プロジェクト管理に必要なシステム開発の標準的な作業項目と作業手順や，見積手法やその支援ツールを提供し，品質や管理の容易性を向上させる。

・分散開発の推進

高機能，高性能ワークステーションを最大限に活用し，安定したレスポンスと高い操作性で開発効率の向上を実現する。

・ディクショナリシステムの活用

上流工程から下流工程までの開発資源情報の連携を図り，システムライフサイクルを通した開発保守の効率化を実現する。

このような狙いを図にあらわしたものが図2.5-2 である。

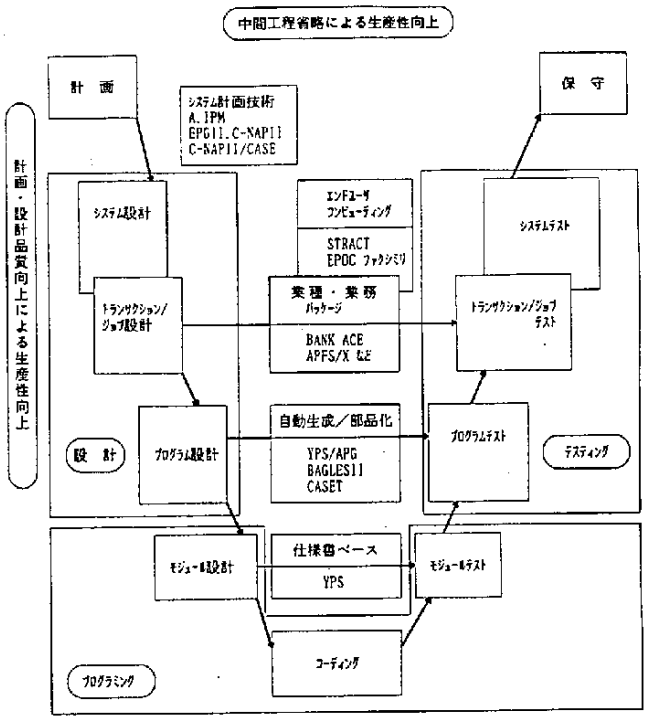


図2.5-2 ＳＤＡＳの狙い

２．５．２　Ｃ－ＮＡＰⅡ／ＣＡＳＥ

Ｃ－ＮＡＰⅡ支援ツール　Ｃ－ＮＡＰⅡ／ＣＡＳＥ（Customer-Needs and systems Analysis Procedures Ⅱ/Computer Aided Software Engineering）は，データ主導型システム分析技法Ｃ－ＮＡＰⅡの分析作業を支援するツールである。

分析過程を支援するツールは，どんな分析技法に基づいているかが本質的である。以下に技法の基本的な考え方と，支援ツールの基本機能を説明する。

(1)　Ｃ－ＮＡＰⅡの基本的な考え方

1)問題意識

基幹業務のシステム化が一巡し，これらを統合した新しいシステムの開発が多くの組織で行われている。このような大規模システムでは，業務の全体像，システムの全体像を利用者・開発者が共有することが困難であり，体系的な業務分析・システム分析の方法論が求められている。

構造化分析やデータベース指向の方法論をベースとしたいわゆる上流CASEツールが話題に

なっている。しかし実際にこれらをシステム開発過程で使う場合，利用者にとって難解であれば協力が得られない，専門の分析者が必ずしもいない，といった難しさがある。即ち；

①利用者の参画の問題

分析に参加する利用者と開発者では本来分析に対する関心事が異なる。また専門の分析者が用いるような分析法を利用者に強要することは難しい。

②分析の量と深さの問題

限られた期間内で，一定品質の分析を行うためにはいたずらに詳細な分析はオーバヘッドを増すだけである。また純粋にトップダウン又はボトムアップで行うことは現実的でなく，両者をうまく組み合わせて用いる方法が必要である。

2)特徴

以上の問題意識から，Ｃ－ＮＡＰⅡでは次のような点を基本姿勢としている。

①利用者の視点と開発者の視点の分離

例えば，データフロー図やＥＲ図などは利用者にとっては抽象的すぎて理解し難い。両者で用いる表現方法やその目的は区別したほうが受け入れられやすい。

②分析情報の一貫性

しかし，両者で用いる分析概念が連続的につながっていくことが必要である。

③分析の多面性

複雑で多様な業務をモデル化するのに，一つの視点からだけとらえたモデルだけでは不十分である。機能階層，順序や動き，データの構造といった多面的な分析を，互いに関連づけて進められることが望ましい。

3)構成

以上の考えから，Ｃ－ＮＡＰⅡは図2.5-3 に示すような体系になっている。

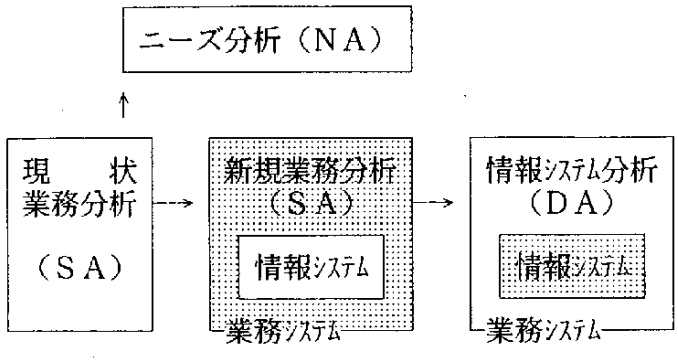


図2.5-3 Ｃ－ＮＡＰⅡの構成

Ｃ－ＮＡＰⅡ／ＮＡ(Needs Analysis)とＳＡ（Systems Analysis）は，それぞれ利用者の視点からのニーズや問題点の分析，及び業務分析を支援する。ＮＡで抽出した業務のクリティ

カルな問題点に対して，その解としての新しい業務像をＳＡで表現する。ＳＡの分析対象は情報システムの外側にある実世界そのものである。

一方，ＤＡ(Data Analysis) は，ＳＡの結果を開発者の視点から読み直すという立場で実施する。ここではＳＡの業務記述を情報システム側の概念にマッピングすることによって，実世界に忠実な情報システムの論理的なモデルを導く。

このような流れは図2.5-4 に示すようにＳＡとＤＡの分析ドキュメント上で連続的につながっていく仕掛けになっている。

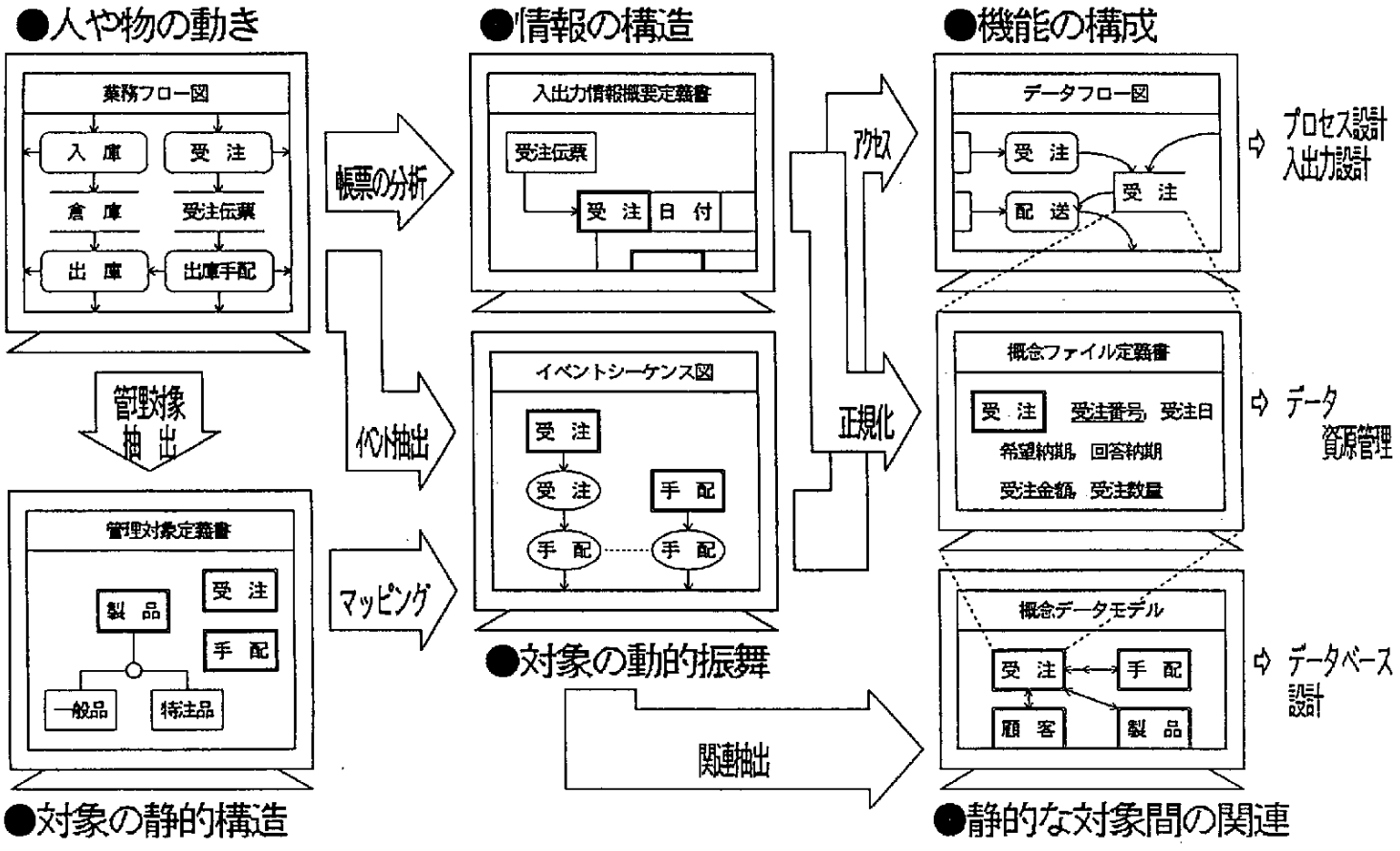


図2.5-4 Ｃ－ＮＡＰⅡの分析ドキュメントと手順

(2) C－NAPⅡ／CASEの基本機能

以上のような技法使った分析作業を支援するツールがC－NAPⅡ／CASEである。C－NAPⅡ／CASEは以下の四つの基本機能を持っている。

1)ドキュメント作成支援機能

C－NAPⅡでは１７種類の表記法が準備されている。このそれぞれに対して専用のエディタが容易されている。図2.5-5 に画面イメージを示す。

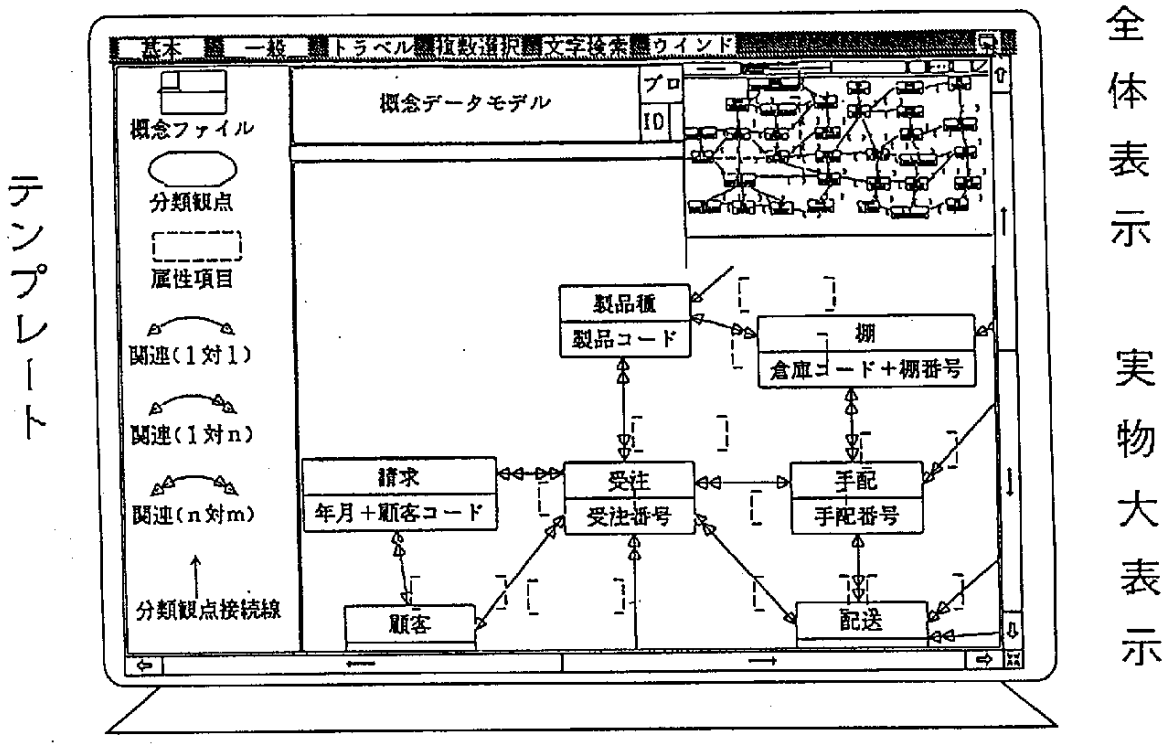


図2.5-5 C－NAPⅡ／CASEの画面イメージ

このエディタは図形を扱えるワープロ以上の機能として，編集操作（移動，複写，削除）の際に，図形の接続関係を適切に維持しつづけたり，複数の図形を同時扱う機能をもっている。記述された内容は適切に解釈されてドキュメントファイルに格納され以下の機能で利用される。

2)複数ドキュメント間のナビゲーション機能

C－NAPⅡの分析で作成されるドキュメントは数百～数千枚となり，相互のドキュメント間に階層関係や参照関係がある。ナビゲーションはこのようなドキュメント間の関係を維持し，簡単な操作で関連ドキュメントを探し出して画面上に表示する機能である。例えば，あるウインド上で編集している業務フロー図のデータフローの詳細仕様を見たい場合，そのデータフローを選択して操作ボタンを押すとツールは関連する入出力情報定義ド

キュメントウインドを開設する（図2.5-6 参照）。

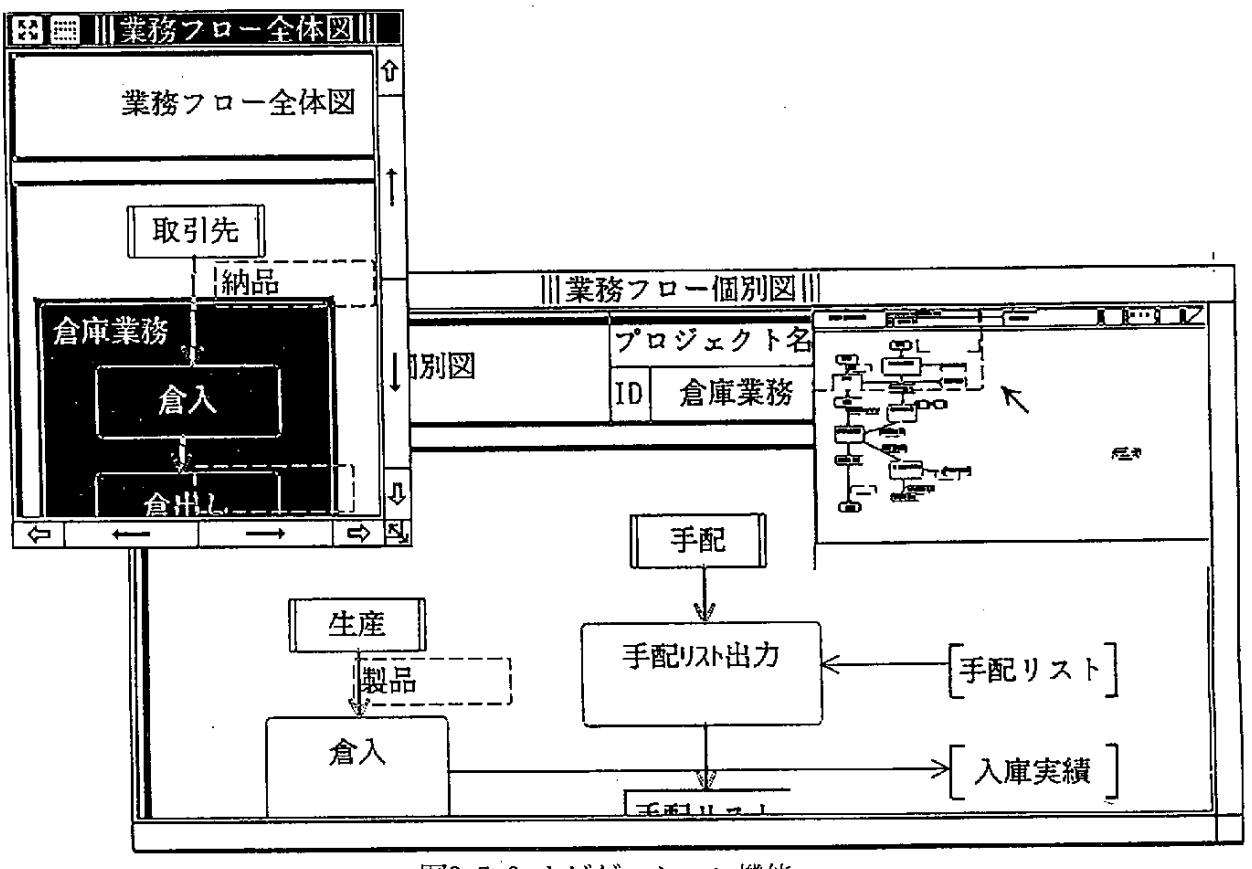


図2.5-6 ナビゲーション機能

3)記述内容チェック機能

C－NAPⅡはドキュメントを書く際に幾つかの約束事を守らなければならない。この約束事をチェックするのがこの機能である。約束事には以下に示すように，一枚のドキュメントの中での形式に関するものと，複数のドキュメント間での一貫性に関するものがある（図2.5-7 参照）。

・一枚のドキュメントの中での記述形式上の約束事

－名前を付けていない図形があってはいけない

－同じ名前を使ってはいけない

－矢印でつながれていない図形があってはいけない

－その他

・複数のドキュメント間での一貫性に関する約束事

－下位のドキュメントが記述されていない

－上位のドキュメント内に下位のドキュメントが引用されていない

－その他

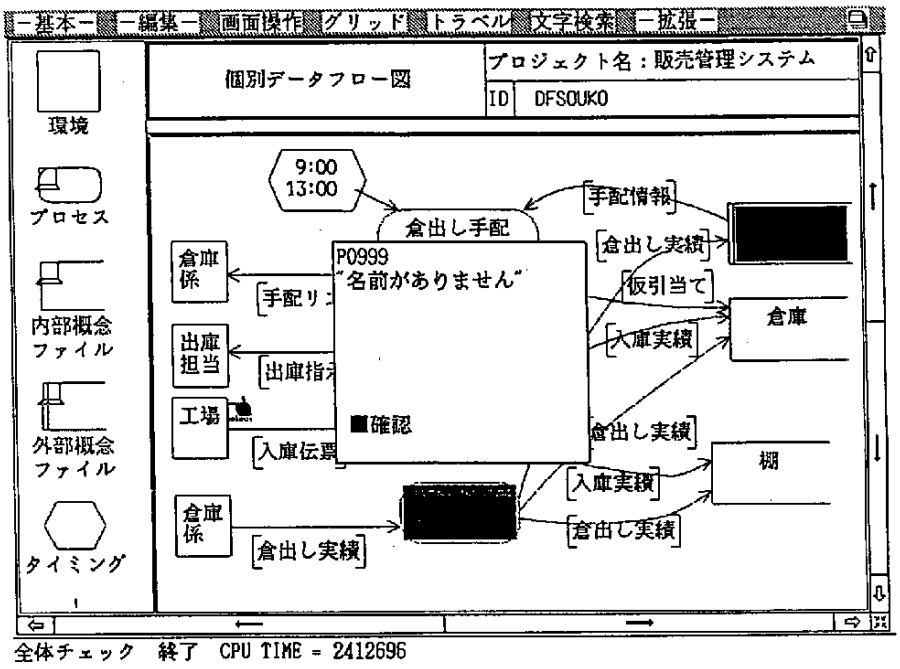


図2.5-7 チェック機能

4)候補抽出機能

C－NAPⅡの一連のドキュメントには相互に強い関連を持つものがある。例えば，業務フロー上のデータは管理対象図を書く際の管理対象の候補となる可能性が高い。このように作業を進める際に既に書いたドキュメントを分析して候補を抽出する機能がこの候補抽出機能である（図2.5-8 参照）。

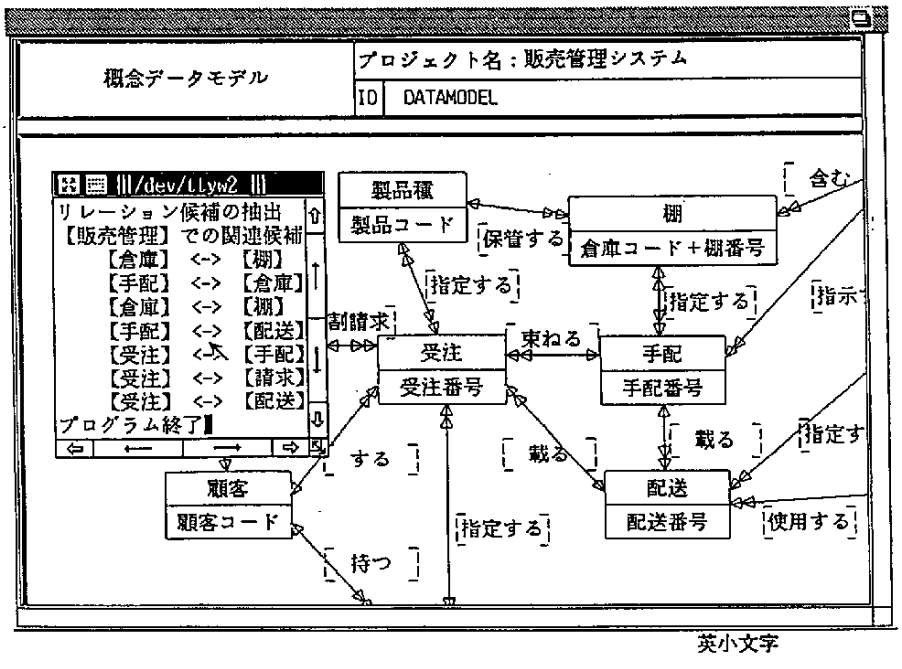


図2.5-8 候補抽出機能

2．5．3　ＹＰＳ／ＡＰＧ

アプリケーション開発・保守一貫支援システム　ＹＰＳ／ＡＰＧ（YPS based Application Program Generation system)は，部品化とその再利用を中心コンセプトとして設計工程からの機械化自動化を目指したシステムである。以下に基本的な考え方，基本機能，利用の流れ，を説明する。

(1)　基本的な考え方

1)ｱﾌﾟﾘｹｰｼｮﾝの構成要素

ＹＰＳ／ＡＰＧでは開発対象のアプリケーションシステムの構成を図2.5-9 のように捉えている。

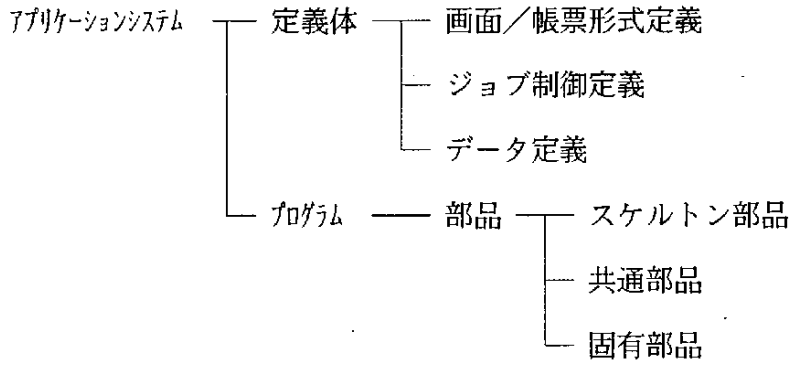


図2.5-9 ＹＰＳ／ＡＰＧにおけるｱﾌﾟﾘｹｰｼｮﾝｼｽﾃﾑ構成要素

すなわち，アプリケーションシステムは，定義体とプログラムにより構成され；

・定義体は画面や帳票の形式定義，データの定義，ジョブを制御するジョブ制御定義から構成される。

・プログラムはプログラムの断片で構成される。この断片のことを総称して部品と呼ぶ。図2.5-10のように，部品を“プログラム内での役割り”という観点から整理して，スケルトン部品，共通部品，固有部品の３種に分類している。

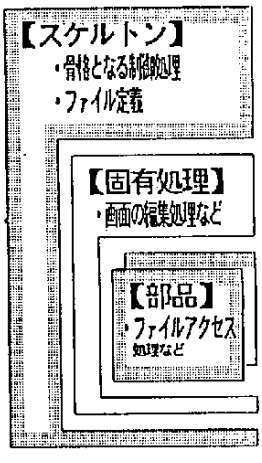


図2.5-10部品のﾌﾟﾛｸﾞﾗﾑ 内での役割

2) 構成要素の設計表記法

上記の構成要素を設計するための仕様表記をそれぞれ次のように準備している。

－定義体の仕様表記

＋画面や帳票の形式定義：スペーシングチャートと同様のイメージ図

＋データの定義：表

＋ジョブを制御するジョブ定義：ジョブフロー図

－部品の仕様表記

＋スケルトン部品：画面遷移図

＋共通部品：決定表，編集表，ＹＡＣⅡ

＋固有部品：　〃　，　〃　，　〃

以下の図2.5-11～16はそれぞれの表記法である。

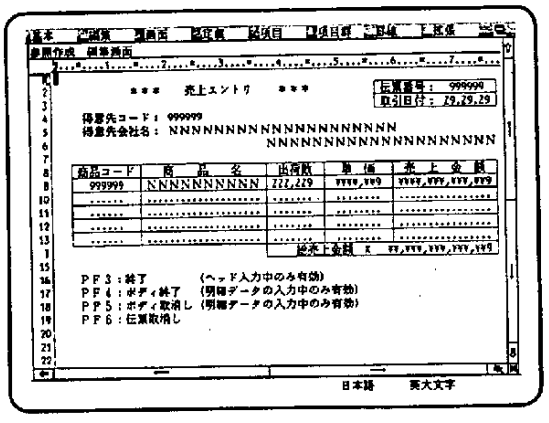


図2.5-11　画面設計書

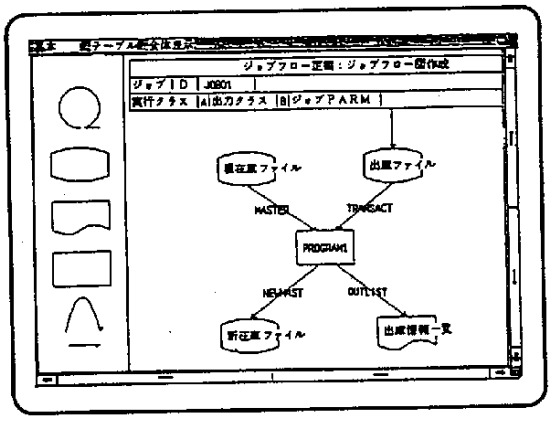


図2.5-12　ジョブフロー図

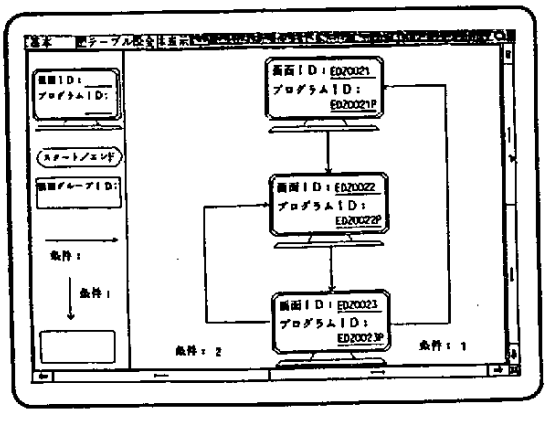


図2.5-13　画面遷移図

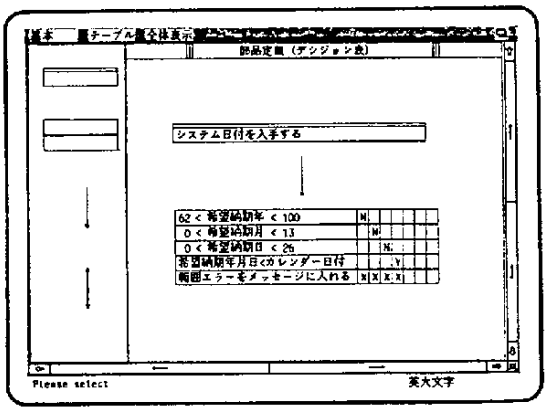


図2.5-14　決定表

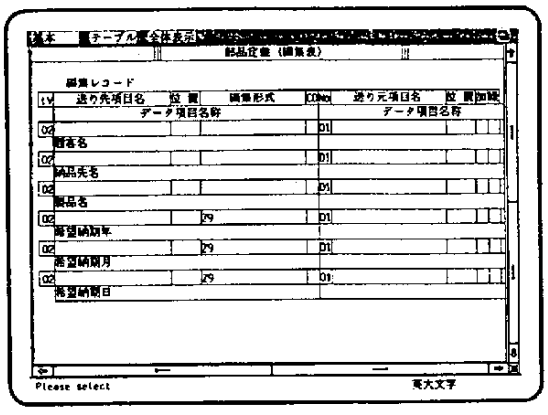


図2.5-15　編集表

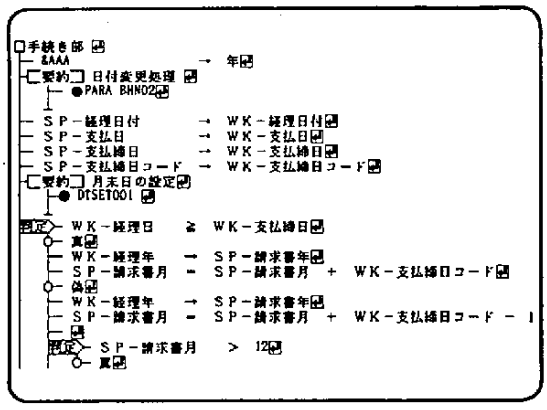


図2.5-16　ＹＡＣⅡ

3) 設計書から構成要素の自動生成

上記の表記法を用いて作成する設計書は計算機の画面上で直接作成し，そこから定義体やソースプログラムの断片である部品（これ自体ＹＰＳ表現）を生成する。

4) 部品からプログラムの合成

部品設計書（画面遷移，決定表，編集表，ＹＡＣⅡ）から生成された部品を合成して一つの実行可能なプログラムを合成する。

(2) 基本機能とその狙い

1) 自動生成による生産生・品質の向上

ＹＰＳ／ＡＰＧではシステム設計で決まったプログラムの入出力情報や画面遷移などの設計情報からスケルトンを自動生成するため，スケルトン部分は，プログラム設計が省略できる。スケルトンはプログラムとしては半完成のため，プログラム設計工程では，スケルトンに組み込む詳細な処理を作成し，スケルトンと合成してプログラムを完成させる。

また，ツールから出力する設計ドキュメントと合成したプログラムとは常に一致しており，プログラムソースと設計ドキュメントの二重管理や矛盾発生といった保守上の問題はなくなる。

2) 部品の再利用による効率化

プログラム間で共通な処理を部品としてあらかじめ準備しておき，システム設計の段階

から再利用を考慮した開発をすることにより，プログラム設計工程で新規に設計が必要となるのはそのプログラムに固有な処理のみとなる。

ＹＰＳ／ＡＰＧでは再利用を支援するために，次の３つのことを行う。

・プログラムの合成，部品の検索，部品とプログラムの相互参照情報出力などを行うための部品管理機能の提供

・適用範囲の広い部品をつくるため，部品の中で使用する名称（データ項目名，部品の中から更に呼ばれる部品名など）を自由に変更する機能の提供

・特定の業務や業種によらず共通的に使用できる処理を部品として提供

3) ワークステーションからの設計書イメージでの入力

基本的考え方で述べたように，ｱﾌﾟﾘｹｰｼｮﾝｼｽﾃﾑの構成要素に対してそれぞれ適切な表記法が準備されており，この表記法を効率良く扱える専用のエディタが提供される。

4) ディクショナリによる情報の管理と再利用

ディクショナリにより一元管理されたデータ項目情報を設計情報の入力時に引用できるため，重複入力や矛盾定義を防止できる。

5) オンライン画面遷移のシミュレート機能によるユーザ要件の早期確認

設計段階で画面の動作をシミュレートさせ，利用者の目で画面レイアウトと画面遷移の仕様を確認できるため，後工程で設計に立ち戻る変更を防止できる。

6) 運用条件に応じた適用パターンの選択が可能

1)～5)の機能を運用条件に応じて選択して部分的に利用することが可能である。

(3) 開発の流れとツールの利用

ｱﾌﾟﾘｹｰｼｮﾝの開発作業とツール利用の関係を図2.5-17に示す。

1)データ項目の標準化と定義(図2.5-17［１］)

同じ意味のデータには同じ名前をつけるなどの標準化を行うことで，保守性の向上と仕様の統一・再利用を容易にする。

標準化したデータ項目の名称は前章の C-NAPⅡ/CASE の分析結果としてADAM/IRDに登録され管理されている。ここではデータ項目の桁数・タイプ・意味などデータ項目で一意に決まる情報の他に，データ項目のチェック仕様や表示属性などの設計情報を定義する時のデフォルト値やプロトタイピングで使用するサンプル値がある。

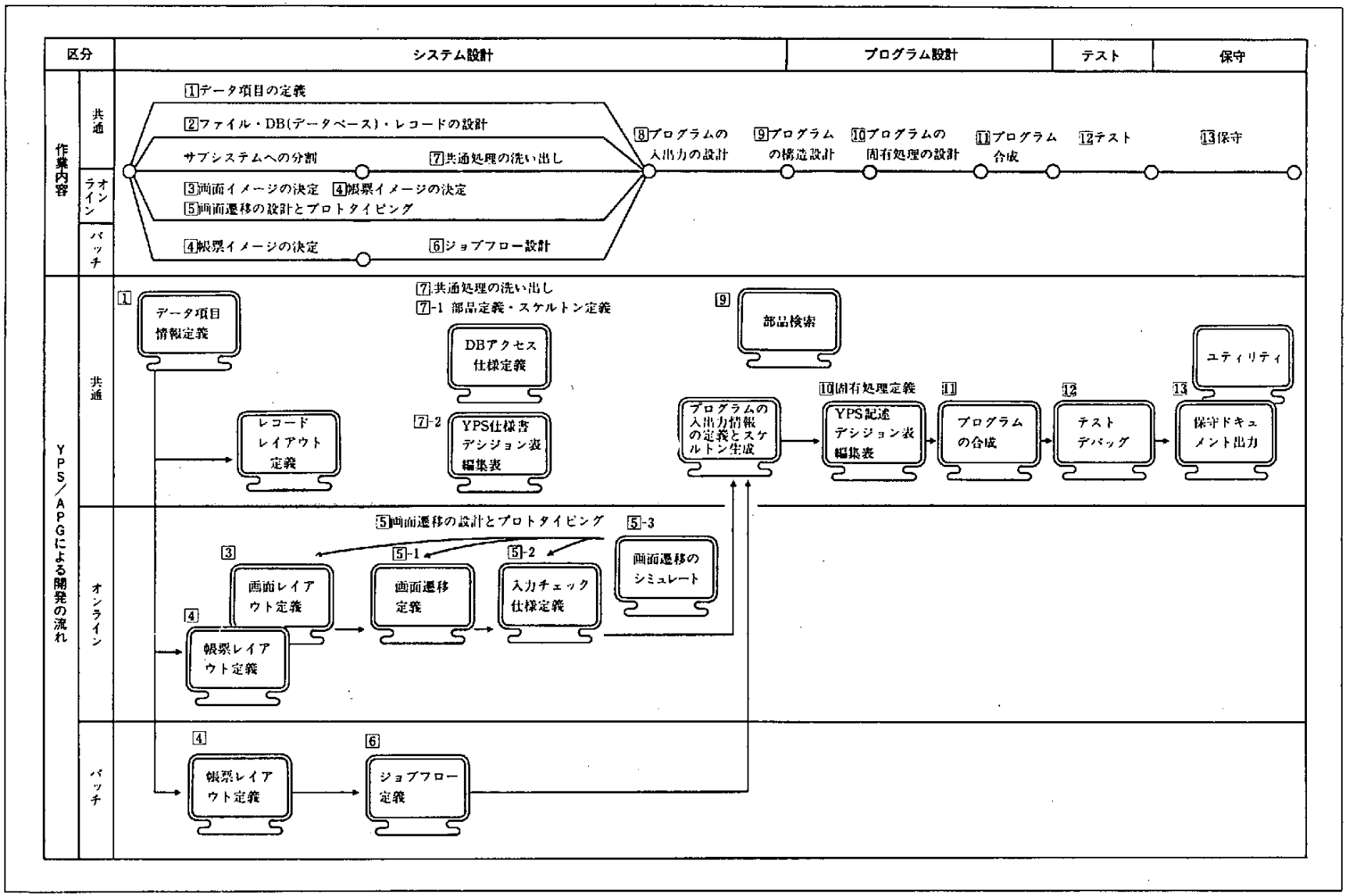


図2.5-17　アプリケーションの開発作業とツールの利用

2) 入出力レイアウトの定義(図2.5-17［2］～［4］)

レコードレイアウト定義・画面レイアウト定義・帳票レイアウト定義を行う。

→FILE, FORM, FORM/OVD

3) プログラム間で共通な処理の洗い出し(図2.5-17［7］)

プログラム間で共通な処理を洗い出し，部品として登録しておく。

共通処理の洗い出しは，いろいろな業務で繰り返し行う処理に着目し，処理の内容や処理の実行順序を標準化して，部品として登録する。

部品の中のデータ項目名やファイル名などの文字列は部品定義の文字列変数として可変にできる。

4) スケルトンの生成

一連の業務処理をプログラムに分割し，分割した各プログラム単位にスケルトンを生成する。

①オンラインプログラム（図2.5-17［5］［8］）

・画面の流れの定義

画面の流れ（画面遷移）を定義し，画面とプログラムを対応づける。画面遷移が決定されるとプログラム分割も決まる。画面遷移の定義が終了したら，各画面で入力されたデータをどのようにチェックするかを定義する（入力チェック仕様定義）。

・画面操作のプロトタイピング

画面遷移定義と画面フォーマット定義を元に，運用端末を使用して実際の画面を表示してレイアウトを確認し，さらにファンクションキーやデータ入力を行って画面の切り換えやチェック仕様を確認する。

・スケルトンの自動生成

画面に対応した各プログラムに対して入出力にかんする仕様（ファイル，サブスキーマなどの情報）を決めるとプログラム環境部，データ定義部の情報が決まる。画面遷移などの情報と併せてスケルトンを生成する。

②バッチプログラム（図2.5-17［6］［8］）

・ジョブフローの定義

ジョブをジョブステップに分割する。このとき各ジョブステップに対して骨格となる制御の種類を選択しながら，ジョブフローを定義する。

・スケルトンの自動生成

図2.5-18のような基本形から必要なものを選択し，これに必要な入出力仕様（ファイル，サブスキーマなどの情報）を組み込んで目的とするプログラムのスケルトンを生成する。

基本形は図2.5-18の標準形の他にユーザ独自に作成して登録することもできる。

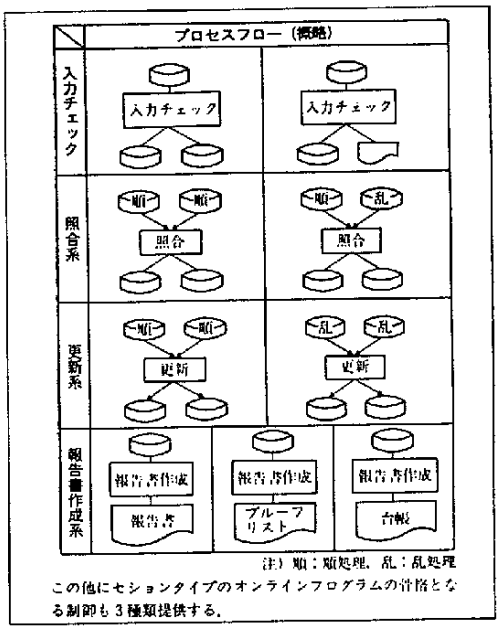


図2.5-18　ＹＰＳ／ＡＰＧで提供する骨格となる制御の種類

5) プログラム設計

スケルトンに組み込む詳細処理の仕様を設計する。

①プログラムの構造設計（図2.5-17［９］）

・部品を検索して再利用可能な部品を見つけ出す。

・スケルトン，部品，固有部品の呼び出し関係を決定する。

②プログラムの固有処理の設計（図2.5-17［10］）

そのプログラムの独自の処理を設計する。ＹＡＣⅡ，決定表，編集表のいずれかを用いて定義する（図2.5-14～16）。

6) プログラムの合成（図2.5-17［11］）

プログラムを合成するために，部品の中で可変になっている呼び出し先や文字列に実際の値を指定する。指定された呼び出し関係や値に従ってプログラムが合成される。

値の与え方には次の２とおりの方法がある。

①記述型

固有処理の中から部品を呼び出す時や，既存の部品を利用して新しい部品を作るときに利用する。呼び出す部品識別子とともに文字列変数の値を一緒に記述する。

②対話型

呼び出しを可変にした部品を適用する場合に使用する。部品の中で未解決の文字列変数をツールが画面上に表示し，これに答えるかたちでツールと対話しながら値を指定していく。

7) デバッグ・保守（図2.5-17［12］［13］）

デバッグ・保守はツールで出力するドキュメントを使用して行う。以下にツールが出力する２８種のドキュメントを示す。

①保守ドキュメント（２５種）

・プログラム設計書

－スケルトン仕様書（画面遷移，プログラムの入出力情報など５種）

－スケルトン，部品，固有処理の構造図

－固有処理仕様書（ＹＡＣⅡ仕様書，決定表，編集表）

・その他設計書（データ項目情報など５種）

・相互参照情報（部品→プログラムなど５種）

・一覧情報（部品一覧など３種）

②デバッグ用ドキュメント（３種）

・部品名トレースリスト

・領域表示リスト

・合成ＹＡＣⅡ仕様書

エラーを修正する場合は合成された情報でなく，ごうせいうのもととなった部品や固有処理に対して行い，再度プログラムの合成と最新ドキュメントの出力を行う。

２．６　ＩＭＡＰ

増大するソフトウェア需要に対応するには、ソフトウェア生産の工業化が社会的な緊急課題である。

このような工業化に対しては、生産工程の明確化による分業化の実現、管理手法の確立による管理の可視化が工業化の必要条件である。

ＩＭＡＰ(Integrated software MAnagement and Production support system)は、1)ソフトウェアの一貫生産ツールの開発、2)成果物管理のデータベースの構築、3)管理データの自動収集、4)管理目的に応じた支援ツールの構築、5)ＥＷＳ(Engineering Work Station)を中心とした分散環境による生産環境を実現し、生産と管理を融合したソフトウェアの工業的生産の汎用化のコンセプトを確立した。またＩＭＡＰの基本思想は現在の一貫ＣＡＳＥ環境の基礎であるといえる。

２．６．１　ＩＭＡＰの基本思想

ソフトウェア生産の工業化とは、生産過程を工程に分割し、各工程間の作業基準を規定し、かつ、品質基準を満たした製品を指定期間内に一定工数で生産出来るようにする事である。そしてこのようなソフトウェア生産の工業化の実現を目的とした一貫ＣＡＳＥ環境ＩＭＡＰの基本コンセプトは以下の項目を実現する事である。

1)情報資産化（Ｒｅｕｓａｂｌｅ　Ｅｎｇｉｎｅｅｒｉｎｇ）

計算機内の全データ、分析・設計情報、ノウハウ、分野知識等の全てを再利用単位として管理・運用しシステムの資産とする。

2)ツール自体が手法、方法論のエキスパート

個々のＣＡＳＥツールを単なるグラフィックエディタ、テキストエディタとしてではなく、基礎となる方法論を実現し利用者をナビゲートする。

3)個人の自由度と協調共同作業の支援

分散開発環境、勤務形態の多様化に対応していくため、技術者個々の個性を尊重すると同時に、組織としての目標を達成するためにプロジェクト全体の管理を行う。

4)管理情報の視覚化と見せない管理

品質・進捗管理等を視覚化し管理者を支援すると同時に、管理データの自動収集により個々の技術者には管理されている事を意識させない。

そしてこれらのコンセプトを実現するためのＩＭＡＰの基本方針として以下の項目が挙げられる。

1)管理手法の確立
2)作業手順の明確化と分業化
3)部品化・再利用の促進
4)生産環境の確立
5)技術者育成のための教育体系の確立
6)グループコンピューティングによる分散開発環境の確立
7)一貫データベースの確立

（１）管理手法の確立

ソフトウェアの生産は知的生産活動であり、生産過程においても、成果物においても、その詳細な意味内容を客観的にみることは出来ない。このため管理者は、技術者の報告を自己の体験と照らして憶測するしかなく、進捗状況、問題点の把握に苦慮している。従ってソフトウェアの詳細内容を理解できなくても、ソフトウェアの生産に直接関与していない第三者が、客観的に現状を把握でき、適切な管理行動がとれる管理手法が必要である。

管理手法の確立には、客観的な実績データの収集が原点である。基礎となる実績データの収集は、正確性、客観性、即時性などの理由により出来る限り技術者の手を煩わせることなく自動的に収集出来るようにするべきであり、そのための制度・機構を提供しなければならない。そして収集されたデータは工程管理、品質管理、成果物管理等の管理目的に応じてモデル化、

統計処理により所要の管理資料に変換する。こうすることにより、抽象的知的成果物であるソフトウェアの管理は、数値化され客観化し、図表などを用いて可視化が行われ、視覚的な管理が実現される。また、データの自動収集等の機構により技術者が管理されているという事を意識することなくソフトウェアの開発に専念する事が可能となる。

さらに、管理者と技術者は数値化された指標の性質や傾向を理解する事により、両者の間で効果的な管理手法を確立する事が出来る。

（２）作業手順の明確化と分業化

ソフトウェア生産の各工程の作業手順をソフトウェア工場として規格化し、各工程で使用する用紙、用語、記述方法を標準化する。これは各工程への入力データ、出力としての成果物を明確にし、その成果物の品質基準を規定することである。

工程間の受け渡しでは、前工程で品質が保証された成果物を入力として該当工程で加工し、成果物の品質を保証して次工程へ引き渡す。分業化とは工程ごとの分業と機能毎の分業の二通りの意味がある。前者は生産の各工程を担当する専門家により、要求仕様から次々に加工され、製品として完成する事を意味する。後者は、直接生産に携わらないが、品質の評価や、管理データの収集や分析、成果物の管理、後述の標準部品作成など間接的に生産を支援する専門家による組織化を意味する。

ＩＭＡＰではソフトウェアの品質保証管理体系(Quality Control Program)を工程、管理階層、管理対象、管理サイクル(Plan-Do-Check-Action)の構成要素によりＱＣＰ(Quality Control Process Chart)として体系化している。この特徴は、1)分業化の基底となる開発工程とその作業手順、2)ＤＲ（デザインレビュー）、ＷＴ（ウォークスルー）を行う単位、3)作業のインプットとアウトプット、4)品質のチェックポイント、5)責任担当部門、を明確に示したことである。これにより、管理者は管理のマイルストン毎のポイントを、開発担当者は、作業手順の進め方のノウハウを再利用して品質向上が図れる。

（３）部品化・再利用の促進

ハードウェアと同様、ソフトウェアにおいても、品質の保証された標準部品の存在が工業化の前提となる。

従来でも同じ分野の類似ソフトウェアの生産において、既存ソフトウェアの流用、あるいは

繰り返し使用される部分を“切り出し”再利用することは行われていた。特に同一組織による類似ソフトウェア作成の場合、６０％以上再利用が可能であり、生産性向上に大いに貢献している。しかし切り出された部品を他の分野あるいは、他の組織のソフトウェア生産に再利用しようとすると、切り出された部品の理解や修正に手間取り、新規に作成する工数より多くかかったり、トラブル時には対応しきれないなどの課題もある。

ソフトウェア生産工業化における標準部品とは、既存ソフトウェアからの“切り出し”ではなく、再利用する事を前提とした部品を標準部品規定に基づき事前に作成し、品質を保証し、正式な手続きを踏んで認知された部品の事である。この標準部品規定には、部品使用の記述法、インタフェース仕様、品質基準と検査方法等は当然として、標準部品を認知する正式機関の位置づけ、権限、構成メンバの規定も含まれる。

このようにして作成された標準部品の再利用を促進するためには、部品の蓄積・検索・払い出しが容易にできる支援ツールの用意が必要である。設計してから、仕様にあった部品を検索するのではなく、標準部品に合わせた設計手法を確立する必要がある。品質の保証された標準部品を使用する事は、生産性向上だけでなく、ソフトウェアの品質向上にとっても不可欠のことである。

（４）生産環境の確立

ソフトウェア製品が、工芸品から工業製品化するためには、工場全体の設備計画や組織体制に裏打ちされた生産環境の確立が必要である。設計者の個性により、ソフトウェア製品設計、生産設備、支援ツールが異なっていたのでは、ソフトウェア工場としての体をなさない。つまり生産環境の確立とは、組織としての分業体制を確立し、工場全体としてのソフトウェア生産管理システムを確立し、各部門共通の汎用設備と、部門毎、分野別あるいは製品別専用設備を明確にし、ソフトウェア工場全体として設備の有機的結合を図ることである。

このようにして、ソフトウェア製品の流れと管理が一体化し、工場として品質保証した製品を制作できる環境整備が必要である。

（５）技術者教育のための教育体系の確立

ソフトウェアの生産は技術者の能力に依存する要素が大きく、ソフトウェア生産の工業化、あるいは一貫ＣＡＳＥ環境の実現にとっても技術者の“しつけ”が重要なテーマである。従っ

てＩＭＡＰにおいては“教育”を環境の構成要素としてとらえている。

教育の主なねらいは、職人から企業人への変革であるが、

1)作業基準、標準化規定の徹底
2)全体システムの理解
3)最新技術の取得

などが主な項目である。技術者のレベルアップ、モラルアップこそソフトウェア生産工業化の基礎である。このためには、教育体系をくみ、計画的に組織的に継続する事が必要である。

（６）グループコンピューティングによる分散開発環境の確立

ソフトウェア生産現場における作業形態は、汎用大型計算機を中心にしたＴＳＳによる集中型作業形態から、ＥＷＳ(Engineering Work Station)の出現によりソフトウェア開発グループ単位あるいは技術者個人単位の分散型作業形態に急速に移りつつある。またこれはソフトウェア開発量の増大・規模の拡大・技術者確保の困難化等の問題解決という社会的要請に応えるのが分散開発環境ということもできる。

しかしソフトウェア開発は、本質的には技術者の創造的作業であり、技術者個々の個性にあった方法で作業が進められる事が望ましい。またソフトウェアは工業製品であり、要求された仕様を満たし、決められたスケジュール内で定められた手続きを踏んで開発を行わなければならない。つまり分業・分散開発には、技術者間のコミュニケーションを如何にしてスムーズに行い、工程管理、成果物管理、セキュリティ管理等をどのように行い、技術者の個性を尊重し組織としての目標を達成するかという課題がある。このような課題を解決する手段としてグループコンピューティングがある。グループコンピューティングとは、

1)技術者個人の計算機環境と組織の計算機環境の融合
2)技術者の個性の尊重と組織目標の協調

を実現することである。

（７）一貫データベースの確立

ソフトウェア開発の各段階において生成される情報を有効に利用し部品化・再利用の促進あるいは、情報資産化を実現していくためにはソフトウェア開発の全工程を一貫して管理する一貫データベース、一貫ＳＥＤＢ(Software Engineering Data Base)の役割が重要である。このような一貫ＳＥＤＢでは各工程で生成される成果物だけでなく、分野依存知識、技術者のノウハウ、情報間の関係等の情報が管理される。但し、実際に一貫ＳＥＤＢで管理する情報は、各応用分野、組織等で定められている方法論に基づき必要な物だけを管理対象とすれば良い。つまり独自の方法論、作業手順において必要な情報をモデル化し、そのモデルに従ってデータベースをカスタマイズしていく。

２．６．２　ＩＭＡＰの支援体系

ＩＭＡＰは前節で述べた基本思想を実現する総合的なソフトウェア生産支援システムである。まずＩＭＡＰでは、工業化という観点からシステムの構成を見直し，1)ソフトウェアライフサイクルを支援するシステムに加えて、2)共通支援ツール群、3)生産支援環境、4)管理システムの４ブロックからなる図２．６－１に示すような支援体系をとった。標準部品の再利用を支援するツールや、機種毎、ＯＳ毎の違いを吸収するツール群を2)共通支援ツール群として分類し、機種毎の違いにとらわれない汎用的な1)ライフサイクル支援システムと区別し、技術移転の容易化を図っている。またソフトウェア生産という観点から、ネットワーク、データベースなどの3)生産環境をＩＭＡＰの基盤として開発する。また工業化という観点から、生産支援と対等な立場で4)管理支援を位置づけ、各ツールの開発を行う。

ライフサイクル上で特徴的な工程として、“５０ＳＭ製造”と“トラブル解析”がある。“５０ＳＭ製造”はＩＭＡＰの特徴の一つであり、思想を具体化したものである。５０ＳＭとは全てのモジュールは、５０ステップ以内にするということである。上位、中位、下位のどのレベルであっても、一つのモジュールは５０ステップ以内と規定している。これは、モジュール作成の容易化、モジュールの品質保証、モジュールの理解性向上のための積極的制約である。次に“トラブル解析”工程であるが、これは通常“保守”といわれる工程である。“保守”という言葉には、“不具合の修正”とともに“機能の拡大”の意味にも使われる。ＩＭＡＰでは、

前者の意味を強調するために“トラブル解析”とした。後者は既存のソフトウェア製品を利用した再生産であると考え、ライフサイクルの各工程を最初からすばやく流れれば良いと考えており、Reverse-Engineering/Re-Engineering等の手法を確立していく。

ソフトウェアライフサイクルを支援する特徴的なツールとして、システム提案を支援する発想支援システム“知恵の泉”、ソフトウェア設計からプログラムの変換までを支援するＴＦＦ(Technical description Formula for Fifty steps / module design)エディタ、レビュー・検査を設計段階で行うためのＤＯＴ(DOcument Tracer)、設計支援システムと連動した部品蓄積・管理・検索システムとしてＳＲＥＤ(Software module Reuse Environment to assist software Development)、ソフトウェアの作り方を形式的に評価するＥＳＱＵＴ(Evaluation of Software Quality from User's view poinT)、技術者の工数管理、スケジュール管理のために携帯用の管理データ収集ツールＭＹＳＴＥＰ等があり、一貫ＣＡＳＥ環境を目指した各種ツールを提供している。またＩＭＡＰにおける一貫ＣＡＳＥ環境の構成は図２．６－２のようになっている。

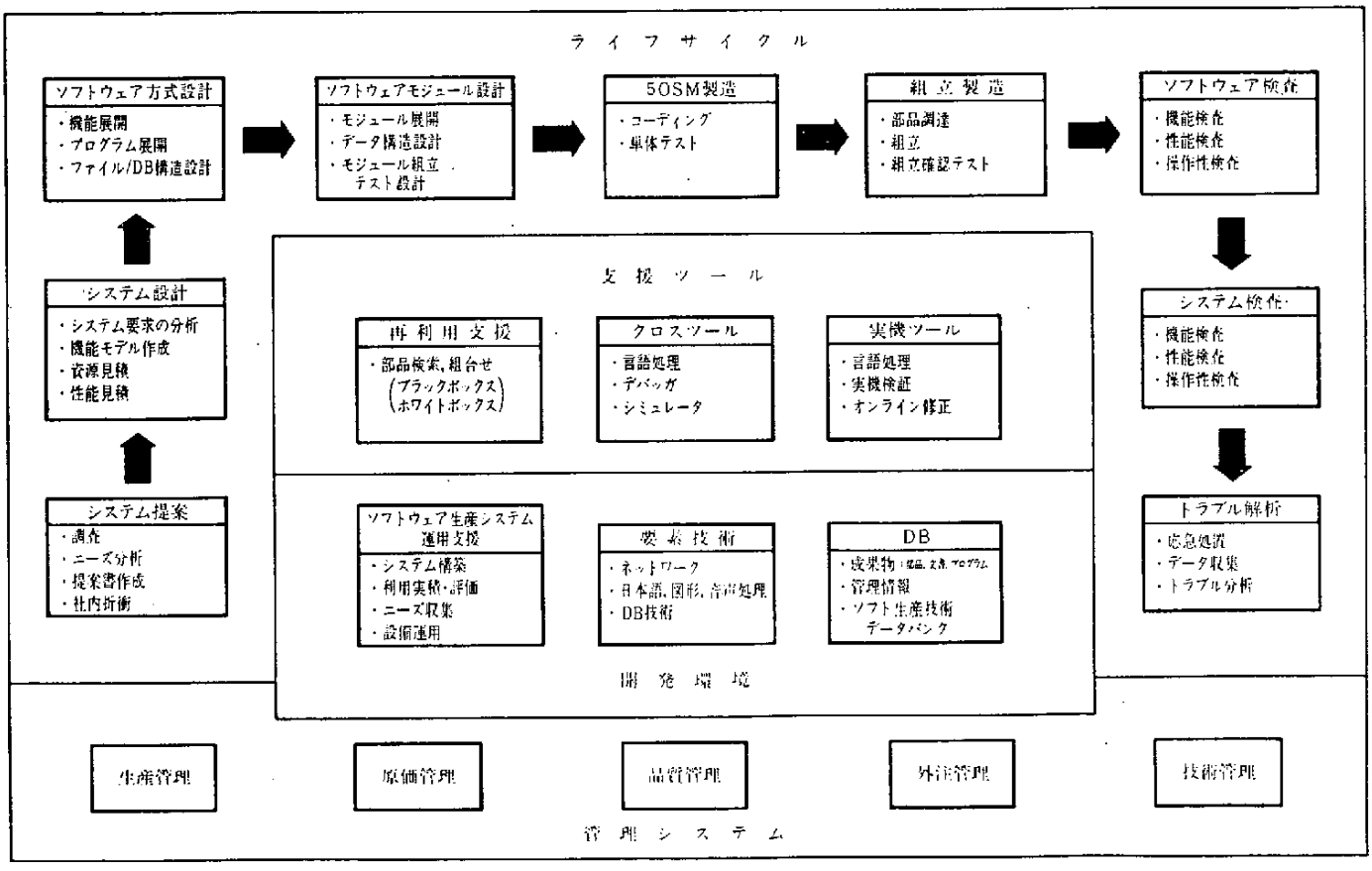


図２．６－１　ＩＭＡＰソフトウェア生産支援体系図

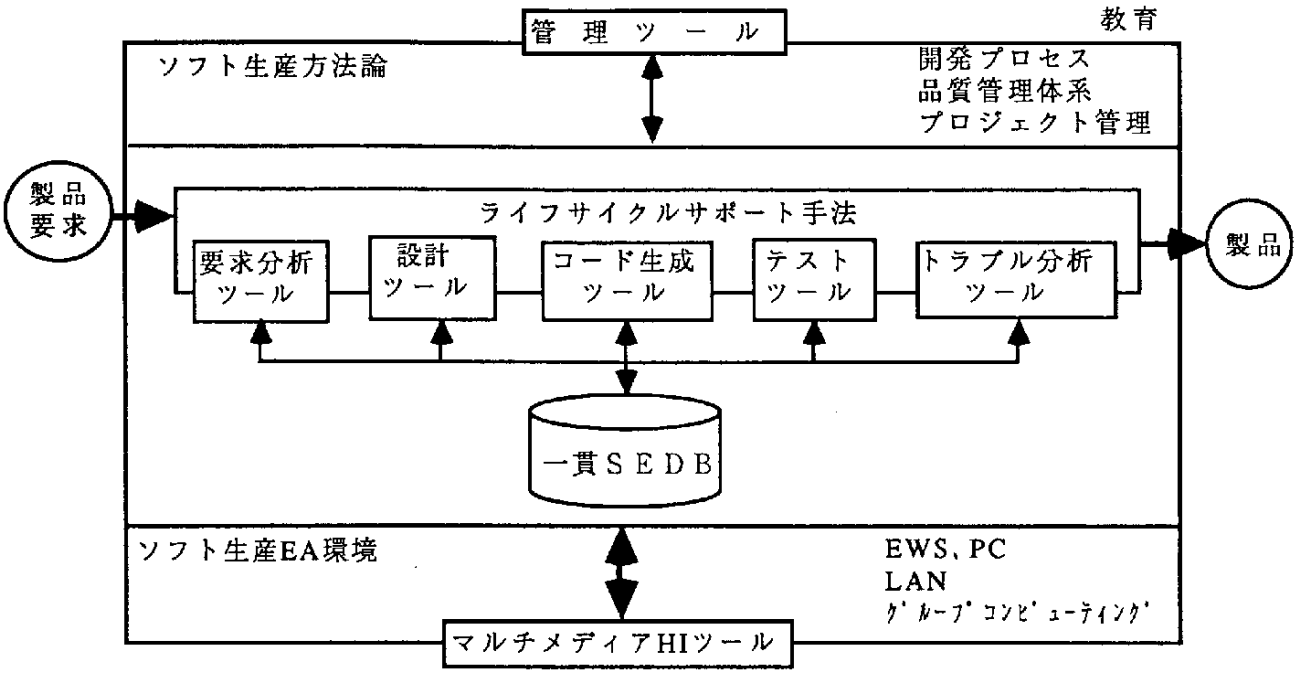


図２．６－２　ＩＭＡＰにおける一貫ＣＡＳＥ環境の構成

２．６．３　ＩＭＡＰの基本思想に基づく実現例

ＩＭＡＰのソフトウェア生産工業化、一貫ＣＡＳＥ環境実現の基本思想に基づき開発された、各応用分野の実現例を示す。制御系のＣＡＳＥ環境としてＮｅｗ－ＳＷＢ(New-Software Work Bench)，事務処理系のＣＡＳＥ環境としてＭＹＳＴＡＲを取り上げる。

（１）Ｎｅｗ－ＳＷＢ

Ｎｅｗ－ＳＷＢは制御系を対象としたＣＡＳＥ環境であり、事例としては火力／原子力発電システム、電力系統システム、鉄鋼システム、上下水道システム、ビル管理システム等がある。Ｎｅｗ－ＳＷＢの基本思想はソフトウェア開発で行うすべての作業をできる限り計算機で支援しようとするものであり、ハードウェアの“物の生産ライン”と同様に、“ソフトウェア生産の流し化（図２．６－３）”を実現しようとするものである。

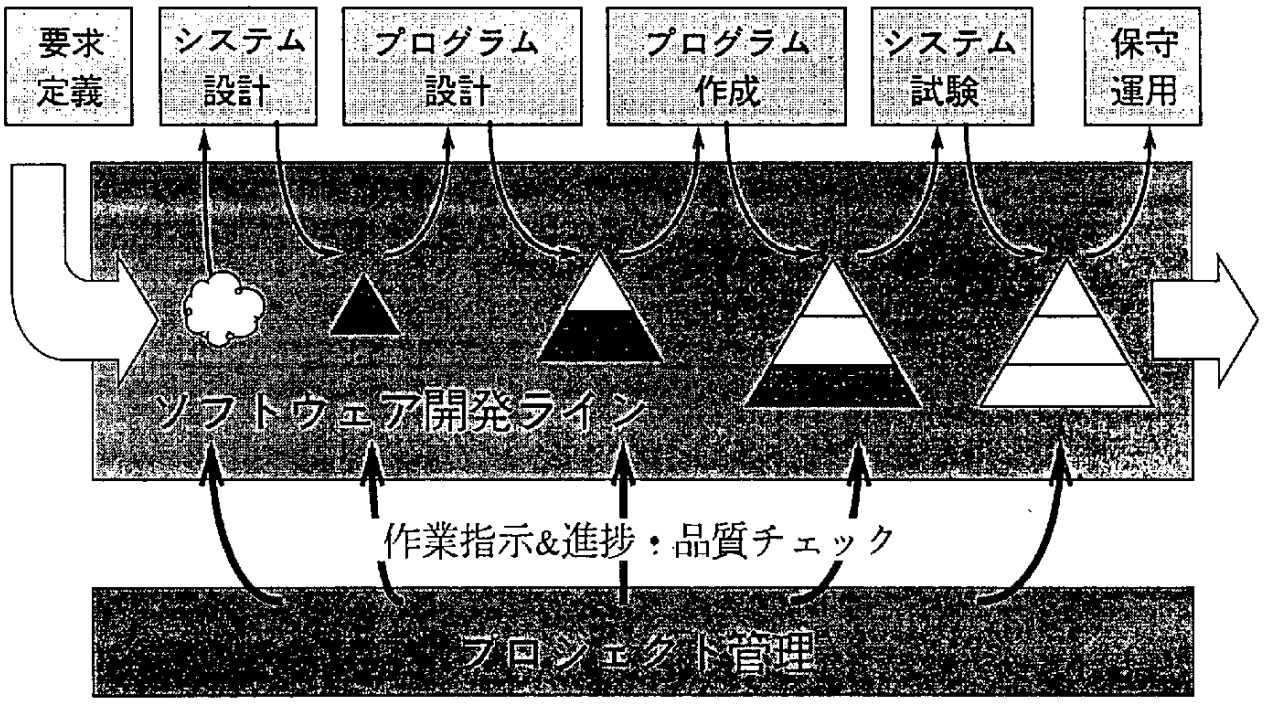


図２．６－３　ソフトウェア開発の流し化

Ｎｅｗ－ＳＷＢでは以下のような支援を行う各種のツールを提供している。

1)ユーザインタフェース支援

2)システム設計支援（図２．６－４）

　　文書作成支援、画面作成支援、システム分析・設計支援、データ設計支援

3)プログラム開発支援（図２．６－５）

　　構造設計支援、詳細設計支援、プログラム作成支援、プログラム試験支援

4)プロジェクト管理支援

　　構成管理支援、品質管理支援（図２．６－６）、生産管理支援、運用管理支援

5)ネットワーク支援

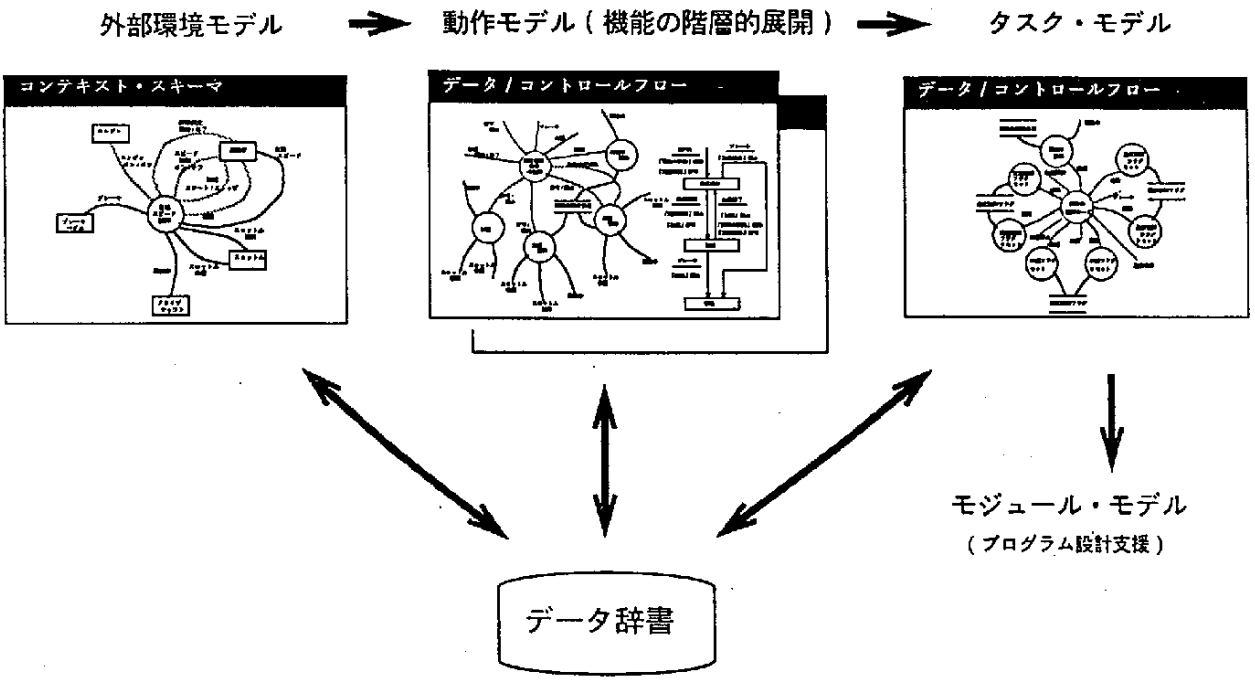


図２．６－４　システム分析・設計支援

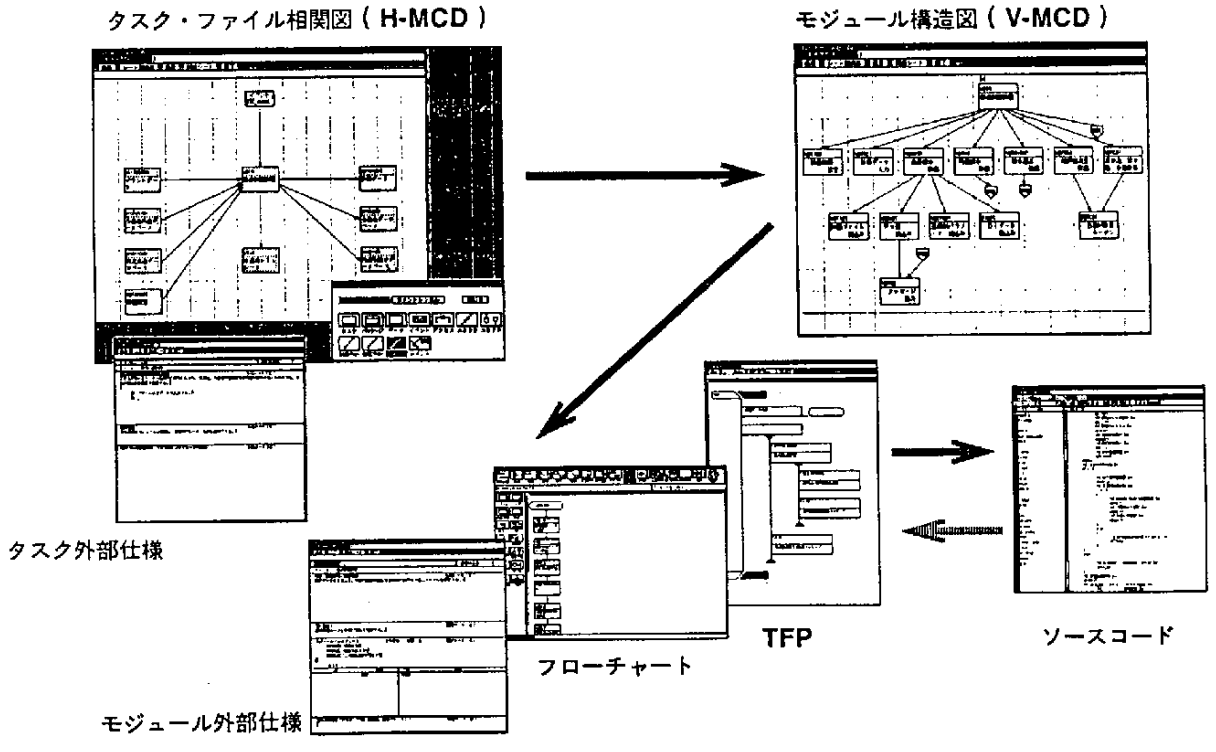


図２．６－５　プログラム設計支援

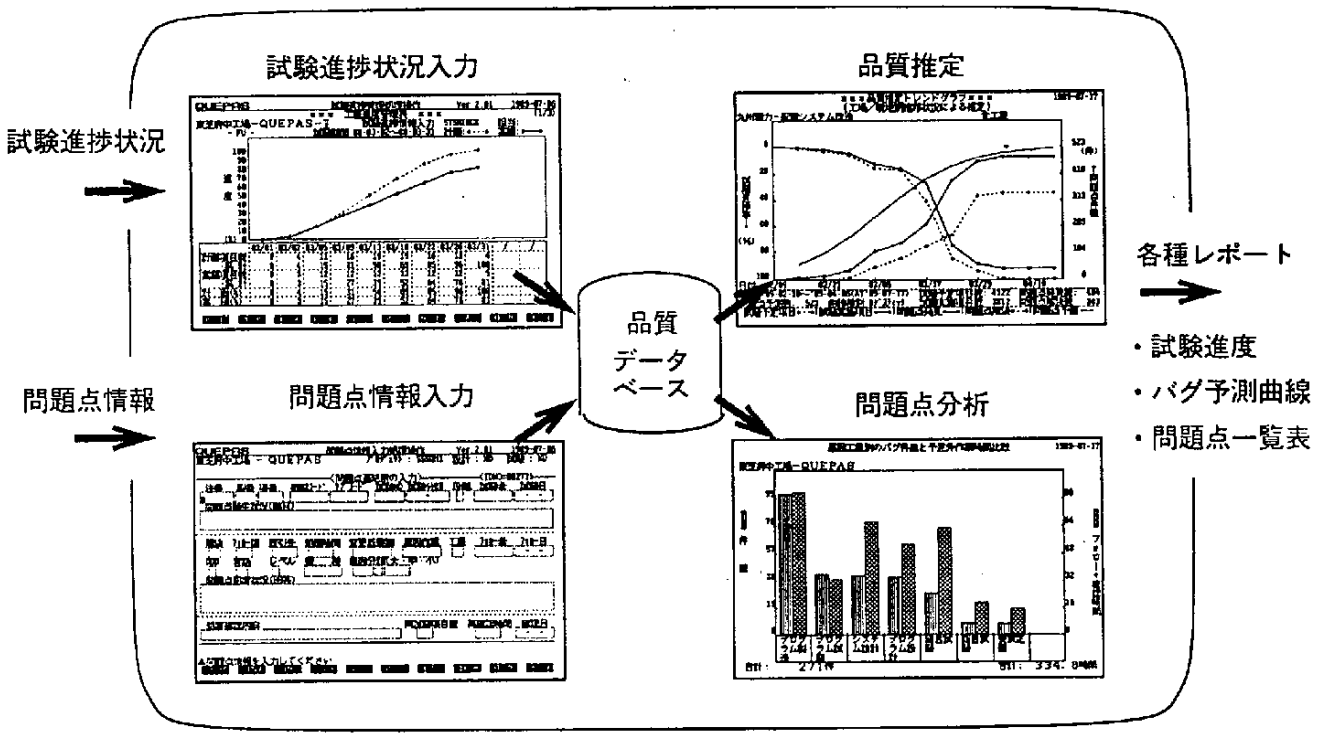


図２．６－６　品質管理支援

（２）ＭＹＳＴＡＲ

ＭＹＳＴＡＲは事務処理系を対象としたＣＡＳＥ環境であり、ソフトウェア開発工程を詳細に渡って標準化し、各工程の成果物をフォームシートの形で規定している。利用者は画面上のフォームシートに必要な情報を入力していくことによりアプリケーションの開発が行える。

まず開発標準としてシステム分析標準、システム設計標準、プログラム設計標準、プログラミング標準、テスト標準を明確に定義し、この標準に基づき各種フォームシートを規定している。そして開発支援システムとしてソフトウェアライフサイクル支援システムの他、画面様式定義支援システム、帳票様式定義支援システム、画面プロトタイピングシステム等の支援システムも提供されている。またＭＹＳＴＡＲでは、システム全体を管理するために工程管理支援システム、シナリオシステムがあり、個々のツール群を総合的に利用するために統合設計支援システム、設計データ相関辞書が提供されている。ＭＹＳＴＡＲにおけるソフトウェア開発手段を図２．６－７に、レコード定義のためのフォームシートの例を図２．６－８に示す。

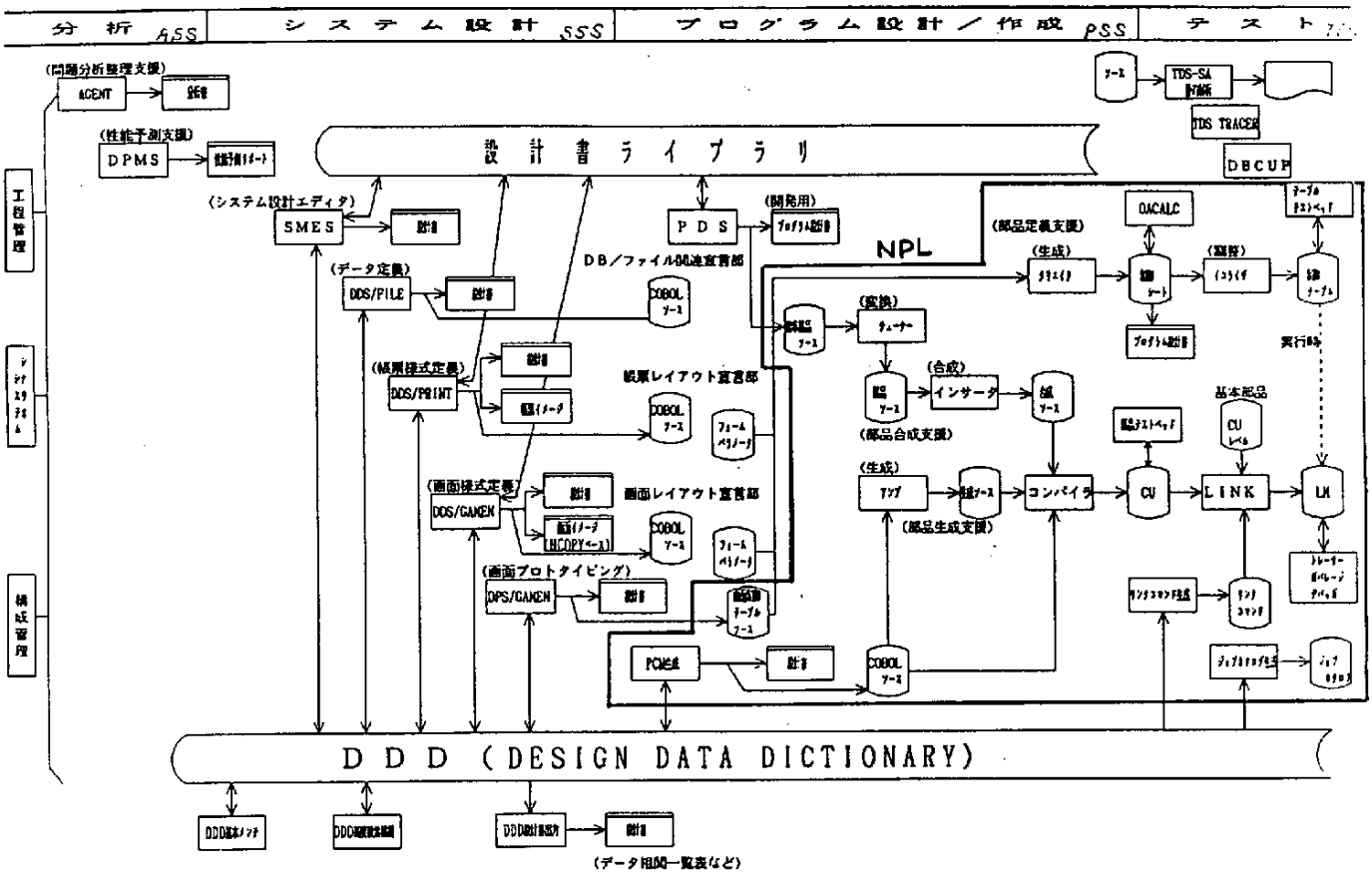


図２．６－７　ＭＹＳＴＡＲにおけるソフトウェア開発手段

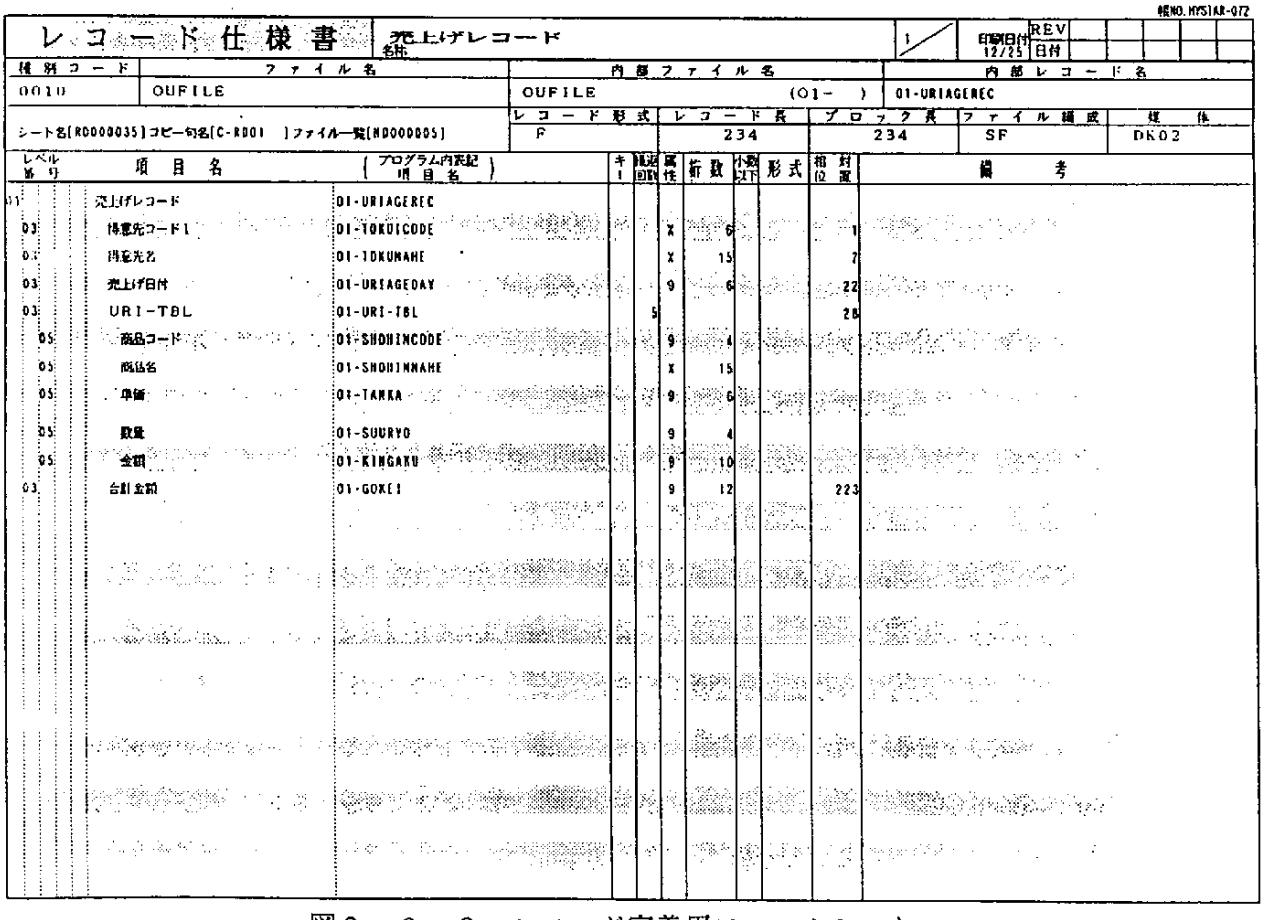


図２．６－８　レコード定義用フォームシート

２．６．４　将来構想

ＩＭＡＰにおいては、今後一貫ＳＥＤＢの一貫性をより向上させると共に、

1)発想支援、要求獲得による要求分析の前段階の支援、

2)部品合成による上流ＣＡＳＥと下流ＣＡＳＥの接続、

3)イメージによるプロトタイピング、

4)検証機能を充実させる、

といった点に注力し一貫ＣＡＳＥ環境の実現を目指す。

また一貫ＣＡＳＥ環境の実現に際しては、特定分野に依存しないような汎用の能力を持った一貫ＣＡＳＥ環境を実現するのは不可能であると考えている。何故なら各応用分野で使用される手法・方法論は異なっており、たとえ同一分野でも組織・部署によっては異なる手法・方法論が使用され独自のライフサイクルに沿って開発が行われている。従って、まず一貫ＣＡＳＥ環境を実現するためのフレームワークを提供し、各分野、組織、部署に応じてカスタマイズしていかなければならないと考える。つまり、一貫ＣＡＳＥ環境を実現する第一段階は個々の分野あるいは組織における開発工程を明確に定義し、厳密な方法論を確立することである。

2．7　ＳＥＡ／Ｉ

2．7．1　ツールの概要

ＳＥＡ／Ｉは、企業レベルの基幹業務システムの開発・保守の効率化をねらいとして、ソフトウェア開発の全工程を体系的に支援するツール体系である。

ＳＥＡ／Ｉの基本的な開発思想は次の項目からなる。

（1）一貫したシステム開発環境の提供

上流工程（システム分析）から下流工程（製造・保守）にいたる作業を支援するツール群の連動により、重複した操作や無駄な作業を除去した、一貫したツール体系を提供する。

（2）ソフトウェア資産の再利用

プログラムの処理部分の制御構造、共通処理およびデータ仕様をコンポーネント化し、その再利用により生産性を飛躍的に向上させる。そのために、コンポーネントの定義、管理及び再利用をしやすくするための環境を提供する。

（3）ソフトウェア資産の一元管理

必要な情報を必要な時点で、必要な形で取得できる環境を提供し、資産の管理ならびに保守の効率化を実現する。

（4）開発作業の視覚化、自動化

設計作業のような、人間の判断、指示、試行を必要とする作業を容易にするために視覚的な開発環境を提供する。例えば、上流工程における業務分析の内容や、画面、帳票、ファイル、プログラム構造等の仕様を図式で表現することにより、ソフトウェア仕様の設計ならびに理解を容易にすることができる。

また、仕様からオブジェクトへ変換する作業の自動化を実現する。

（5）言語の高度化

高生産性言語システムとして第４世代言語（ＩＤＬⅡ）及びその対話的プログラム開発環境（ＩＤＬＴＯＯＬ）を提供する。

ＳＥＡ／Ｉは、図2.7-1に示すように、いわゆる上流工程を支援する業務設計支援システムＳＰＥＣＤＥＳＳＩＮおよびＣＯＢＯＬベースのソフトウェア開発支援システムＳＯＦＰＩＡと、第４世代言語システムＩＤＬⅡから構成されている。ＳＰＥＣＤＥＳＳＩＮを利用して作

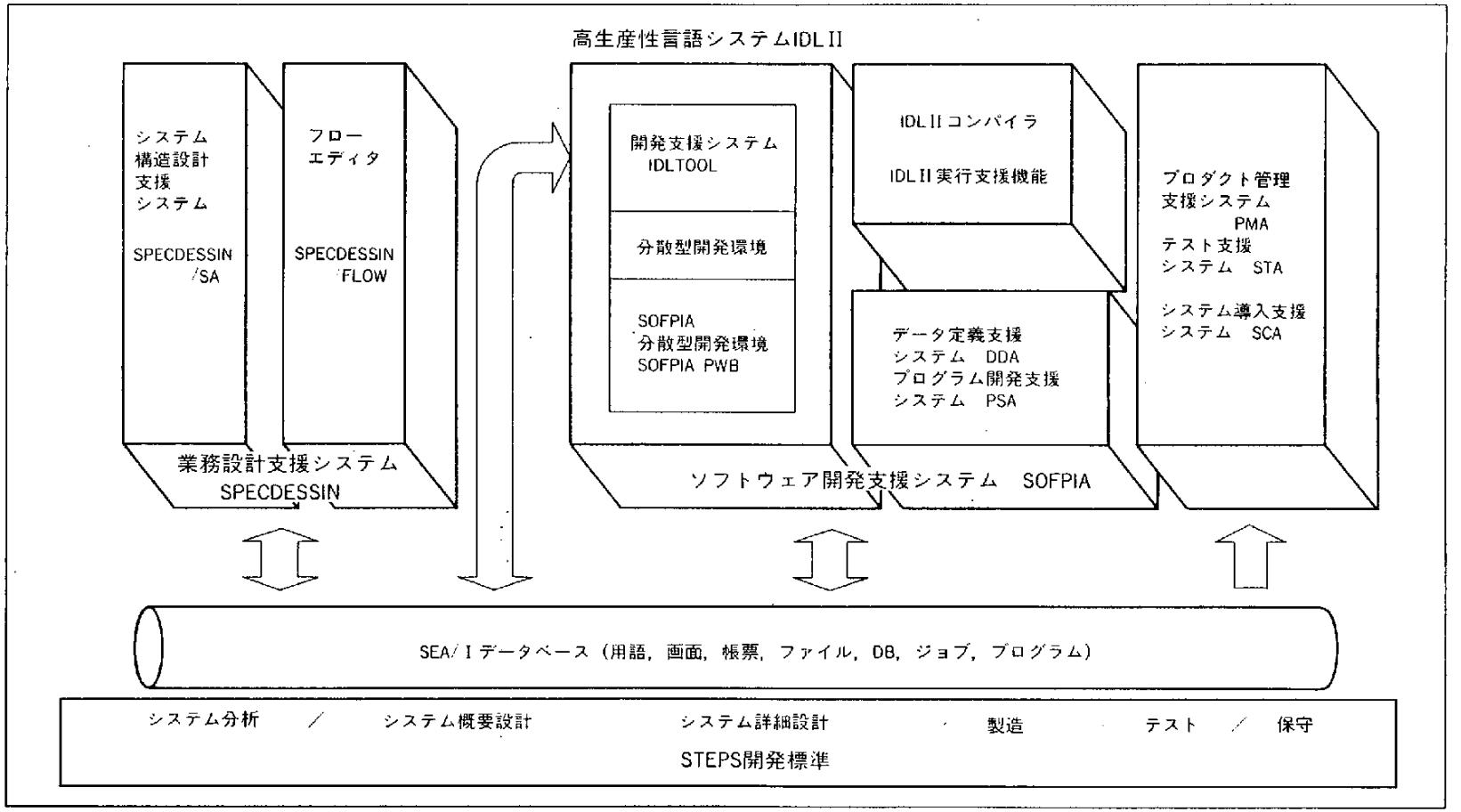


図2.7-1 ＳＥＡ／Ｉの体系図

成された設計情報は、詳細設計や製造工程の支援システムであるＳＯＦＰＩＡやＩＤＬⅡへ引き継がれ、ライフサイクルを通した一貫したソフトウェア開発支援を実現する。

次に、ＳＥＡ／Ｉのそれぞれの構成システムの概要を説明する。

（1）ＳＰＥＣＤＥＳＳＩＮ

ＳＰＥＣＤＥＳＳＩＮは、ソフトウェア開発の上流工程における分析、設計作業を支援するシステムであり、ＳＰＥＣＤＥＳＳＩＮ／ＳＡとＳＰＥＣＤＥＳＳＩＮ／ＦＬＯＷの２種類のツールからなる（図2.7-2）。

ＳＰＥＣＤＥＳＳＩＮ／ＳＡは、システム分析工程における、業務システムの機能と情報の関連の分析／設計作業の視覚化と標準化を支援するツールである。業務システムの現状と実現イメージ、新システムの業務の流れを機能・情報・タイミングの観点からモデル化する手段として機能階層図、機能情報関連図、業務フロー図の３種類の図式による記述を容易にする機能を提供する（図2.7-3，図2.7-4）。これらのモデル化した情報の一貫性チェックやモデル修正の関連波及箇所の検索機能も備えている。

ＳＰＥＣＤＥＳＳＩＮ／ＦＬＯＷは、システムの概要設計工程における計算機処理フローの設計を支援するツールである。図形テンプレートの利用により、計算機処理フロー図、プログラム網図の記述手段を提供し、計算機処理フローの段階的詳細化を支援する（図2.7-5）。計算機処理フロー図は、サブシステム単位に表現した計算機系の業務の流れを表す図である。プログラム網図は、業務単位の処理手順の流れ図であり、パターン集の利用により、パターンプログラム指向のフロー設計が可能となる。プログラム網図からは、プログラム単位での入出力関係を表現したプログラム仕様書を出力する。

ＳＰＥＣＤＥＳＳＩＮにより、

・図の活用により分析・設計作業が容易

・設計情報の一元管理により、修正／再利用が容易

・最新の設計情報が文書化可能

・下流の工程へ設計情報が引き継がれ、作業の後戻りが減少

という効果がある。

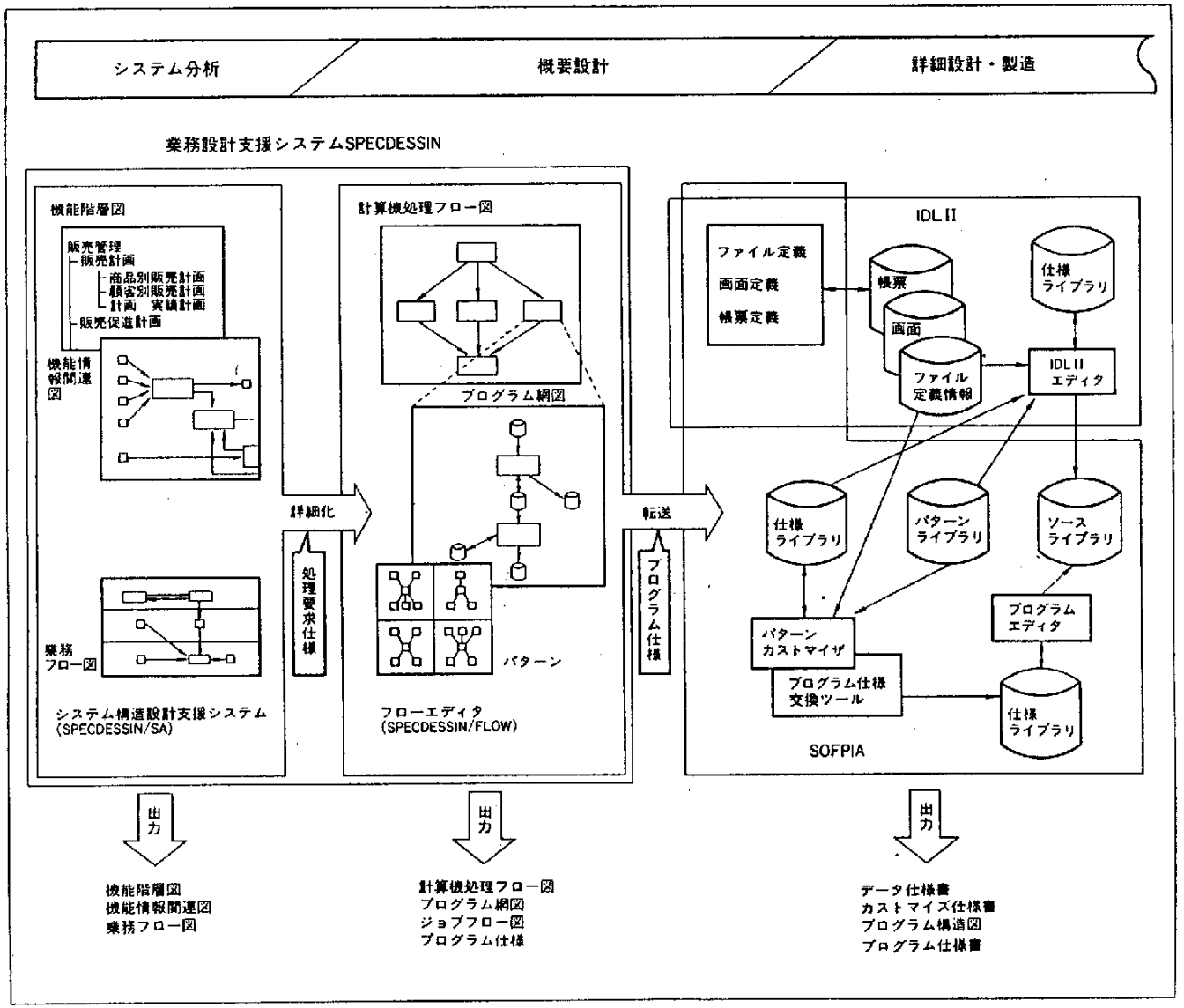


図2.7-2　ＳＰＥＣＤＥＳＳＩＮの構成と位置付け

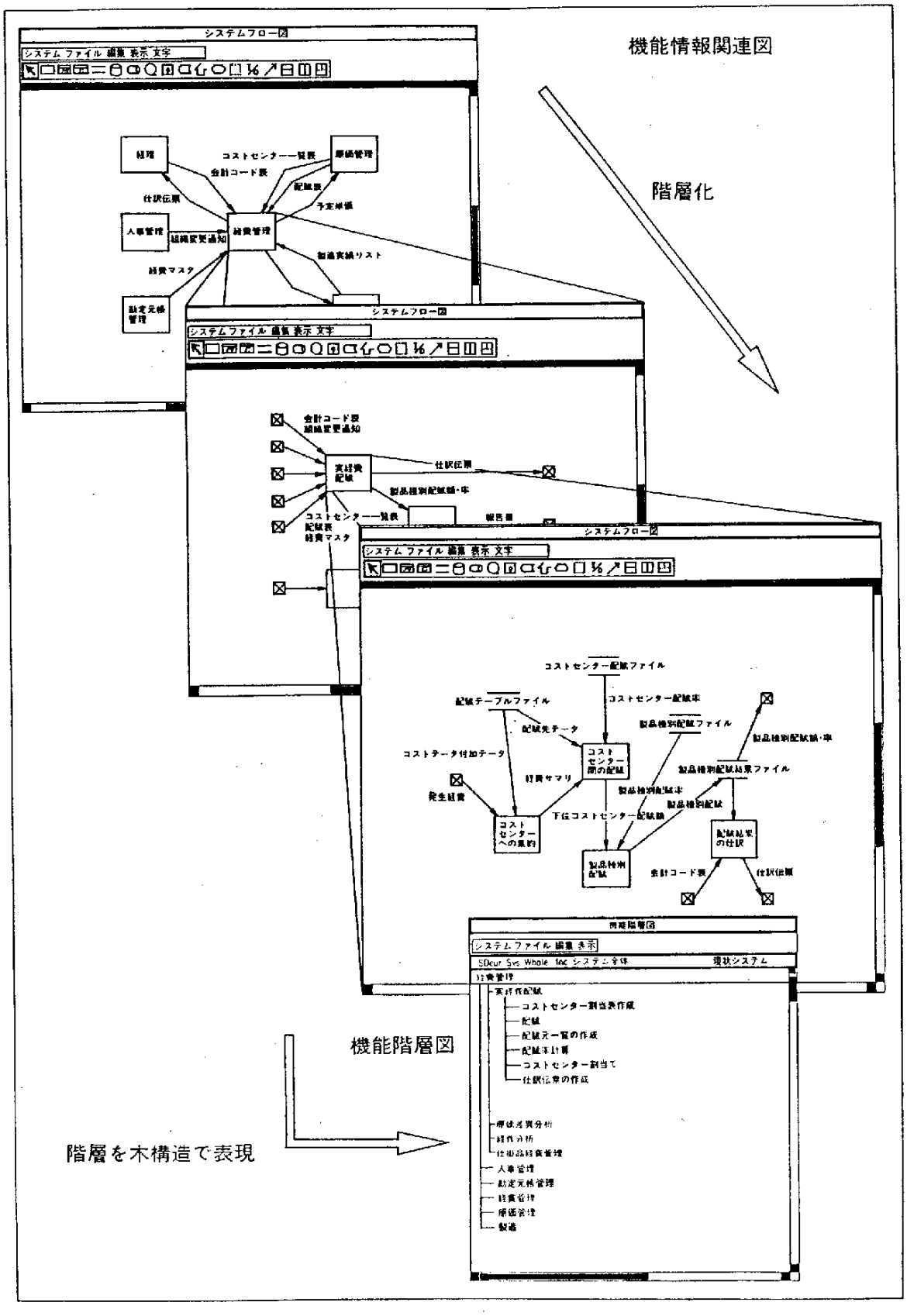


図2.7-3　機能情報関連図と機能階層図

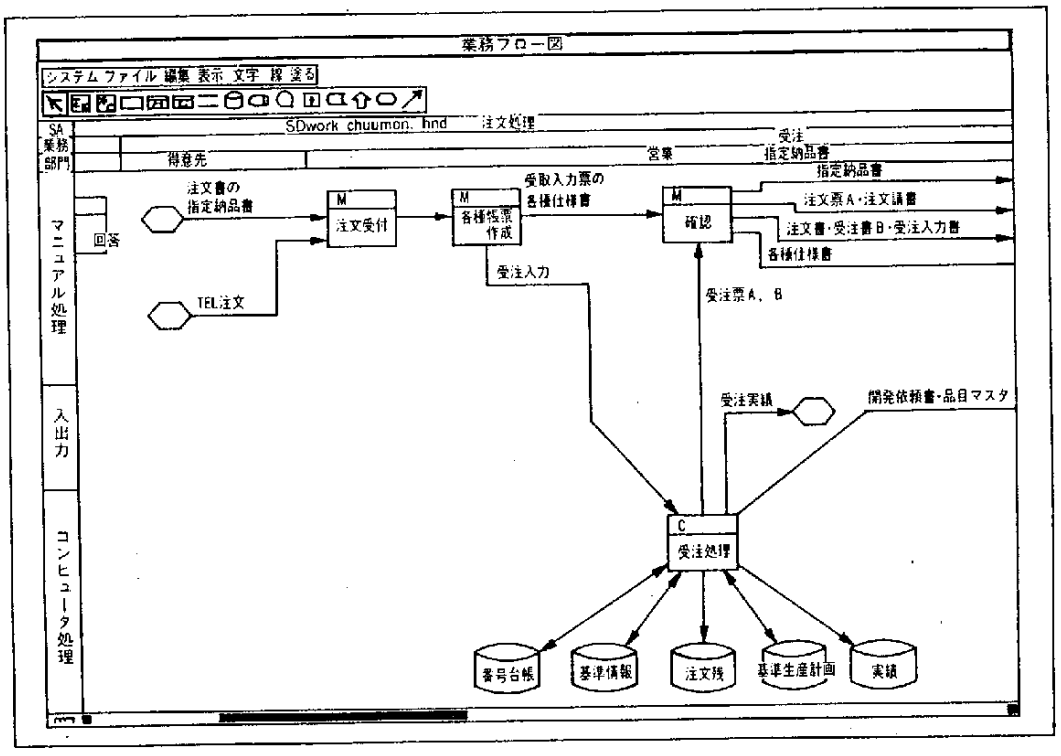
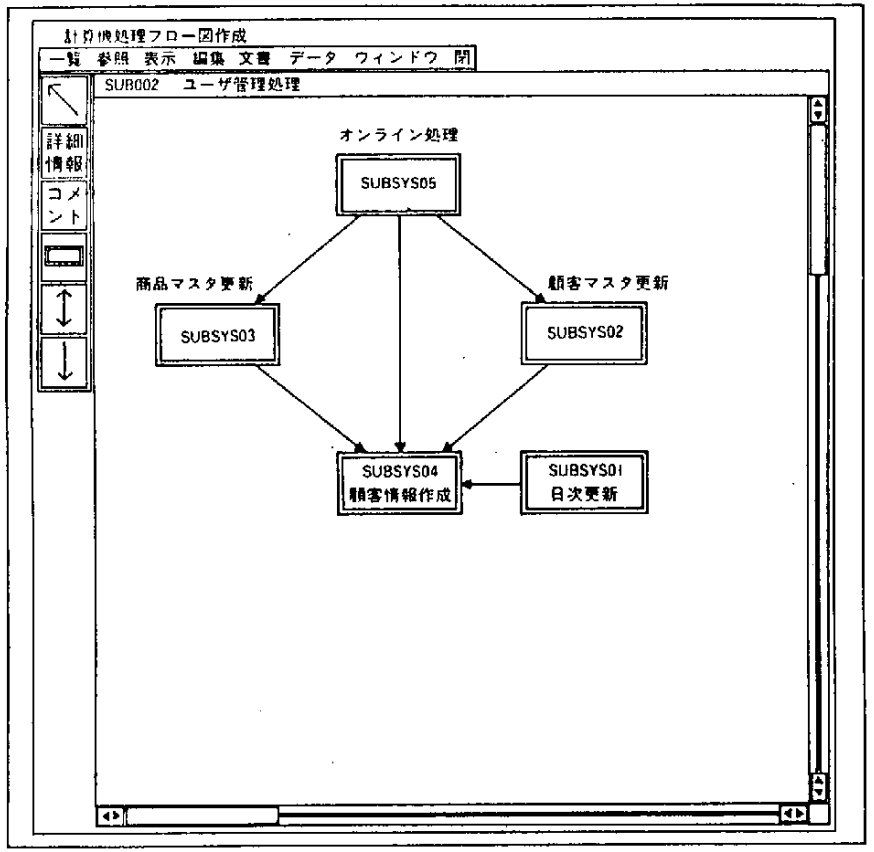
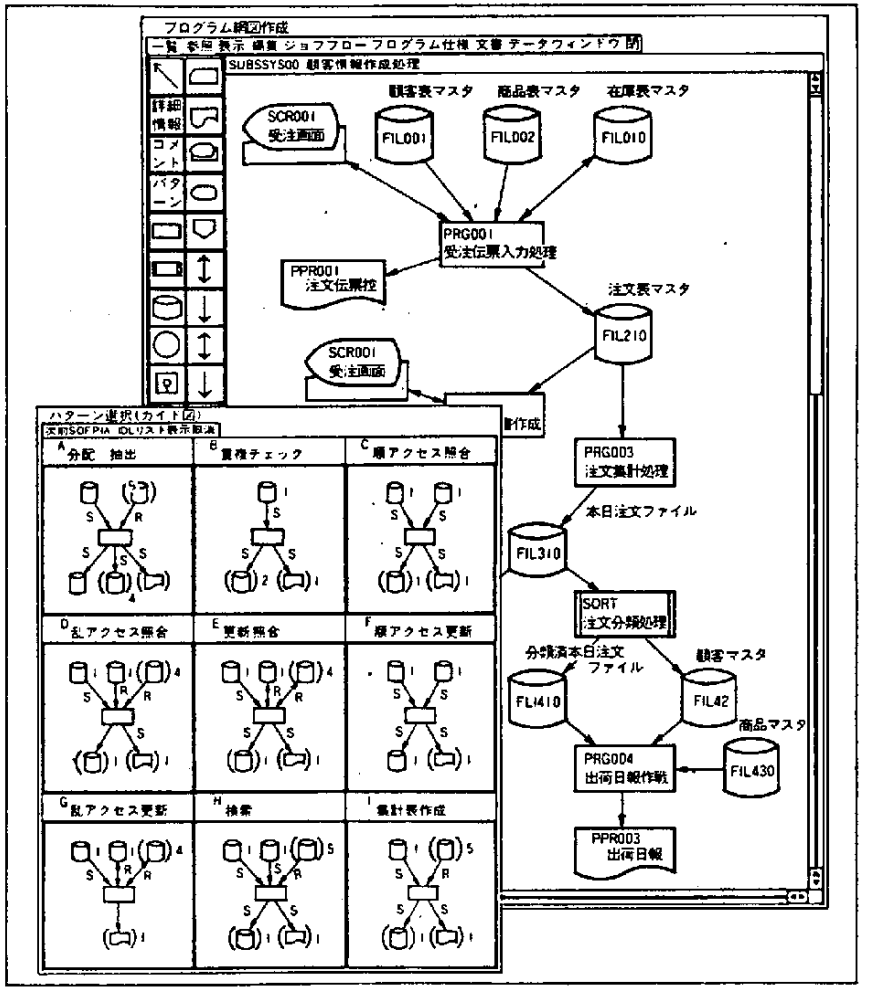

業務フロー図
システム ファイル 編集 表示 文字 線 塗る
SDwork chuumon. hnd 注文処理
SA
業務
部門
得意先
営業
受注
指定納品書
マニュアル処理
入出力
コンピュータ処理
注文書の
指定納品書
回答
TEL注文
M
注文受付
M
各種帳票
作成
受取入力票の
各種仕様書
M
確認
指定納品書
注文票A・注文請書
注文書・受注書B・受注入力書
各種仕様書
受注入力
受注票A，B
受注実績
開発依頼書・品目マスタ
C
受注処理
番号台帳
基準情報
注文残
基準生産計画
実績

図2.7-4　業務フロー図



（1）計算機処理フロー図



（2）プログラム網図（パターン利用時）

図2.7-5 SPECDESSIN／FLOW

（2）高生産性言語システムＩＤＬⅡ

ＩＤＬⅡ（Integrated Data Oriented Language Ⅱ）は、第４世代言語として言語レベルの大幅な引き上げを実現したものであり、アクション（事務処理パターン）を選択し、入出力ファイル、画面等を定義した後、入出力記述を非手続き的に定義するだけでプログラムが生成される（図2.7-6）。このとき、プログラム全体の制御ロジックや定型的なエラー処理ロジックの自動生成機能により、ＣＯＢＯＬに比べユーザの記述量を大幅に削減することができる。また、言語システムとしての対話的プログラム開発環境（ＩＤＬＴＯＯＬ）を提供しており、プログラムの非手続き的記述（仕様）をベースにしたプログラム開発が可能となる。

ＩＤＬⅡを使うことにより、適用分野をうまく設定した場合に、プログラムの生産性が大幅に向上する（３～１０倍）。

（3）ＳＯＦＰＩＡ

ＳＯＦＰＩＡはＣＯＢＯＬ系言語を対象とする開発支援システムである（図2.7-7）。

①分散型の簡易なプログラム開発環境

ワークステーション上での高度なヒューマンインタフェースを利用したプログラム設計作業とホストを接続したオブジェクト生成等のソフトウェア分散開発を支援する。

②視覚的なデータ設計支援

ファイルや画面、帳票などのデータ構造、レイアウト、項目属性の定義を視覚的に行うことができ、設計時点でシステム利用者と動作イメージの確認と調整を行うことができる。

③部品化・パターン化によるソフトウェア資産の再利用

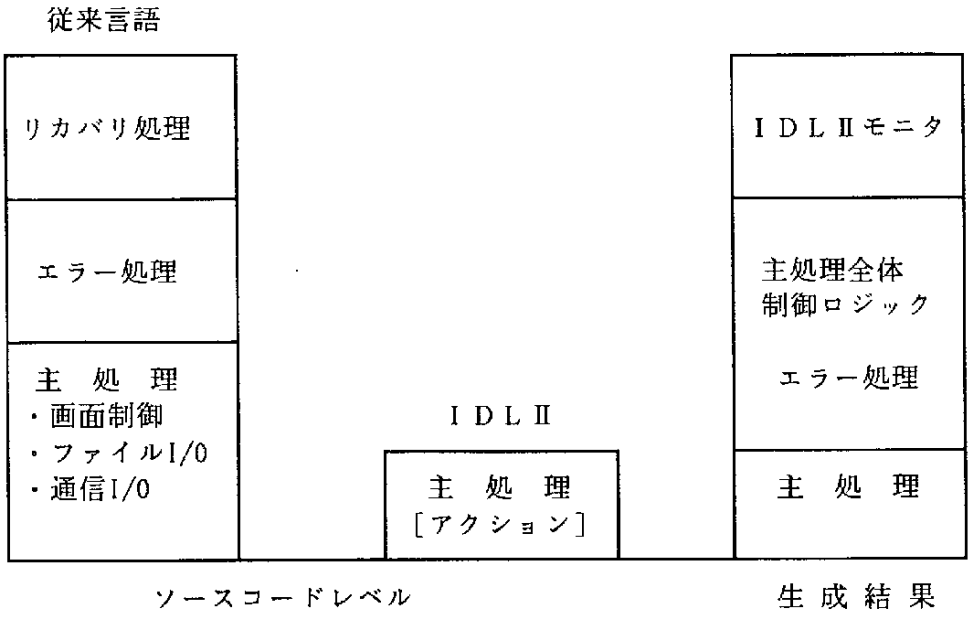
プログラムコードの部品化・パターン化のための言語や、部品とパターンの登録／検索機能、カスタマイズ機能、部品の自動組み込み機能を提供し、再利用のための環境を整備している。

④ソフトウェア資産の統合管理

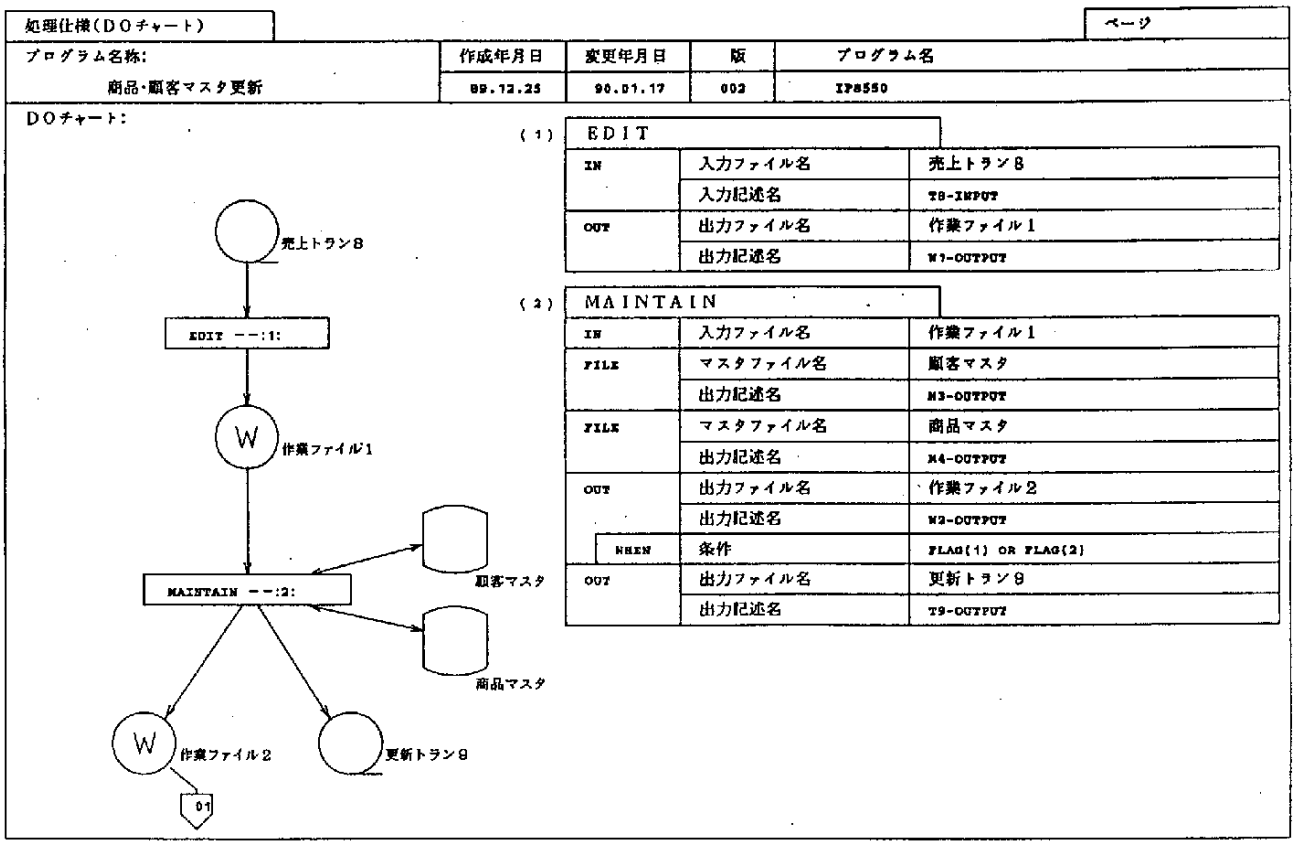
ソフトウェア資産の一元管理により、最新の資産状況の検索や文書化、仕様変更時の影響範囲の把握を対話形式で行える。

⑤業務設計支援システムＳＰＥＣＤＥＳＳＩＮとの連携

ＳＰＥＣＤＥＳＳＩＮ／ＦＬＯＷでプログラム単位毎に指定したプログラムパターンや使用ファイルの定義等の情報からプログラム構成仕様を出力することができる。ＳＯＦＰＩＡでは、このプログラム構成仕様をパターンカスタマイズ時の情報として利用することができる。

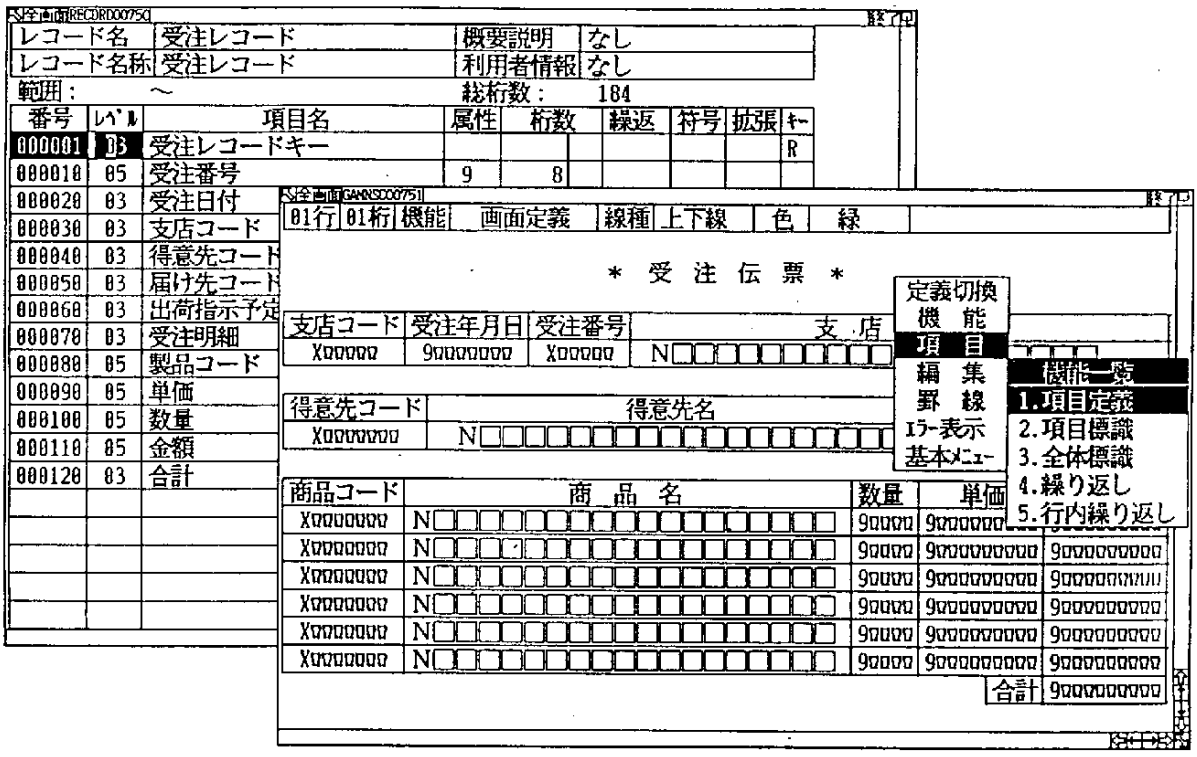


(1) ＩＤＬⅡによる生産効率

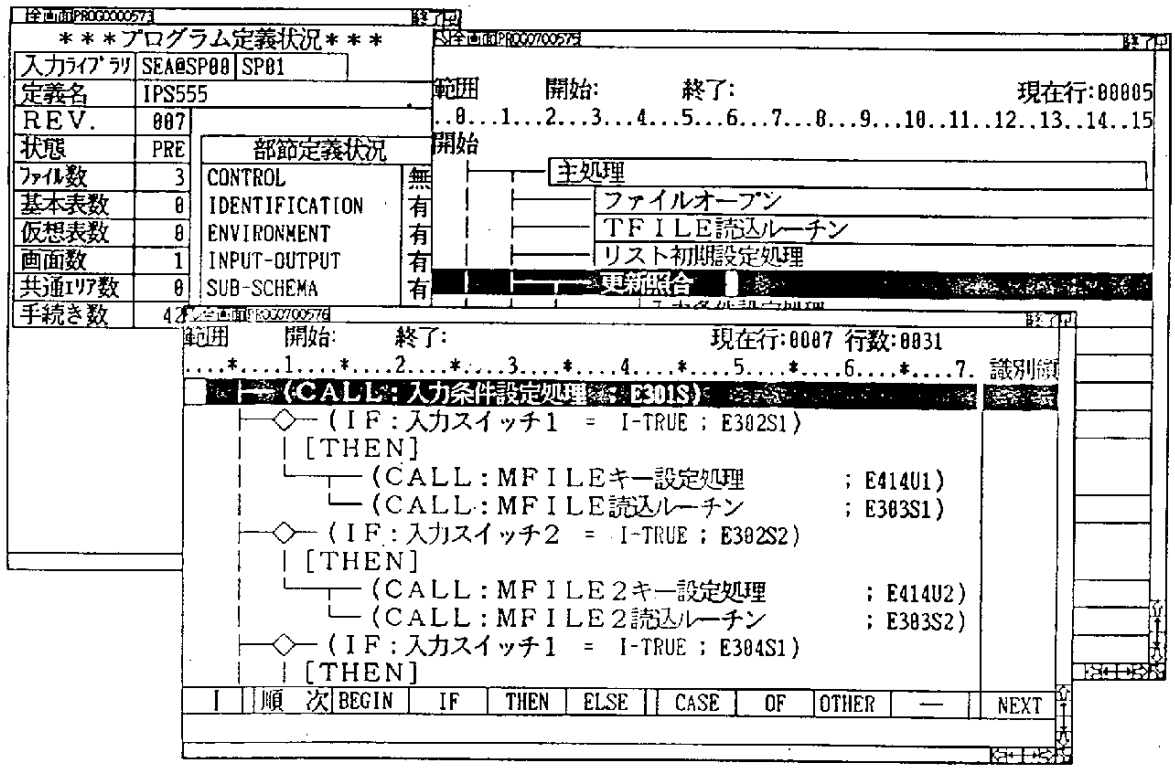


(2) ＩＤＬⅡによる処理仕様記述

図2.7-6 ＩＤＬⅡ



(1) ファイル定義、画面定義



(2) プログラム定義

図2.7-7 ＳＯＦＰＩＡ

2．7．2　効果

ＳＥＡ／Ｉは現在３、０００以上のユーザで使われている。ＳＥＡ／Ｉの導入による主な効果を整理すると次のようになる。

①生産性の向上

安定したパターン・部品の再利用により記述量を少なくすることができる。さらに、品質確保作業を削減することができ、生産性の向上がはかれる。特に、ＩＤＬⅡでは、ＣＯＢＯＬでコーディングした場合に比較して1/3以下のコーディング量ですみ、開発工数が大幅に削減できる。また、標準化の推進により個人差の少ないプログラム開発が可能となる。

②保守性の向上

ＳＥＡ／Ｉのプロダクト管理機能により、修正箇所の確認、影響範囲の検索が可能になりシステムの保守効率が向上する。プログラム生産物と同期のとれたドキュメントが自動的に得られることによる効果が大きい。また、標準的なパターンの利用により誰が見ても分かりやすいプログラムになり、比較的簡単に機能変更、機能追加が行えるようになる。

③管理面の向上

開発ソフトウェアの検収物件としてＳＥＡ／Ｉで生成される各種のドキュメントは有効である。

④要員スキルの向上

システム開発技法やプログラム作成能力を短期的に習得することができる。

2．7．3　適用上の注意

ＳＥＡ／Ｉを導入することによる生産性向上の効果をより顕著にするための留意点について説明する。

①標準化の推進

ＳＥＡ／Ｉによる生産性の向上の成果を顕著にするためには、開発チーム全体のソフトウェア環境の標準化が不可欠である。あるいは、ＳＥＡ／Ｉは、標準化を進めるための道具であると考えることもできる。標準化の項目としては、データ項目名のネーミング規則から端末の運用形態にいたるまで多岐に渡って存在する。標準化がどの程度まで進められるかは、それぞれの会社の事情に依存する事項ではあるが、標準化のノウハウの深化が重要であり、ＳＥとの協力関係が必要であることは言うまでもない。

②部品化

開発したものがそのまま簡単に部品として再利用できるというものではない。部品化を進めるためには、最初から部品として開発するものとしての十分な考慮をしておく必要がある。

③導入投資

新しくＳＥＡ／Ｉを導入するためには要員の教育や習熟するまでに必要な期間を見込んだ計画を立てておくことが必要である。

２．７．４　課題

ＳＥＡ／Ｉは、以上の説明したように上流工程（システム分析）から下流工程（製造・保守）にいたるソフトウェア開発の全工程を体系的に支援するツール体系を提供するものあるが、まだ不十分な機能もある。ＳＥＡ／Ｉをさらに強化する必要があると思われる点としては、次のような項目がある。

①データ分析からデータ設計へつなげるためのツールなどの、上流工程支援強化。

②分散環境下での開発資源の管理・保護機能の強化。開発時の各種の承認作業をどう手順化するかのような問題もある。

③ワープロ等の文書処理システムや、運用システムとの連携強化。

④自動化機能強化。上流工程、テストの自動化などはまだ工夫する余地がある。

⑤ＣＡＳＥのインタフェースの標準化などにより他のソフトウェア開発支援ツールとの連携。

2.8　ＰＲＩＤＥ

2.8.1　ＰＲＩＤＥの概要

情報システムの設計・開発・維持管理を支援する方法論パッケージであるＰＲＩＤＥファミリ・パッケージは、米国ＭＢＡ社が開発し、国内においては日本プライド社が販売・サポートしている。

このパッケージの原型であるＰＲＩＤＥ方法論が１９７１年に提供されて以来、方法論やツールが次々と開発され、現在のファミリ・パッケージを構成してきている。構成するパッケージを方法論とツールの順に次に示す。

・情報戦略の立案を支援する　ＥＥＭとＣＡＰ

・システムの設計・開発・維持を支援する　ＩＳＥＭとＡＳＥ

・データベースの設計・構築を支援する　ＤＢＥＭとＤＢＤ

・プロジェクト管理を支援する　ＰＭＳ

・上記パッケージの管理対象を部品として管理する　ＩＲＭ

これらの方法論とツールによってＰＲＩＤＥは、企業の中に情報工場を実現しようとしている。

2.8.2　情報工場とＰＲＩＤＥ

(1)　情報工場

企業におけるシステム部門の役割は、単にソフトウェアを開発して運用することではない。企業のビジネスをささえる情報システムを生産してユーザに提供し続けていくことにある。ちょうど自動車メーカーが車を生産してユーザに提供するといった役割に類似している。

即ち、システムの設計・開発・維持を行うということは、情報システムを生産する工場を運営することと同義になる。これを「情報工場」と呼んでいる。

情報工場を運営するには、工学アプローチをもった方法論と技法とツールが必要となる。これを図2.8－1に示すようにＰＲＩＤＥファミリ・パッケージが提供しているのである。

(2)　情報工場の部品表＝ＩＲＭ

工場の基幹データベースは、何といっても部品表である。この部品表に相当するものがＩＲＭであり、対象となる情報工場の部品を情報資源という。ＩＲＭには情報資源の管理機能があ

るため、情報資源管理システムと呼ばれている。

このシステムは、図2.8－2に示すように、設計者ならびに管理者に情報システムに関する仕様情報を提供することを狙っている。同時に、開発者に対しても種々の開発ツールの定義体を提供して開発作業を支援するものである。

(3) 情報資源と仕様情報

ＩＲＭの管理する情報資源は、図2.8－3に示すように、情報を活用する組織（誰が、何のために）、情報要求（いつ、どんなデータを使って何をみたいのか）、情報を提供するために必要なデータ（情報アウトプット、データを収集するインプット、データを蓄積するファイル、データのハンドリングをするレコード、データ項目、コール形式）データを処理するシステム（システム、サブシステム、プロセジャ・プログラム、モジュール）、情報システムを開発・維持するプロジェクト（プロジェクト、要員、スキル、修正改善要望）の20種類がある。

これらの情報資源は、全て部品として管理され相互に関係づけられている。この相互関係を利用することによって、さまざまな仕様情報を提供することができる。その例を図2.8－4に示す。

(4) 情報ＣＩＭへの挑戦

製造工場においてＣＩＭ (Computer Integrated Manufacturing)が指向されているが、情報工場においても例外ではない。このためＰＲＩＤＥでは、図2.8－5に示すように多くの企業で使われてる実践的ツールとの統合化による情報ＣＩＭに果敢に挑戦をいどんでいる。このケースでは、ドキュメントツールとして富士ＸＥＲＯＸ社のＪ－ＳＴＡＲ、開発ツールとしてパンソフィック社のＴＥＬＯＮをつなぎ、設計から開発・メンテなどの一貫したプラットフォームを実現しようとするものである。

また、米国ＭＢＡ社では、図2.8－1に示したＰＲＩＤＥファミリ・パッケージのツール群を、ＩＢＭ社のＰＳ２／ＯＳ２に移植しようとしており、ワークステーションを活用した分散開発環境の実現を狙っている。

このように，ＰＲＩＤＥファミリ・パッケージは、ＣＡＳＥ(Computer-Aided Softwar Engineering)の枠を越えて情報工場の実現を狙ったエンジニアリング・ツールなのである。

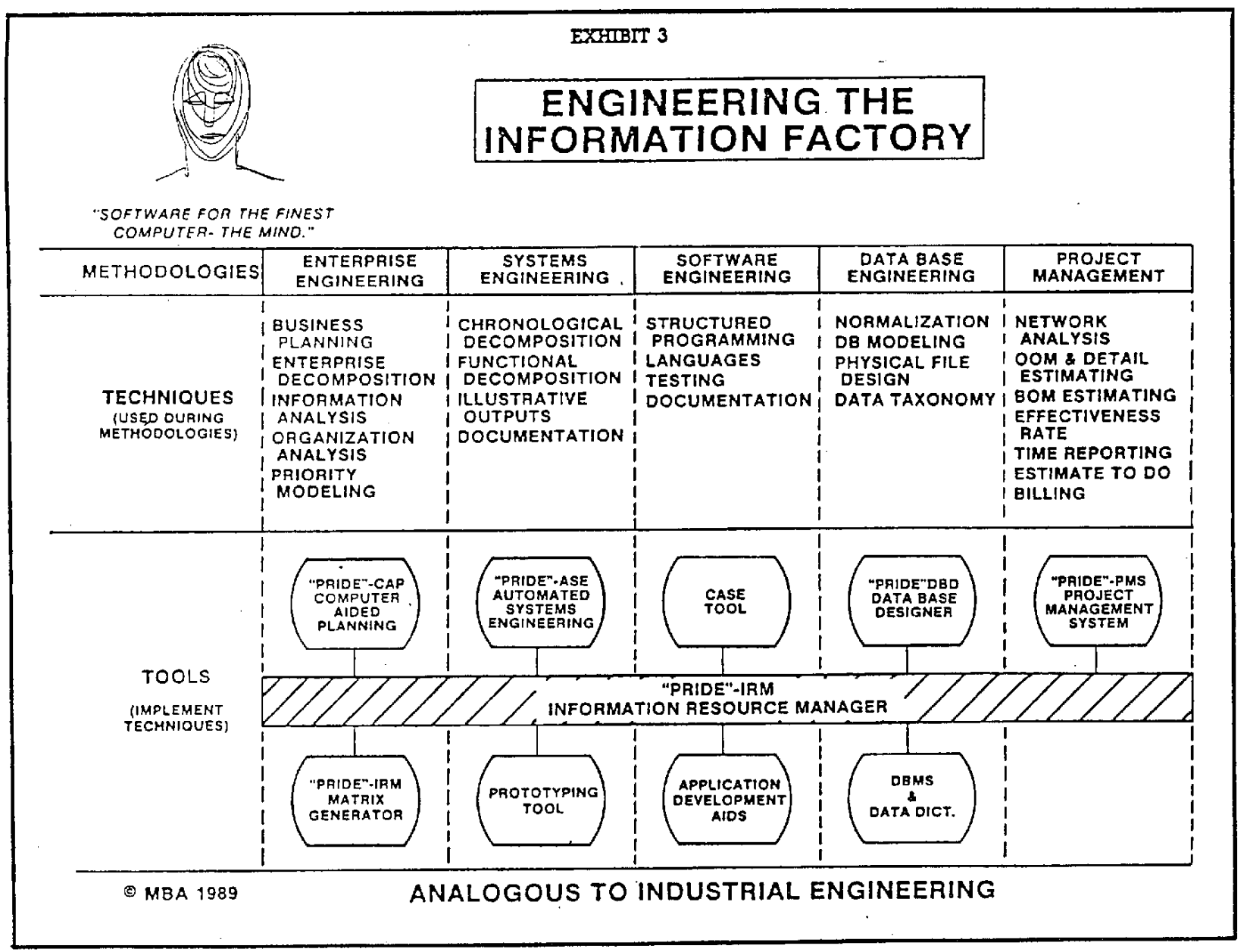


図2.8－1　情報工場の実現を支援するＰＲＩＤＥファミリ・パッケージ

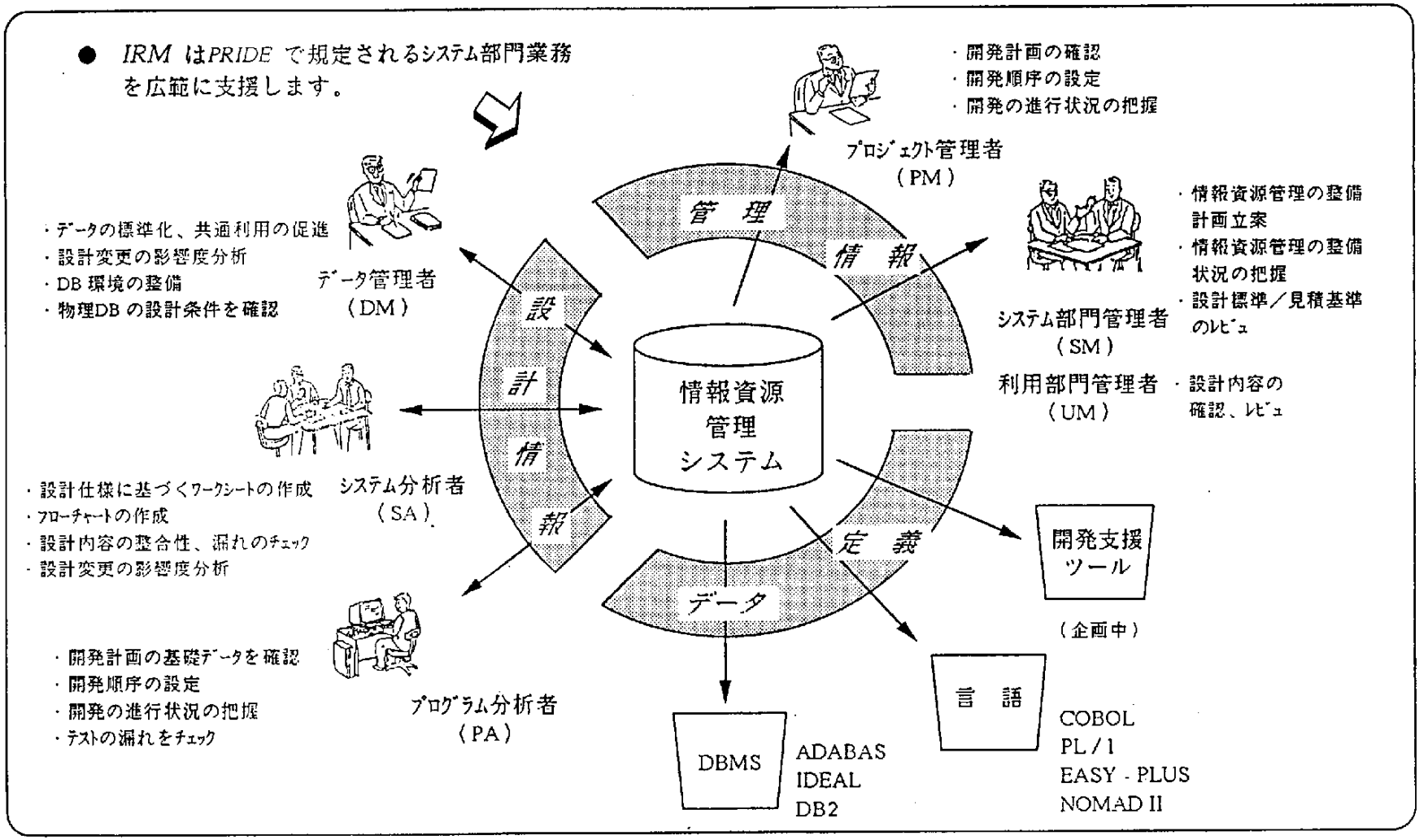


図2.8－2　情報資源管理の狙い

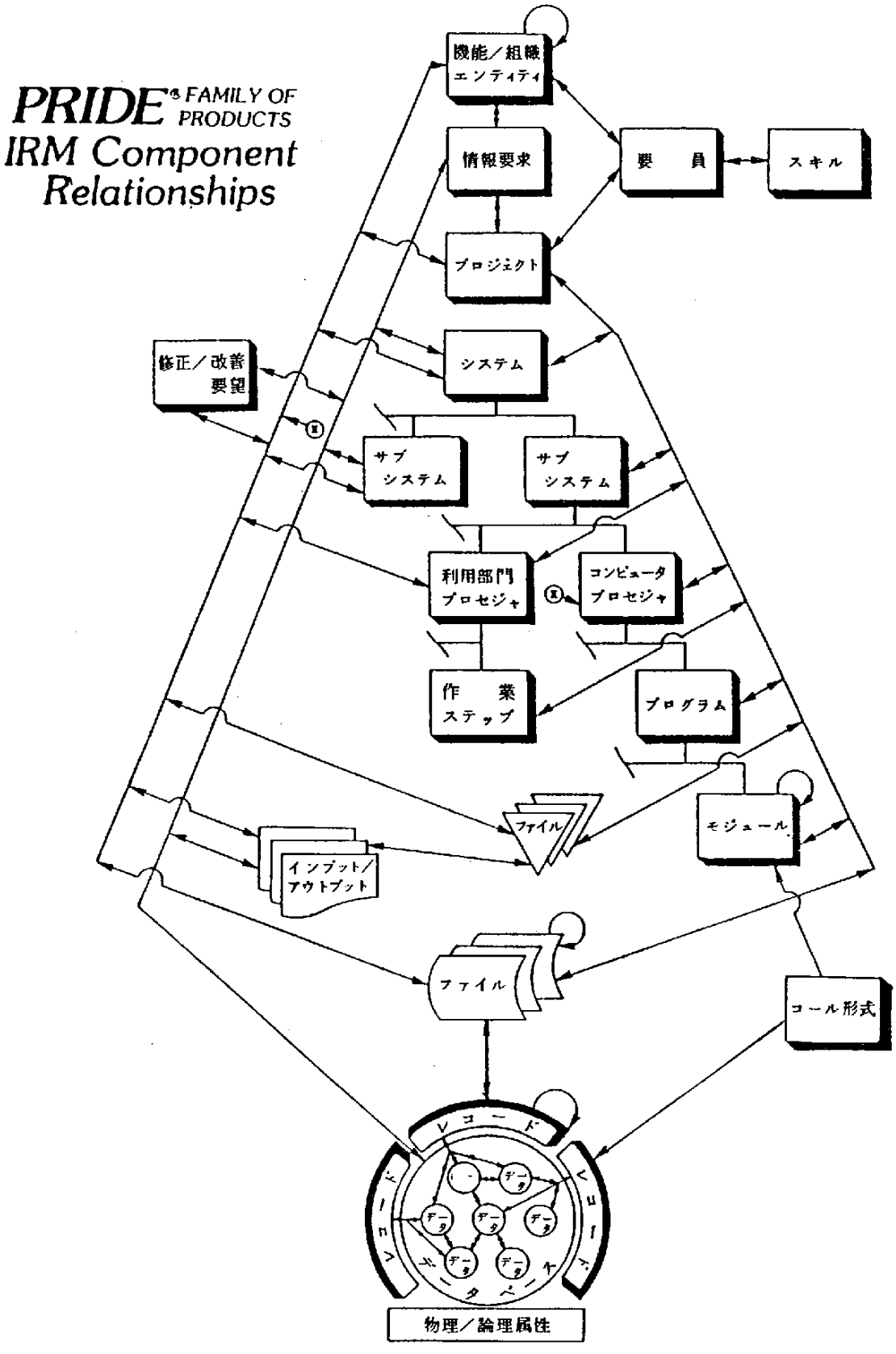
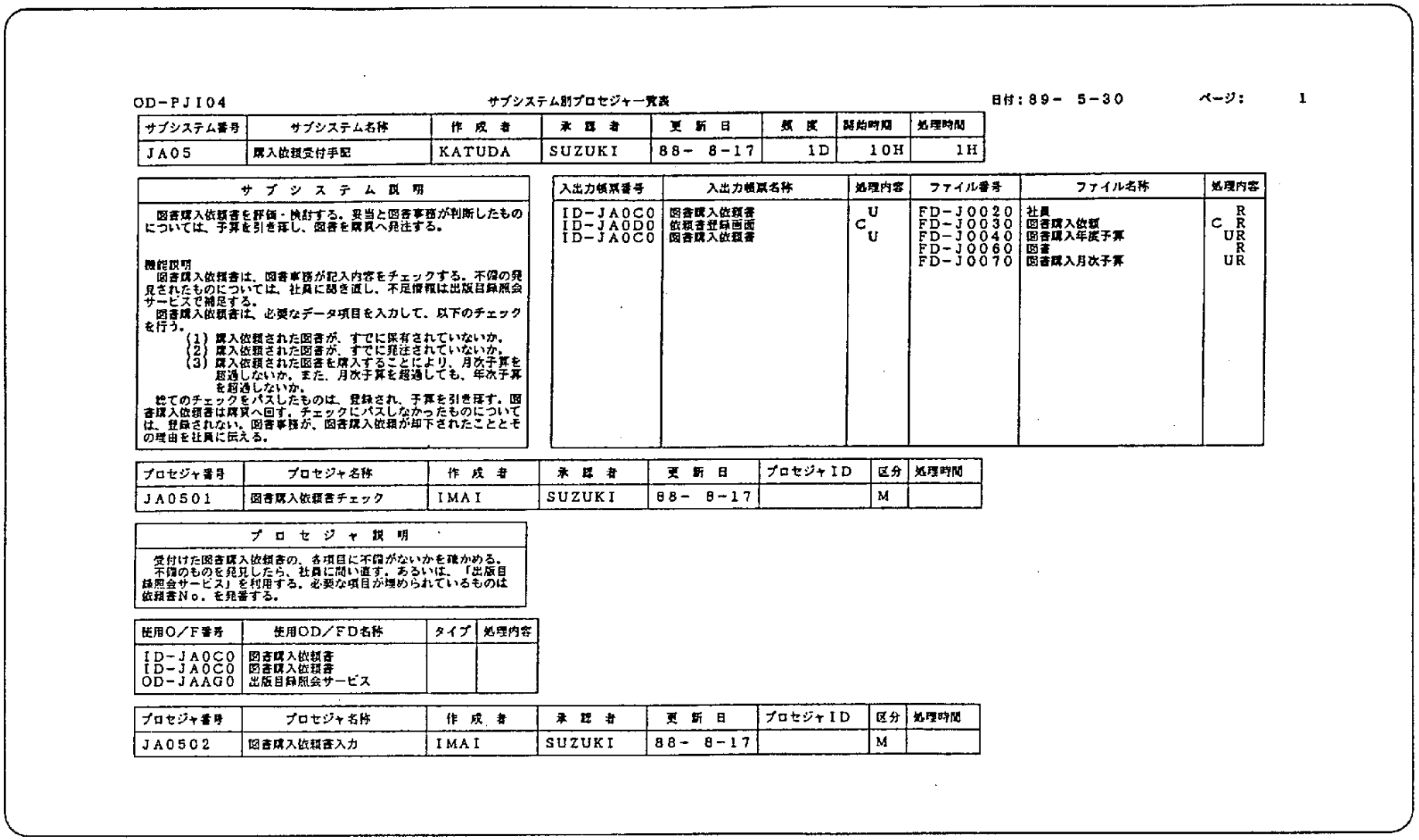

PRIDE® FAMILY OF PRODUCTS
IRM Component Relationships
機能／組織エンティティ
情報要求
要　員
スキル
プロジェクト
修正／改善要望
システム
サブシステム
サブシステム
利用部門プロセジャ
コンピュータプロセジャ
作業ステップ
プログラム
ファイル
モジュール
インプット／アウトプット
ファイル
コール形式
Vコード
Vユニット
データ
物理／論理属性

図2.8－3　情報資源



OD-PJI04　　サブシステム別プロセジャ一覧表　　日付：89- 5-30　　ページ： 1

サブシステム番号	サブシステム名称	作成者	承認者	更新日	頻度	開始時期	処理時間
JA05	購入依頼受付手配	KATUDA	SUZUKI	88- 8-17	1D	10H	1H

サブシステム説明

図書購入依頼書を評価・検討する。妥当と図書事務が判断したものについては、予算を引き落し、図書を購買へ発注する。

機能説明
図書購入依頼書は、図書事務が記入内容をチェックする。不備の発見されたものについては、社員に聞き直し、不足情報は出版目録照会サービスで補足する。
図書購入依頼書は、必要なデータ項目を入力して、以下のチェックを行う。
(1) 購入依頼された図書が、すでに保有されていないか。
(2) 購入依頼された図書が、すでに発注されていないか。
(3) 購入依頼された図書を購入することにより、月次予算を超過しないか。また、月次予算を超過しても、年次予算を超過しないか。
総てのチェックをパスしたものは、登録され、予算を引き落す。図書購入依頼書は購買へ回す。チェックにパスしなかったものについては、登録されない。図書事務が、図書購入依頼が却下されたこととその理由を社員に伝える。

入出力帳票番号	入出力帳票名称	処理内容	ファイル番号	ファイル名称	処理内容
ID-JA0C0	図書購入依頼書	U	FD-J0020	社員	R
ID-JA0D0	依頼書登録画面	C	FD-J0030	図書購入依頼	C R
ID-JA0C0	図書購入依頼書	U	FD-J0040	図書購入年度予算	UR
			FD-J0060	図書	R
			FD-J0070	図書購入月次予算	UR

プロセジャ番号	プロセジャ名称	作成者	承認者	更新日	プロセジャID	区分	処理時間
JA0501	図書購入依頼書チェック	IMAI	SUZUKI	88- 8-17		M	

プロセジャ説明

受付けた図書購入依頼書の、各項目に不備がないかを確かめる。
不備のものを発見したら、社員に問い直す。あるいは、「出版目録照会サービス」を利用する。必要な項目が埋められているものは依頼書No.を発番する。

使用O/F番号	使用OD/FD名称	タイプ	処理内容
ID-JA0C0	図書購入依頼書		
ID-JA0C0	図書購入依頼書		
OD-JAAG0	出版目録照会サービス		

プロセジャ番号	プロセジャ名称	作成者	承認者	更新日	プロセジャID	区分	処理時間
JA0502	図書購入依頼書入力	IMAI	SUZUKI	88- 8-17		M	

図2.8－4(1)　ＩＲＭの提供する仕様情報の例（設計レポート）

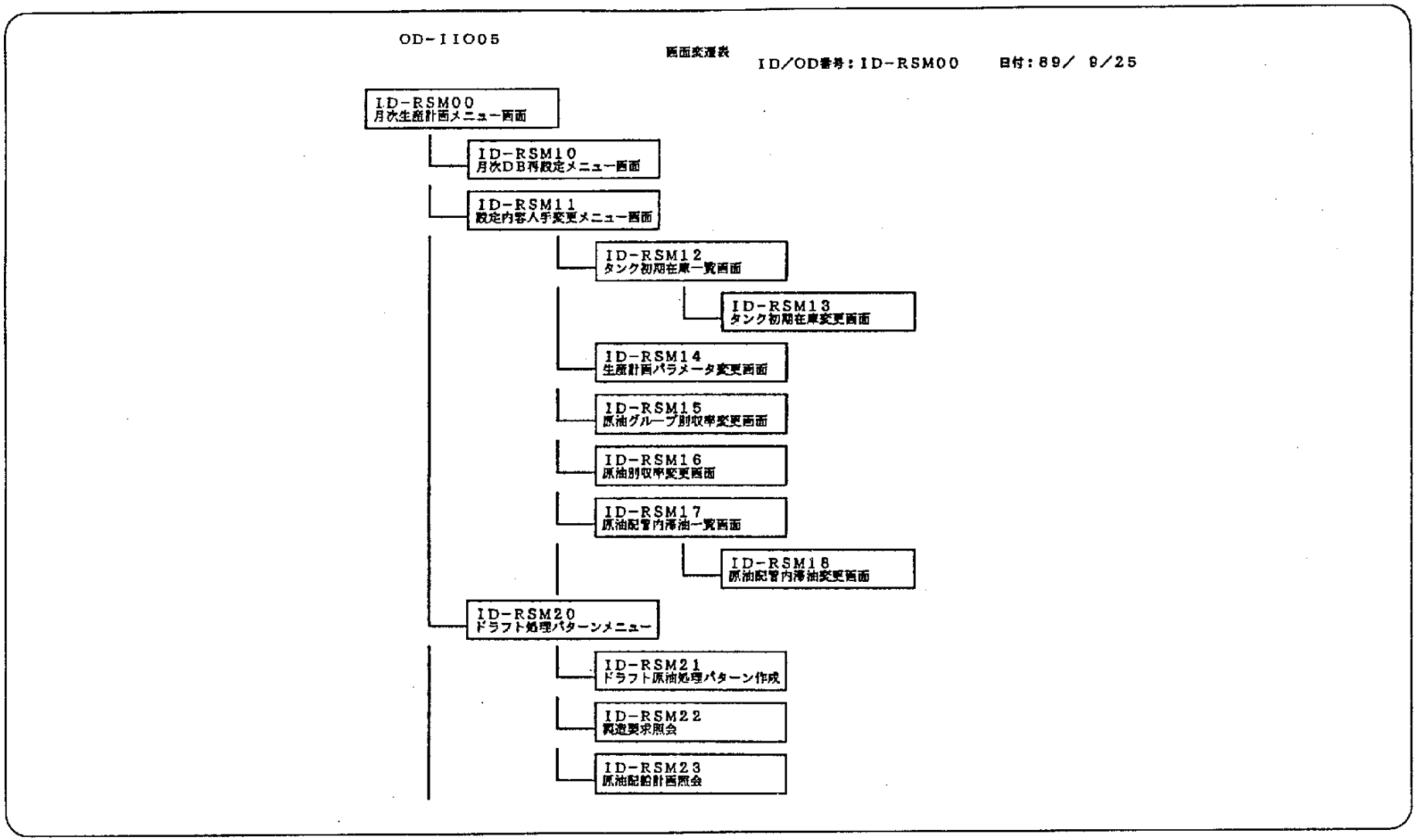


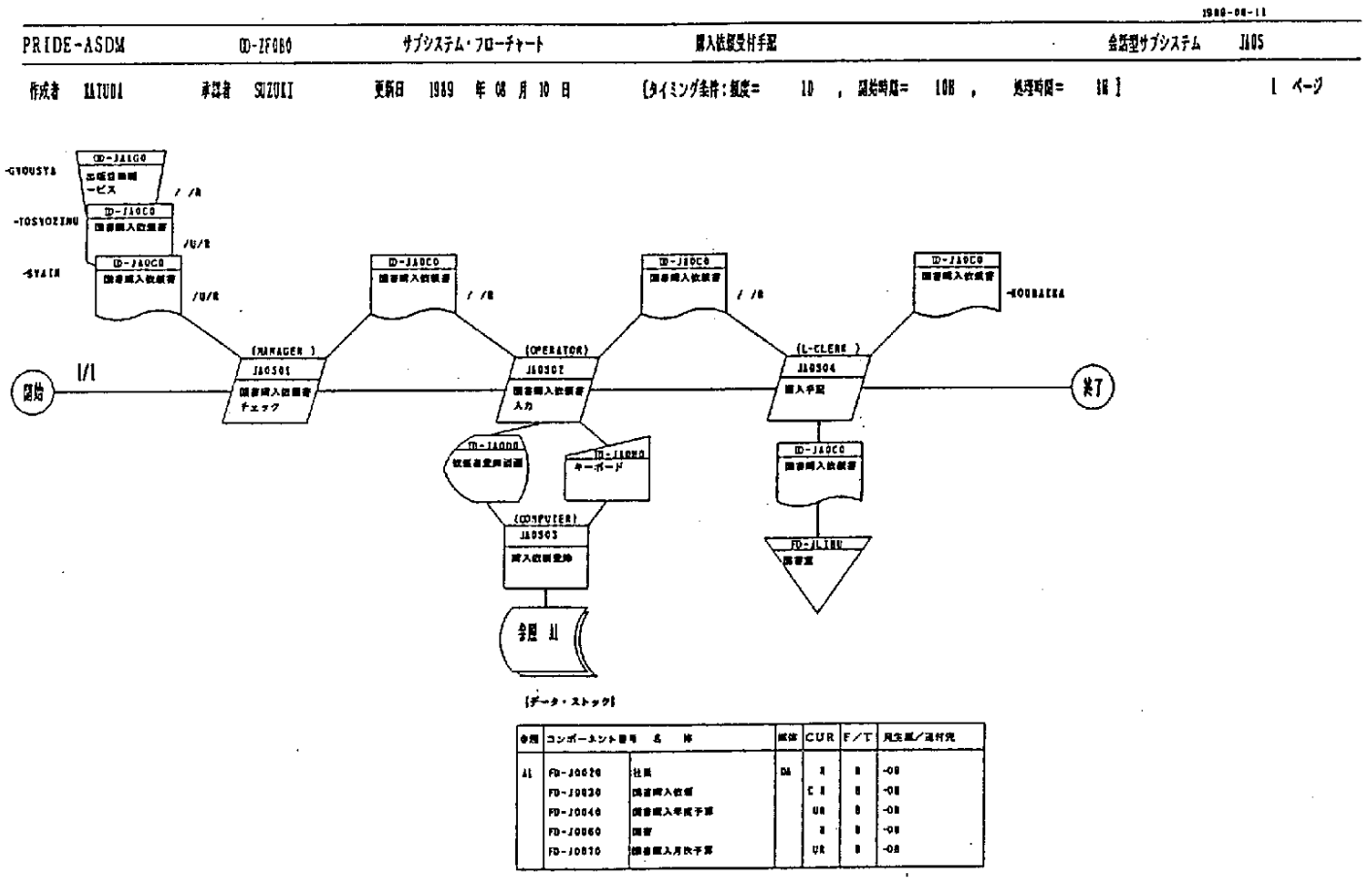
図2.8－4(2)　ＩＲＭの提供する仕様情報の例（階層図）

PRIDE-ASDM ZDJU

OD-ZDBI0　　サブシステム関連テーブル　　JA

コンポーネント番号	コンポーネント名称	JA01 年次図書利用購入管理	JA02 年次図書購入計画作成	JA03 図書購入予算修正	JA04 計画購入図書選定	JA05 購入依頼受付手配	JA06 図書検収	JA07 月次図書購入管理	JA08 図書登録
FD-J0020	社員	/　/R				C/　/R	/　/R		/U/
FD-J0030	図書購入依頼		/　/R	/　/R	/　/R	C/　/R	/U/R	/　/R	
FD-J0040	図書購入年度予算					/U/R	/U/	/　/R	C/　/R
FD-J0050	図書分類	/　/R							
FD-J0060	図書	/　/R				/　/R			C/　/
FD-J0070	図書購入月次予算		C/　/	/U/		/U/R	/U/	/　/R	
ID-JA0A0	月次予算登録画面		/U/						
ID-JA0B0	図書購入予算修正画面			/U/					
ID-JA0C0	図書購入依頼書				C/　/	/U/			
ID-JA0D0	依頼書登録画面					C/　/			
ID-JA0E0	納品書						/U/		
ID-JA0F0	検収登録画面						/U/		
ID-JA0G0	図書登録画面								/U/
OD-JAAA0	図書利用実績リスト		C/　/R						
OD-JAAB0	購入依頼月別実績リスト	C/　/	C/　/R						
OD-JAAC0	未納入図書一覧表				C/　/				
OD-JAAD0	検収済み図書一覧表						C/　/		/　/R
OD-JAAE0	図書購入管理表							C/　/	
OD-JAAF0	図書分類体系表								/　/R

図2.8－4(3)　ＩＲＭの提供する仕様情報の例（マトリックス表）



〔データ・ストック〕

参照	コンポーネント番号	名　称	媒体	CUR	F／T	発生源／通行先
JI	FD-J0020	社員	DA	R	D	-OD
	FD-J0030	図書購入依頼		C R	D	-OD
	FD-J0040	図書購入年度予算		UR	D	-OD
	FD-J0050	図書		R	D	-OD
	FD-J0070	図書購入月次予算		UR	D	-OD

図2.8－4(4)　ＩＲＭの提供する仕様情報の例（フローチャート）

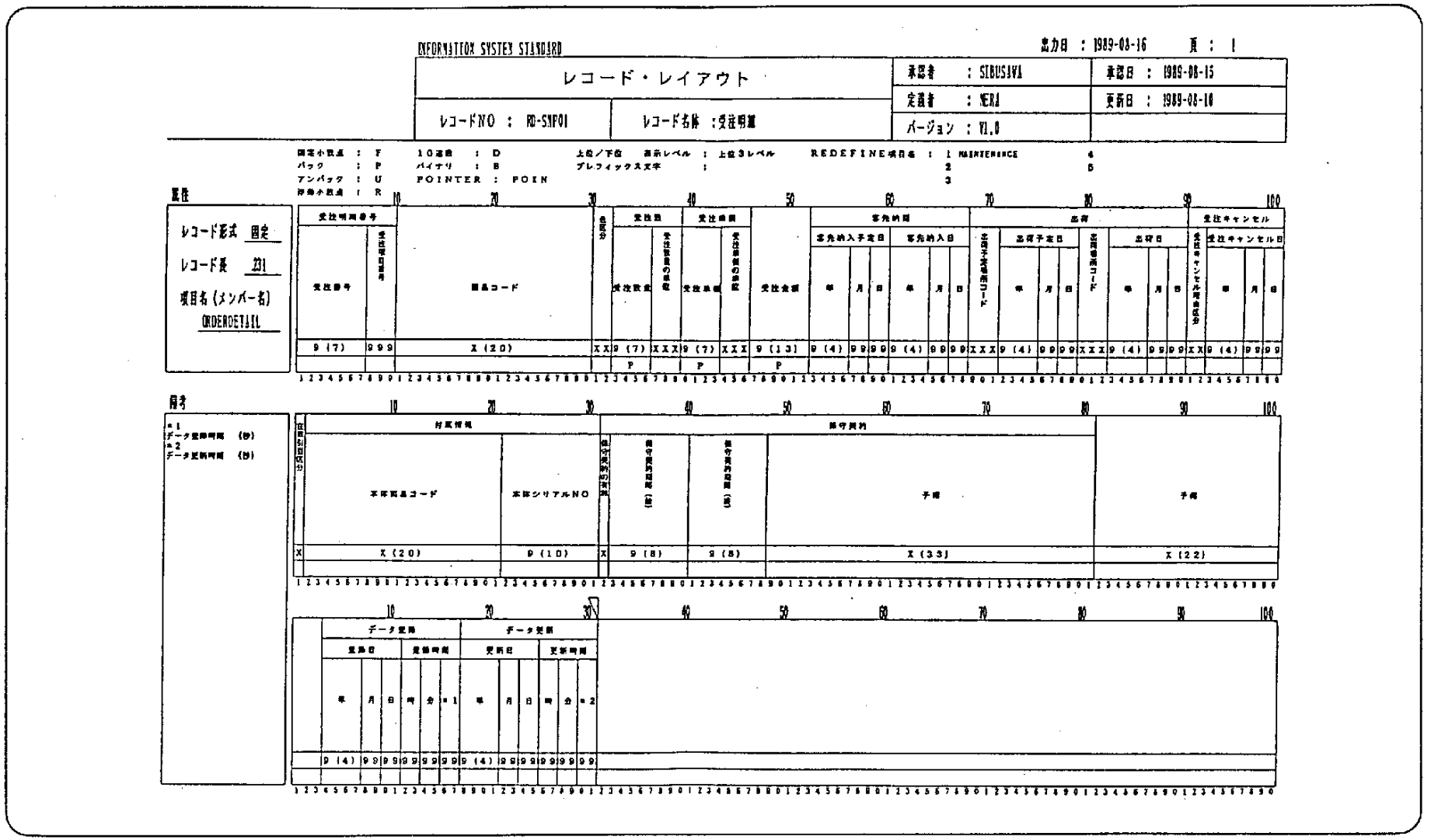


図2.8－4(5)　ＩＲＭの提供する仕様情報の例（レコードレイアウト）

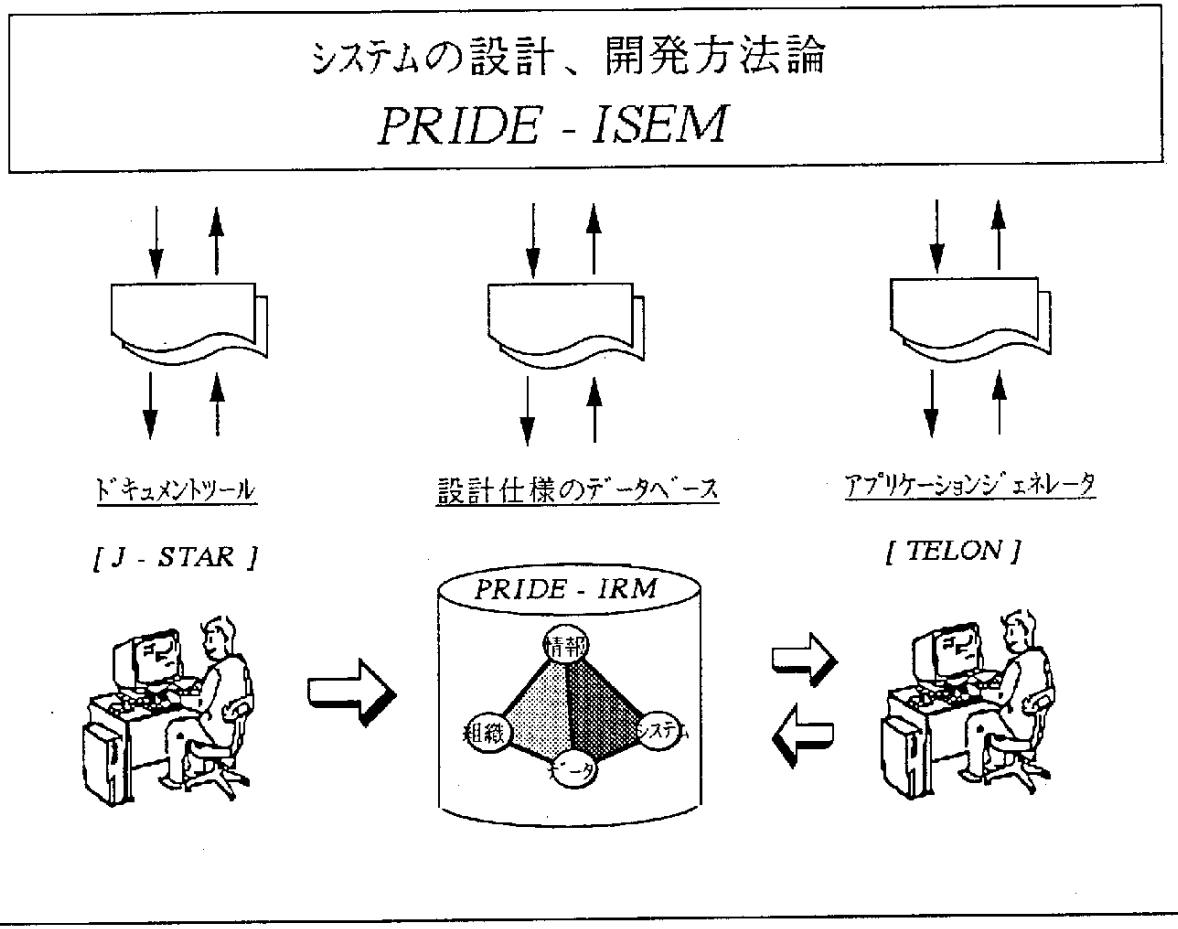


図2.8－5　実践的ツールとの統合化による情報ＣＩＭへの挑戦

２．９　ＸＰＴ－Ⅱによる開発環境の構築

２．９．１　プロセス・プログラミングとソフトウェア開発

（１）プロセス・プログラミングとは

ソフトウェア・プロセスに関する研究は第９回ソフトウェアプロセス工学国際会議(9th, ICSE)でコロラド大学のL.Osterweilの基調講演”Software Prosesses are Software too.”によっている。

ここでいう、プロセスとは以下の様なものである。

・プロセスとは

ある仕事や作業を行うために、人間が行う知的作業

・ソフトウェア・プロセスとは

ソフトウェア開発に伴う人間の様々な知的活動

即ち、ソフトウェア開発手順そのものである。

一般に様々な方法論があるが、方法論を一言で言うと以下の様に考えられるのではないか。方法論とは、物をステップ・バイ・ステップに作るために

・そのステップを工程として定義

・その工程の成果物を定義

・その成果物を作成の仕方を決める

・また、その成果物の評価法もあわせて定める

これらを、集大成したものが方法論であるといっても過言ではないだろうか。その、方法論の

うえにその成果物を作成する際に使用する書式、図式等が技法に該当するのではないか。また、それらを効率的に作成するための工具が、一般に言われているＣＡＳＥツールではないか。ここでは、ＣＡＳＥの定義をもっと広くとる。

ＣＡＳＥとは

人間がソフトウェアを作成する際の知的活動を支援するもの。

従って、上記の狭い意味でのＣＡＳＥも含まれる。また、知的活動そのものを自動化・無人化することも含まれる。即ち、ソフトウェア・プロセスを対象として、それを含んだ開発環境を作り上げることを意味する。

（２）ツールの統合の仕方

既存のツールを統合しようとする試みがあるが、それには以下の３通りの方法がある。

１）弱い統合

ツール間のインターフェイスやユーザインタフェイスを統一することでツールを統合するもの

２）強い結合

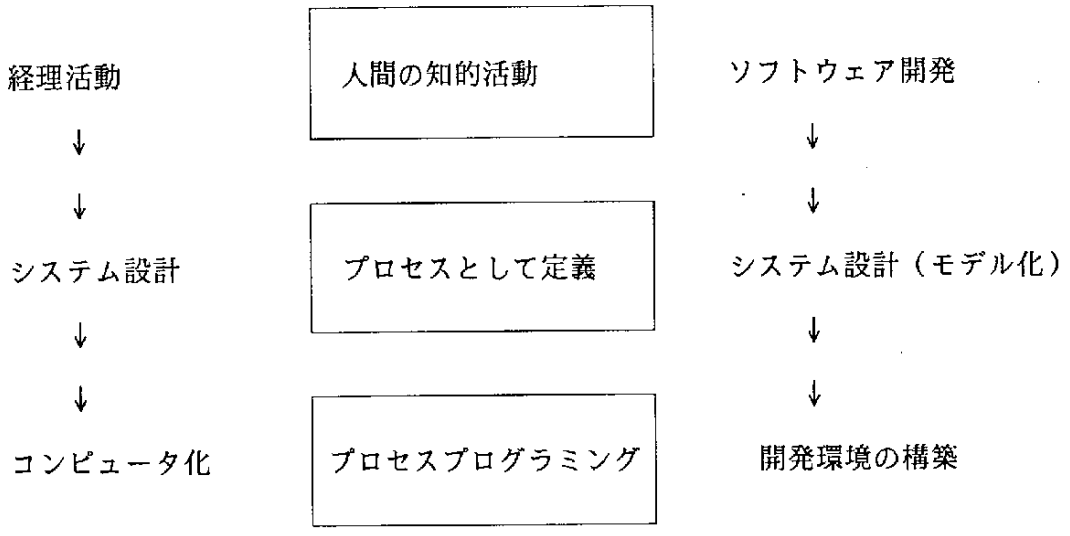
あるメソッドをベースにおいて、それに適合するツールを統合する

３）モデルを基礎とした統合

開発過程をモデル化して、その過程にあわせたツールを統合する

弱い結合は、統合する人が何等かの方法論を想定して結合することになると考えられる。そうでないと、世の中の全てのツールを統合することになってしまう。強い結合は旧来からある方

法論を基盤としてそれ用のツールを作成していくことである。よって、新しい方法論が提唱され、それに変更する場合は無効である。三番目のモデルを基礎とした結合は、我々が今まで行ってきたソフトウェア開発そのものである。すなわち、経理システムを作成する場合、経理部門を担当している経理部員にヒアリングをして対象システムをモデル化する（事前調査、基本設計）、そのモデルをもとに人間の活動をプログラミングしていく（詳細設計、プログラミング）というものとおなじである。



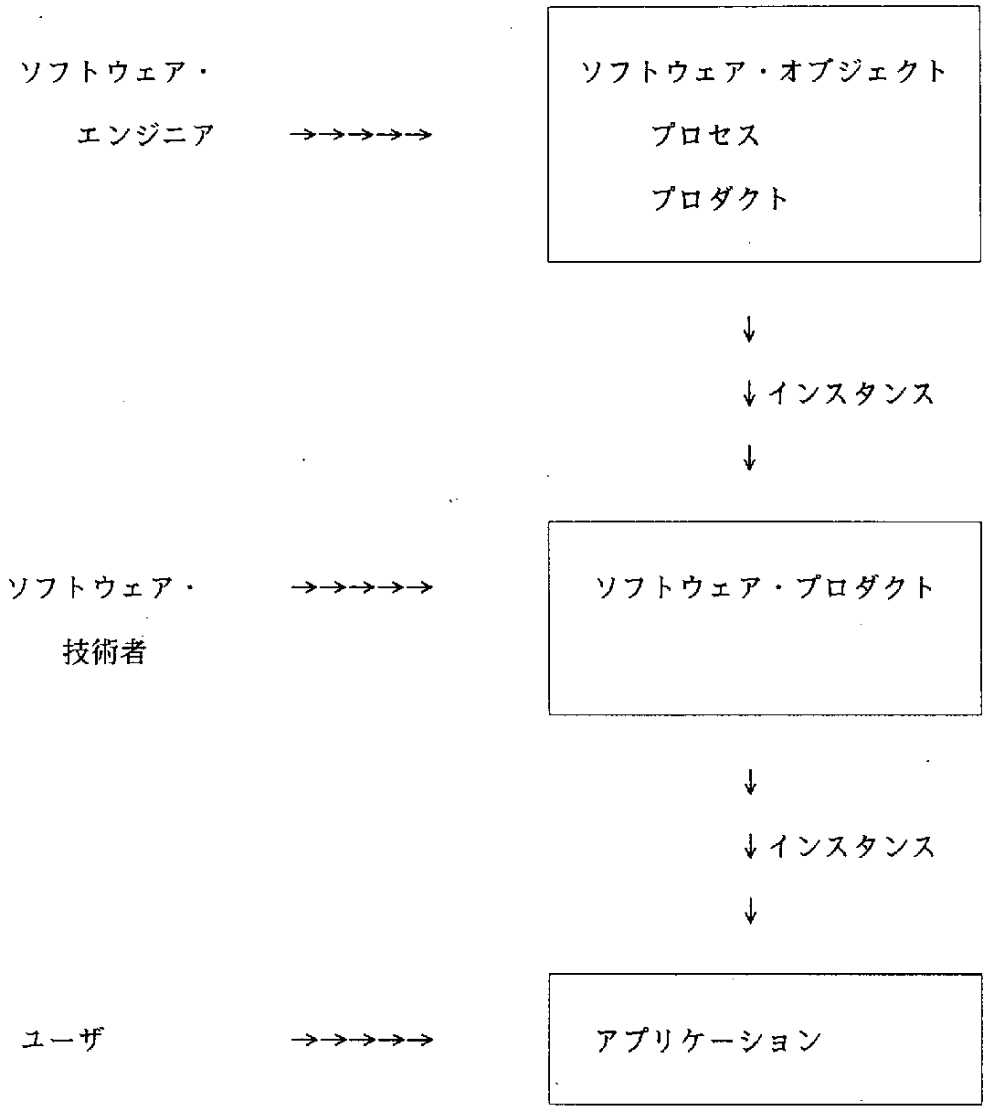
（３）プロセス・プログラミングにおけるソフトウェア開発

プロセス・プログラミングが可能になれば、ソフトウェア開発は次のようになる。

１）ソフトウェア・エンジニア（ＳＥの神様的存在）が、プロセス・オブジェクトを作成する。プロセス・オブジェクトにはプロセスを記述するものと、プロダクトを記述するものがある。

２）ソフトウェア技術者はプロセス・アルゴリズムを実行しながら、仕様、コード等を埋め込んでいきながら、ソフトウェア・プロダクトを生成する。即ち、プロセス・オブジェクトのインスタンスを生成する。

3）また、ユーザはそのソフトウェア・プロダクトをカスタマイズして自分の仕事を成し遂げていく。すなわち、ソフトウェア・プロダクトからインスタンスを生成していく。



2．9．2　ＸＰＴ－Ⅱによるプロセス・プログラミングの実現

ＣＳＫ総合研究所で開発されたエキスパートシステム開発支援ツールＸＰＴ－Ⅱを用いてプロセス・プログラミングのパラダイムを実現することを考えてみた。ＸＰＴ－Ⅱの簡単な紹介をしておくと次のようになる。

（１）エキスパートシステムＸＰＴ－Ⅱ

稼働マシン

ＩＢＭ汎用機（ＭＶＳ、ＶＭ）

ＯＳ／２，ＵＮＩＸが稼働するパソコン、ワークステション

知識表現

オブジェクト指向をベースにしたフレーム表現

プロダクション・ルール表現

ｐｒｏｌｏｇ述語表現

推論機構

黒板モデル

マンマシン・インターフェス

汎用機上でのマルチウィンドウ、マウス等

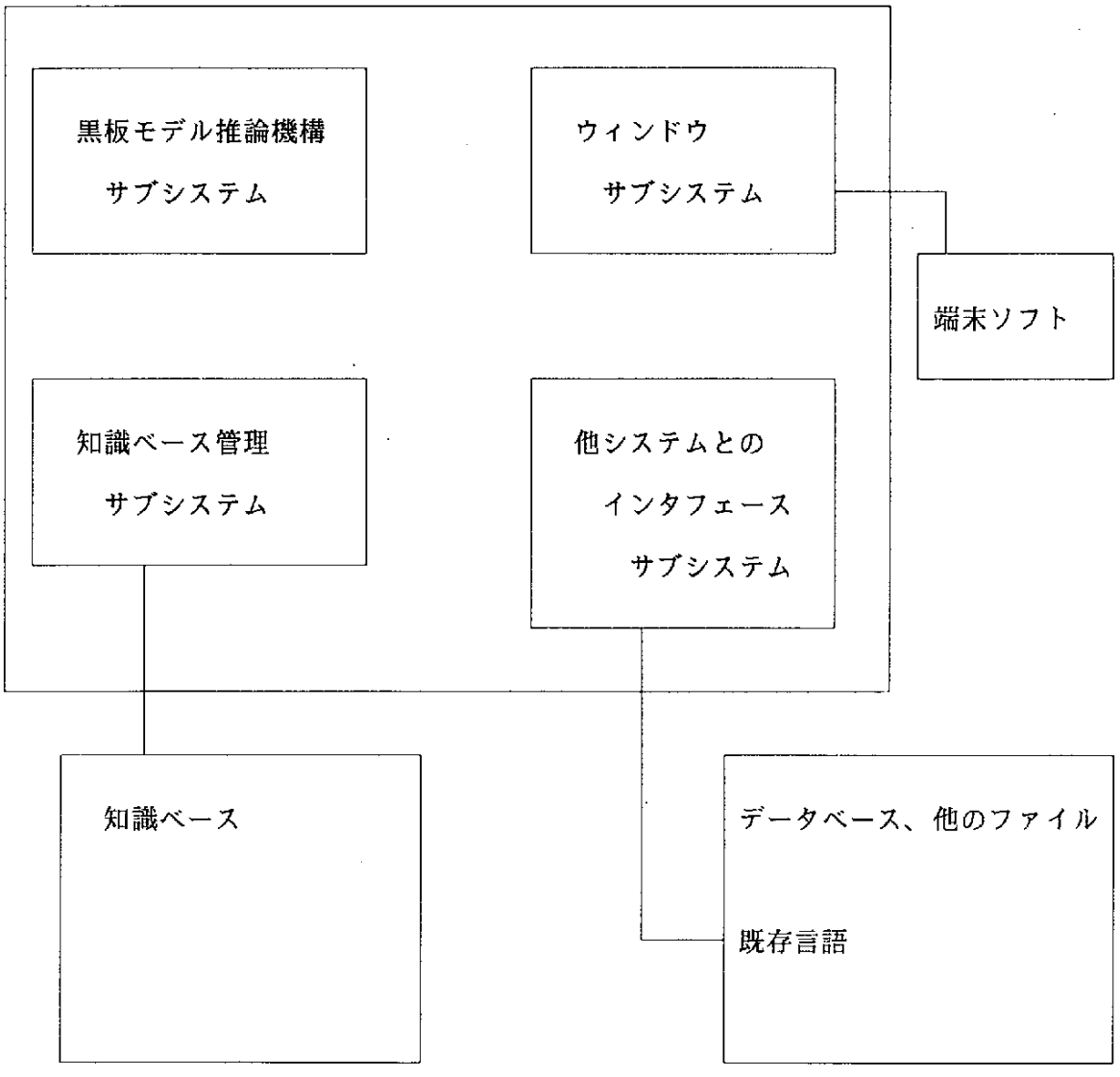


図2.9-1　　ＸＰＴ－Ⅱの汎用機での構成

（２）システムの構成

①プロセス管理

プロセスをオブジェクトとして登録する。情報としては、以下のものを登録した。

プロセス名

開始条件（そのプロセスが開始してよい条件、開始前に完成されていなければならない成果物）

終了条件（そのプロセスが完了したときの継続プロセスと作成された成果物）

見積ルール（所要日数）

②プロダクト管理

作成されるドキュメントを登録する。

ドキュメント名称

作成手順（参照するドキュメント、書式、ツール）

プロセスとの関連をつける

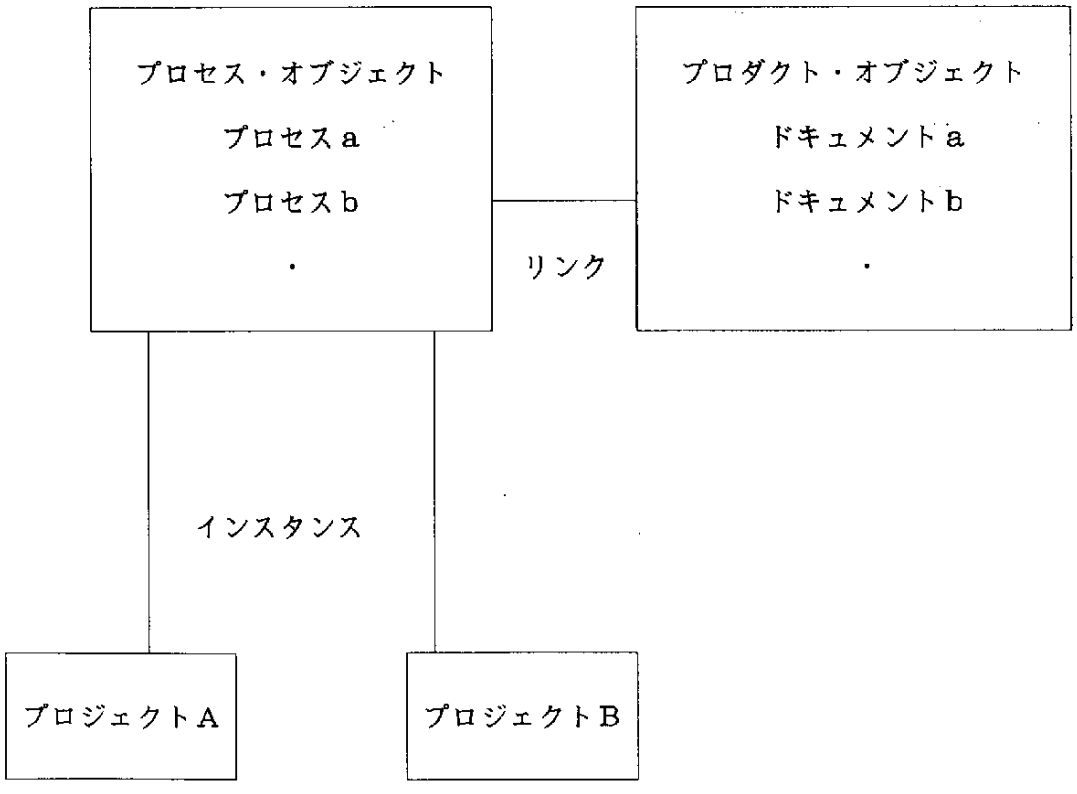


図2.9-2　　システムの概念図

（３）問題点

・プロセス自体の記述ができていない

あるメーカーの方法論をインプリメントしたが、結果的にはドキュメント間の相互関連、書式の提示にしかすぎなかった。インプリメントにも問題は有ると考えるが、現状の方法論を鑑みるにドキュメントの書式は示されるが、それをどの様に作成するかは記述されていない。即ち、プロセス自体がまだ記述されていないということである。

・作成用のツールとのインターフェイスがとれていない

これは、単にインプリメントできなかったということで技術的には時間と工数の問題である。

・ユーザインタフェイスがまだまだである。

いくつかのプロセスをつないでプロジェクトの工程を作成するためにＰＥＲＴのネットワーク図を用いた。この際、ビジュアルな操作で出来るようにしたが、画面の大きさの制限で思うような操作性を得ることができなかった。

（４）効果

・結果的には、プロジェクト管理用のツールしか出来なかった。

計画作成の為の支援ツール（過去の類似プロジェクトを持ってきて新規プロジェクトの計画立案に利用）

・初心者には開発の手順書として使える。

初心者には教育なしでも（言い過ぎか）開発計画が立てられる。

２．９．３　プロセス・プログラミングの課題

プロセス・プログラムは「簡単に書けるがしかし意味のないものか、ほとんど書けないものか、の２極に分類されるだろう」という予想がある。現在の方法論を考えてみると、肝心な部

分は、当り前かも知れないがほとんど人間の直感に頼っている。この直感によっている部分が記述できるかが、問題である。今後の研究課題としては次の様に考える。

・プロセスを記述する言語の開発

手順として記述するのではなく、宣言的に記述されるべきであろう。

・人間の協調作業をも記述できること

開発の段階では人間の協調作業（時間的、空間的に隔たっている）があるが、これらをどのようにインプリメントするか

・プロセスをとにかく書き下す努力

自然言語でもよいから活動のすべてを書き下す努力がいる

・方法論と、それをサポートするツールの定義

方法論の洗い出しと、既存ツールとの関連を明確にする。

・ツールを構成要素（ＦＲＡＧＭＥＮＴ）に分解して、それを組み立てる仕組みと管理する仕組みの構築

２．９．４　ＣＡＳＥツールに要求される事項

仮に、ＣＡＳＥツールの要求仕様書をまとめるとしたら、次の様になると思われる。

・開発・保守の全工程をサポートすること。即ち、ソフトウェアのライフサイクル全体をサポートする。

縦＝垂直の統合　　情報の連続性

・分散開発環境であること。

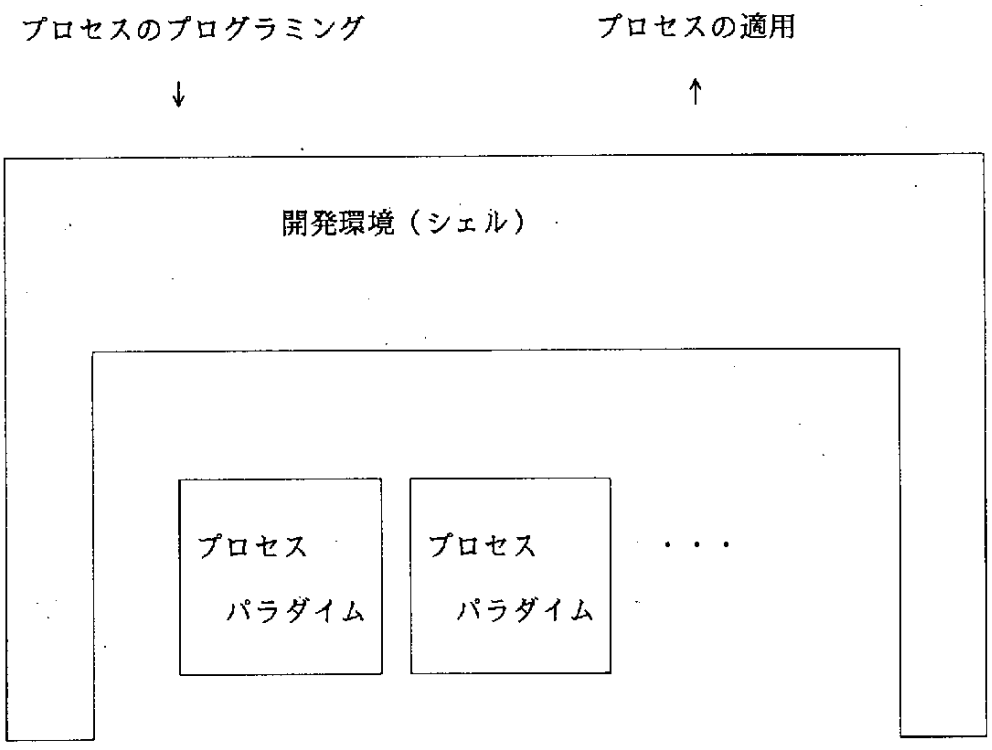
横＝水平の統合　　ＭＭＬの実現

・一つの方法論に依存しないこと。

方法論が取捨選択できる　　方法論がprogram 可能

・ＯＰＥＮ　ＡＲＣＨＩＴＥＣＴＵＲＥであること。

これらの要求を満たすためには下記の様な開発環境を作成するシェルが必要である。



今までの記述から明らかなように、プロセスを記述出来たとすると、それをインプリメントするのにエキスパートシステムは有効である。その際の、システム構成は次の様になると考える。

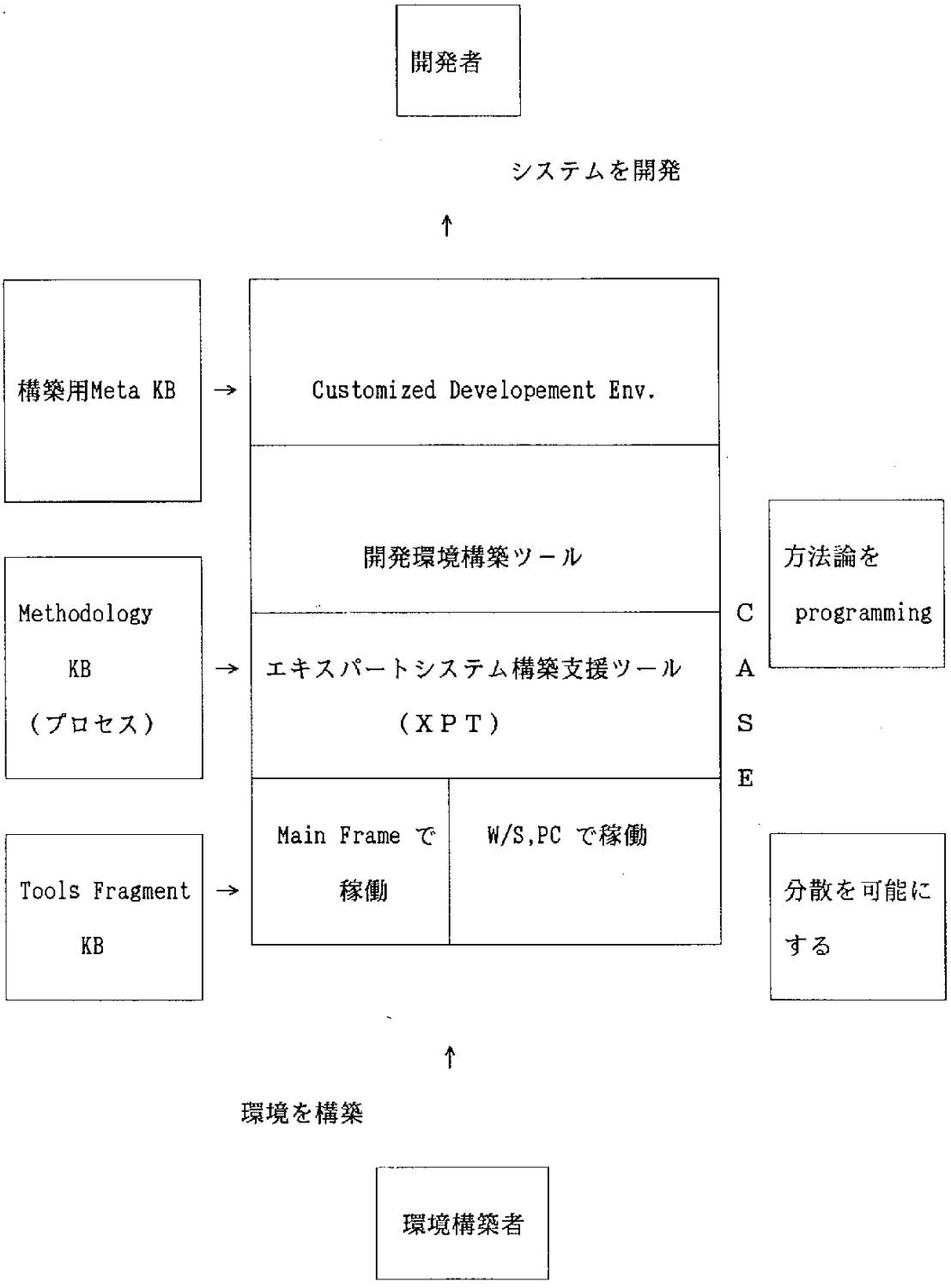


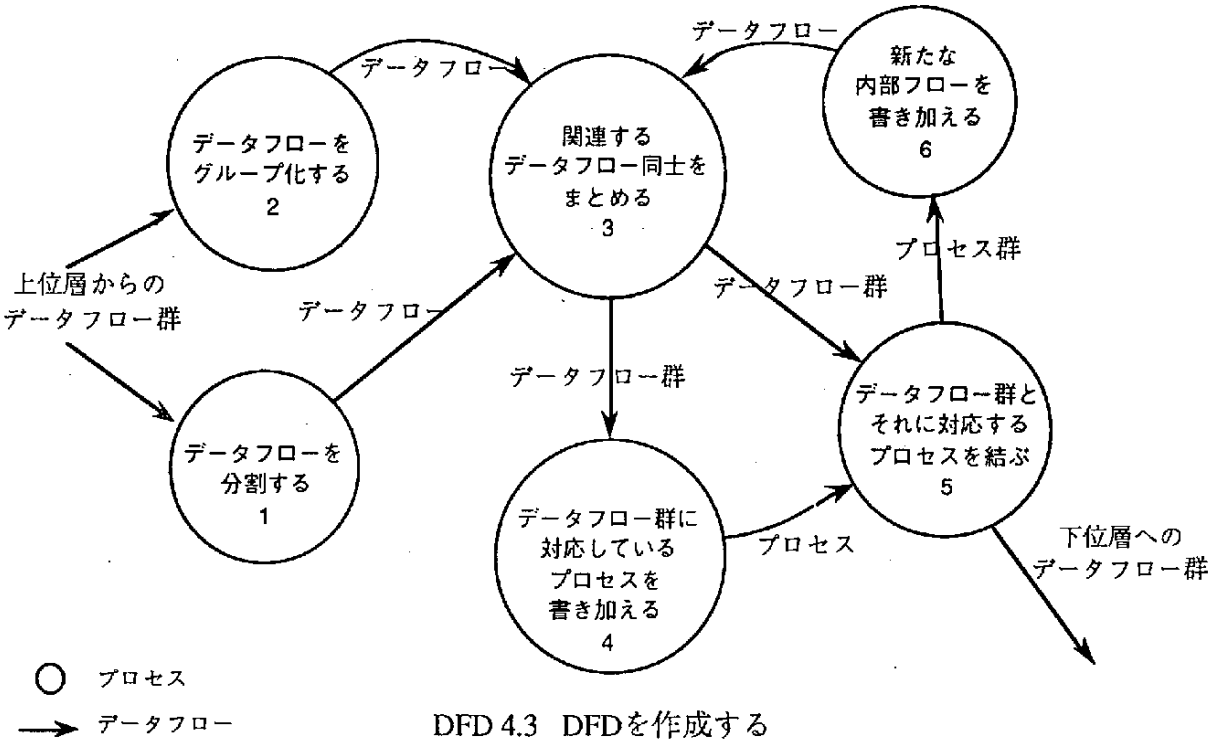
図2.9-3　　エキスパートシステムをシェルとする開発環境

2.10　構造化分析を中心とした方法論

2.10.1　構造化分析の概要

ソフトウェアの開発はいくつかの段階を経て進められる。その中でも初期の段階、すなわち「あいまいな要求をそれなりに形式化した仕様書に変換する過程」（要求定義）を支援する方法論が構造化分析（Ｓｔｒｕｃｔｕｒｅｄ　Ａｎａｌｙｓｉｓ、略してＳＡとも呼ぶ）である。構造化分析は、出版物、セミナー会社、ＣＡＳＥツール・ベンダーなど様々な手段によって、１９７０年末頃から欧米を中心に広く普及した。現在、ＣＡＳＥツールと呼ばれているものは、この構造化分析を支援していることが多い。構造化分析は、様々な媒体を通じて普及したため別の呼び方をしたり、厳密にはその表記形式や方法論の指針などにおいて多少違いが見られる場合もあるが、一般に構造化分析と言えば、ＤｅＭａｒｃｏらが開発した構造化分析を指す[1)]。

ＤｅＭａｒｃｏらの構造化分析で使われる仕様書の内、その特徴をよく表わしているのがＤＦＤ（Ｄａｔａ　Ｆｌｏｗ　Ｄｉａｇｒａｍ）である（図2.10-1）。ＤＦＤは主として、機能



DFD 4.3　DFDを作成する

図2.10-1　Data Flow Diagram (DFD)

要求を表わすプロセスと、プロセスに入／出力するデータフローやファイルから成り立っており、プロセスには処理機能を表わす名前が、データフローにはフローが結んでいる要素間のインターフェイスを表わす名前が付けられる。ＤｅＭａｒｃｏらの構造化分析では、プロセスの起動タイミングやデータフローの処理タイミングは、注釈として表わすことを除けばＤＦＤに直接記述しないため、ＤＦＤの読み方は非常にシンプルである。例えば、図2.10-1のプロセス「データフローを分割する」とそれに入／出力するデータフローについては、単に、「上位層からのデータフロー群」は「データフローを分割する」に入りそこで「データフロー」に変換される、という読み方をするに過ぎない。

ＤＦＤ上のプロセスは、別の新たなＤＦＤへと詳細化できる。その結果、ＤＦＤは階層的に分割され、元のプロセスとそれに対応する新たなＤＦＤとの間には平衡性が成り立つ。すなわ

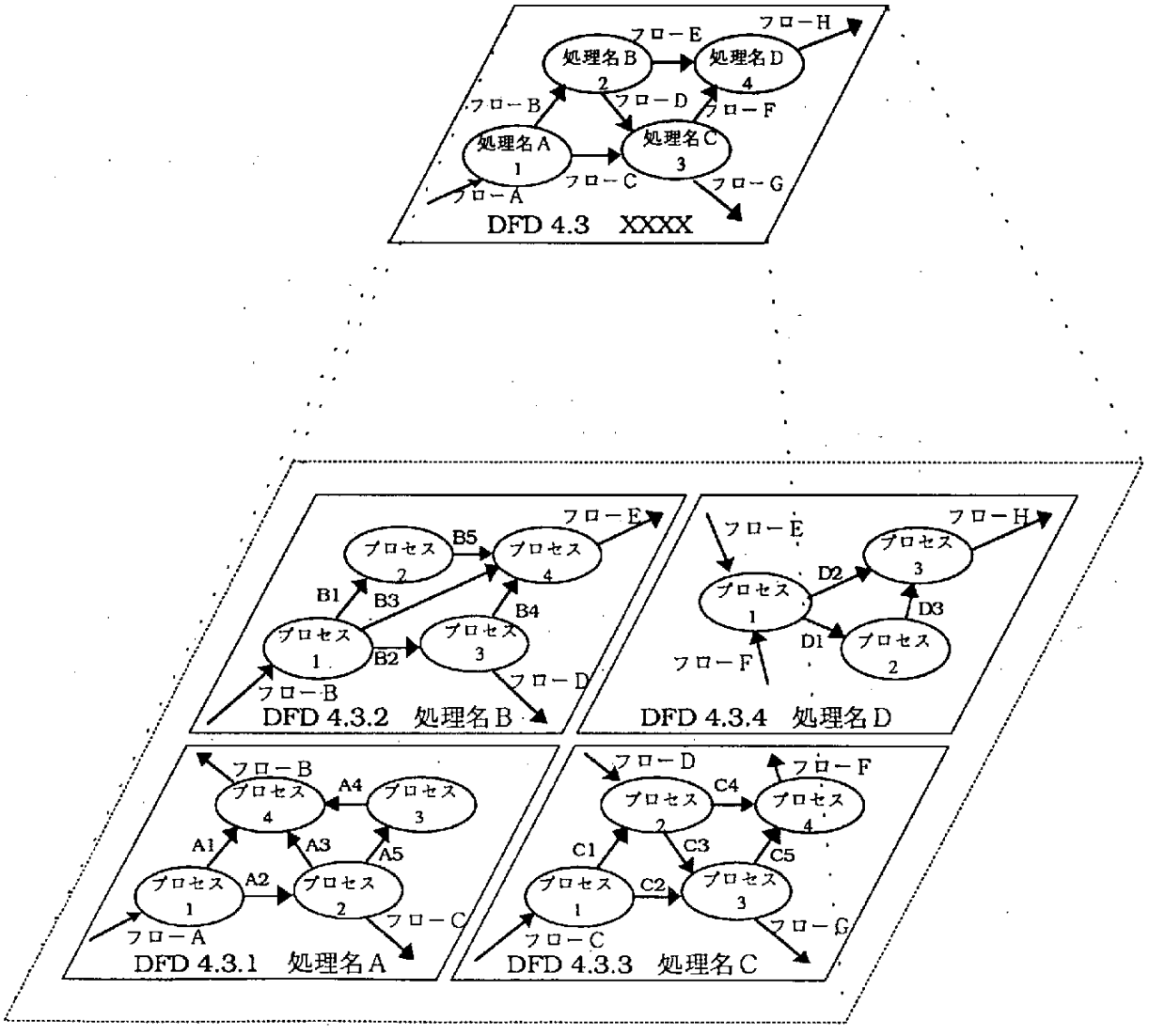


図2.10-2　DFDの階層間における平衡性

ち、元のプロセスの名前や番号は、新たなＤＦＤの名前や番号と一致する（番号に関しては、便宜上、元のプロセスが属するＤＦＤの番号の最小桁に、元のプロセス番号を追加したものを新たなＤＦＤの番号とする場合が一般的である）。さらに、元のプロセスへの入／出力データフロー名は、新たたなＤＦＤへの入／出力データフロー名と一致する（図2.10-2）。この平衡性のおかげで、ＤＦＤの詳細化において抜けや矛盾がないかを容易に検証できる。

これ以上分解できない（プリミティブな）プロセスは、ミニ仕様書に置き換える（図2.10-3）。ミニ仕様書には、設計に必要なすべての情報を記述する。記述の仕方は、構造化言語、デ

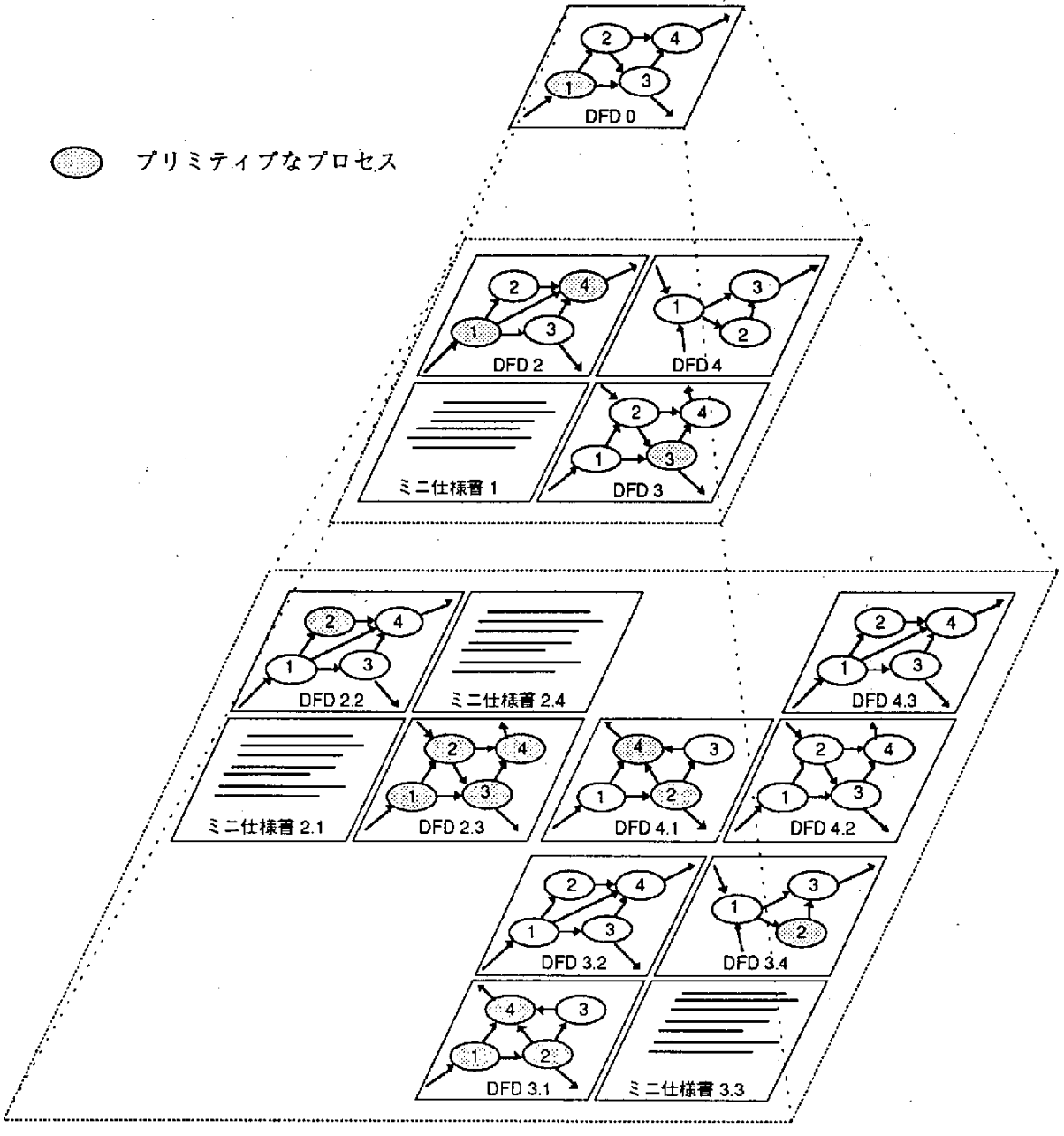


図2.10-3　全体から見た階層構造

シジョン・テーブル、デシジョン・ツリーなどを用いるのが一般的だが、制御構造を記述できるものであれば他の手段を用いても構わない。プロセスの分割に並行して、データフローの分割／クループ化もなされる。その際、データフローをどう定義したかをデータ・ディクショナリに記述する。また、データ・ディクショナリにはファイルの定義も記述する。

以上３つの仕様書（ＤＦＤ、ミニ仕様書、およびデータ・ディクショナリ）は、ＤｅＭａｒｃｏらの構造化分析で使われる主な仕様書であり、それぞれは互いに密接に絡み合っている（図2.10-4）。

要求仕様書を作成した後は、それを「どう実現するかを決定」（設計）する必要がある。その場合の方法論としては、構造化設計（Ｓｔｒｕｃｔｕｒｅｄ　Ｄｅｓｉｇｎ、略してＳＤと

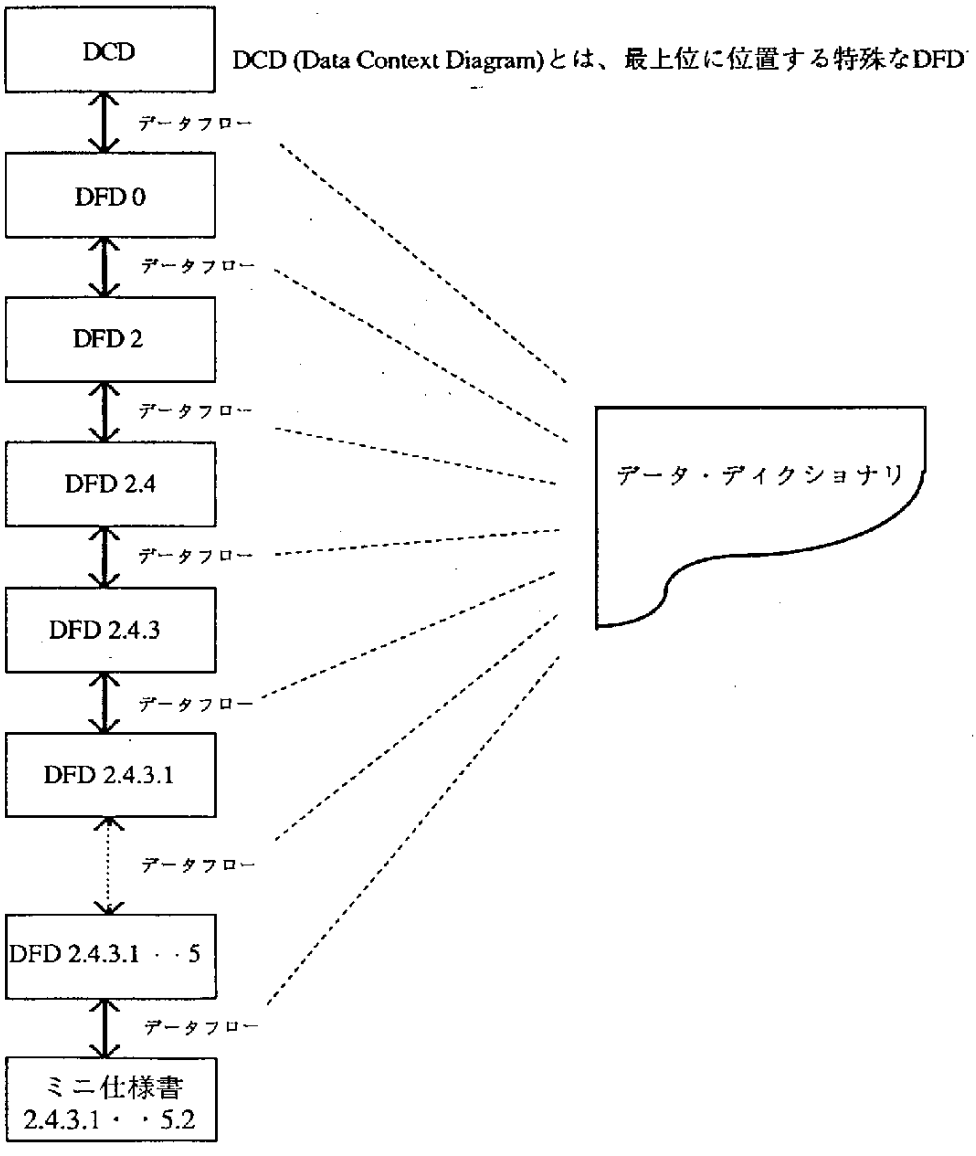


図2.10-4　３つの仕様書のつながりを示す複合図

も呼ぶ）を採用するケースが多く見受けられる[2]。

2.10.2　効果

ＤｅＭａｒｃｏらの構造化分析によるソフトウェア開発は、１９７０年代末から１９８０年代始めにかけて盛んに行なわれている。

米国の４００余りの研究機関に対して１９８７年に実施されたアンケート調査では、何らかの標準化技術を使っていると答えた研究機関の内、６割以上がＤｅＭａｒｃｏらの構造化分析、もしくはそれに準じた方法論を使用しているという結果が出ている（表2.10-1）。さらに、このアンケート調査では、何らかの標準化技術を使っていると答えた研究機関に「標準化技術を使うことによってソフトウェア開発の生産性が向上したと感じているか」尋ねたところ、構造化分析を使っている研究機関の多くが「イエス」と答えているのに対して、それ以外の標準化技術を使っている研究機関のほとんどが「あまり向上したとは感じられない」と答えている。

表2.10-1　標準化手法に関する調査結果

	古典的な方法	1
※	**Data Flow Diagram**	32
※	**Gane-Sarson**	1
	HIPO	12
	Jackson	1
	自己流の方法	6
	METHOD/1	2
	SDM-70	3
※	**STRADIS**	1
	Warnier-Orr	4
※	**Yourdon**	20
	標準化手法を使用せず	103
	計	186

※　ＤｅＭａｒｃｏらの構造化分析、もしくはそれに準じた方法論

[Eastern Washington大学、Marshall Drummond and Arther Reitsch
「Programmer Productivity and User Satisfaction Changes Related to Design Methods」
12th Structured Methods Conference, pp.67-73, 1987年]より引用

2.10.3　適用上の注意

ＤｅＭａｒｃｏらの構造化分析は欧米を中心に広く普及したものの、プロセスの起動タイミングやデータフローの処理タイミングなどの記述には積極的に取り組んでいないため、動作パターンが時間に依存するリアルタイム・システムの分野にはあまり受け入れられなかった。

この問題に対処するため、１９８２～３年頃からＰａｕｌ　Ｗａｒｄらが中心となり、制御とタイミングの要求をとらえるべくＤｅＭａｒｃｏらの構造化分析の拡張に着手した[39]。また、

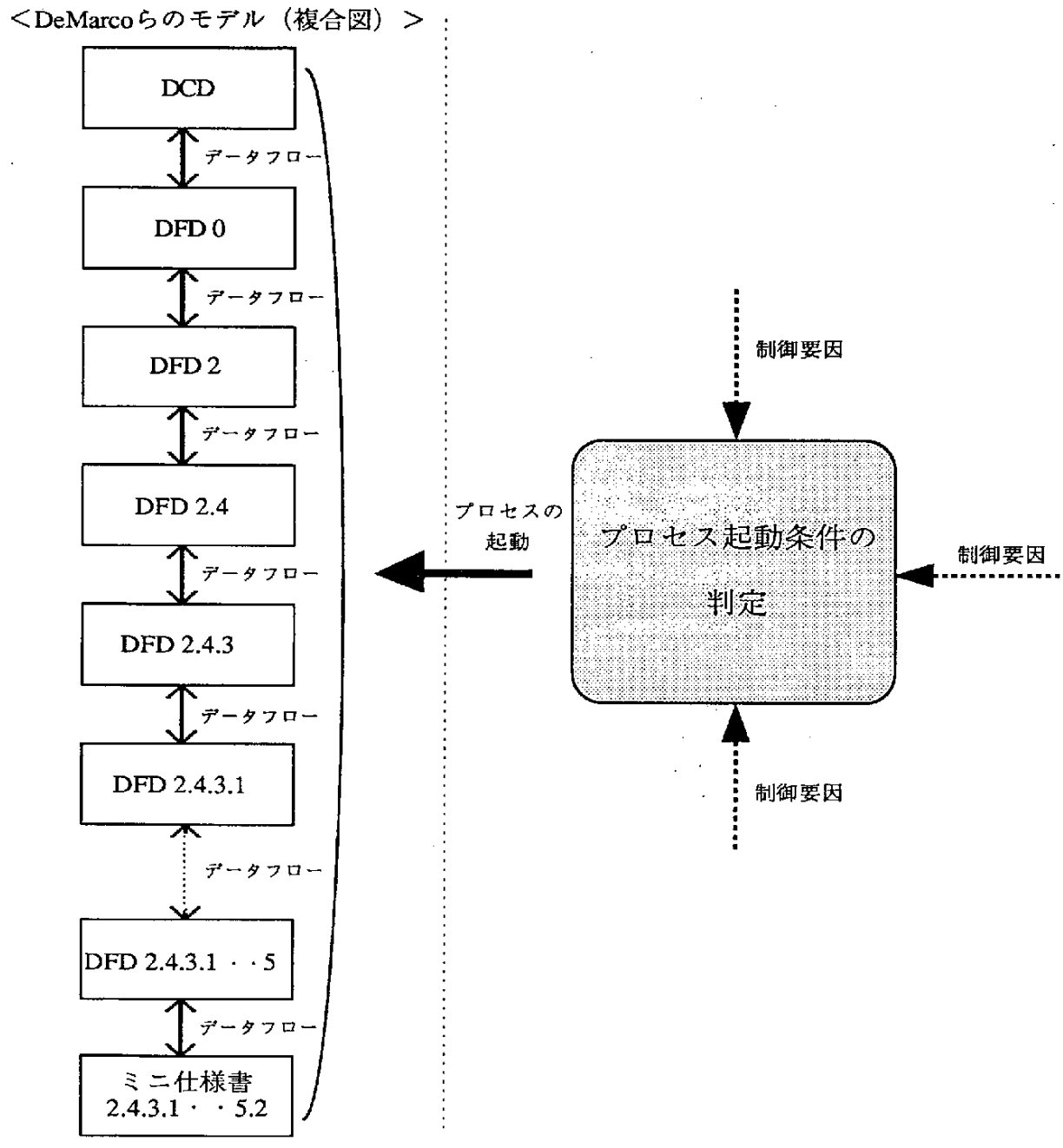


図2.10-5　DeMarcoらの構造化分析を拡張

これに時期をほぼ同じくして、Ｄｅｒｅｋ　Ｈａｔｌｅｙらは、ボーイング社の技術者数名と共同で、同じ目的を達成するための研究を開始した[5][6]。これら二つの研究成果はまるで異なっていたが、その表記方法は非常に似通っていた。前者はＷａｒｄのリアルタイムＳＡ、後者はＨａｔｌｅｙのリアルタイムＳＡと呼ばれている。

最近のＣＡＳＥツールは、少なくともどちらか一方の方法論を支援することが多くなってきている。これら２つの方法論は、その基本部分がＤｅＭａｒｃｏらの構造化分析に似ているため、これまで構造化分析を利用してきた人々には、それほど抵抗なく受け入れられた。特に、

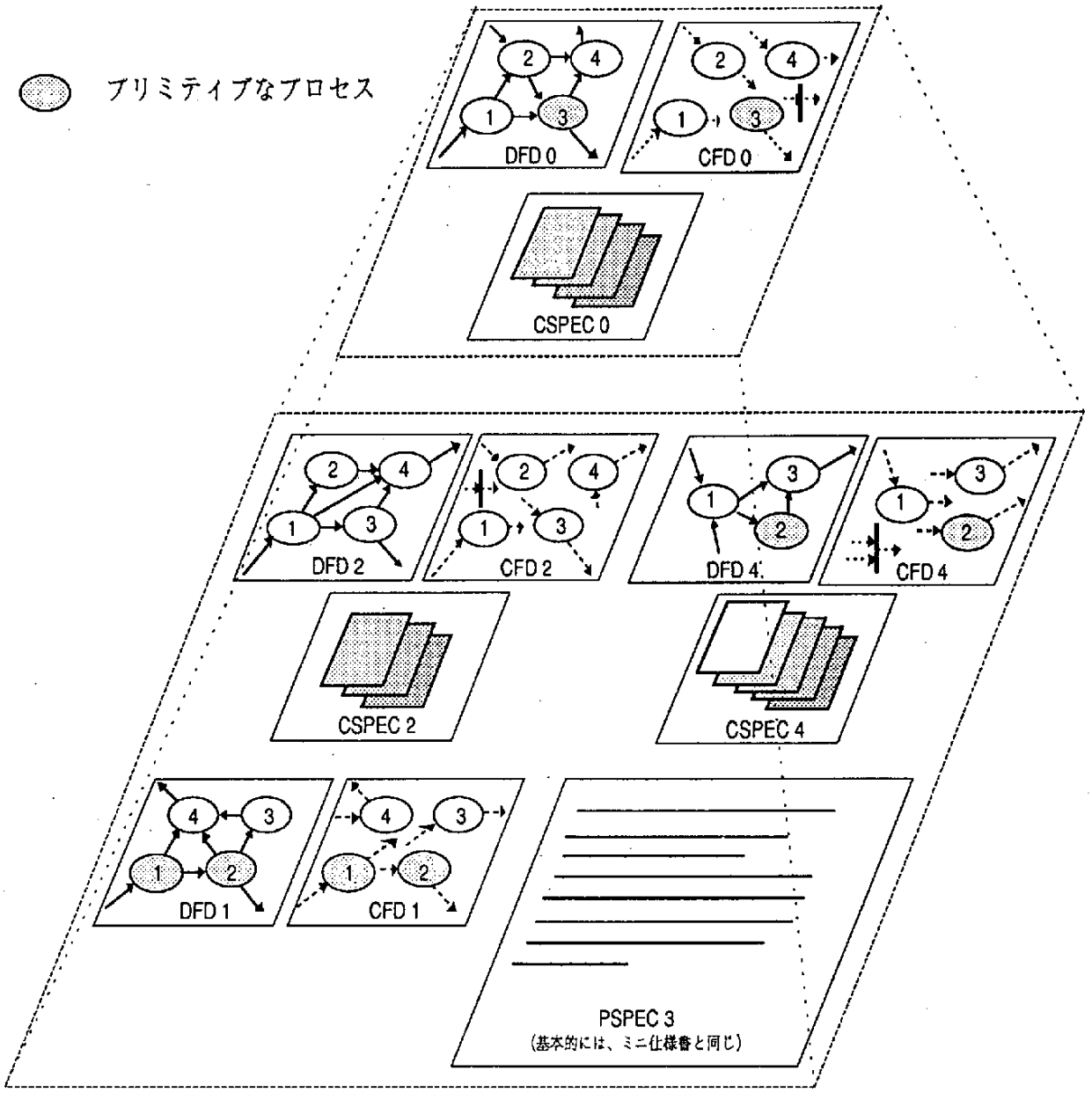


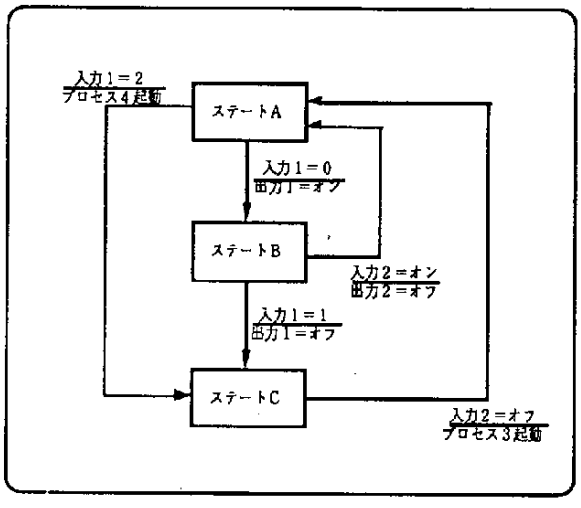
図2.10-6　全体から見た階層構造

ＨａｔｌｅｙのリアルタイムＳＡは、ＤｅＭａｒｃｏらの構造化分析の特徴であった階層性や平衡性をできる限りそのまま残し、その上から新たにプロセス起動の判定メカニズムを付け加えたようになっているので、処理部分と制御部分を分けて理解することが容易である（図2.10-5）。

ＨａｔｌｅｙのリアルタイムＳＡでは、プロセスの起動条件を判定するメカニズムはＣＳＰＥＣ（Ｃｏｎｔｒｏｌ　Ｓｐｅｃｉｆｉｃａｔｉｏｎ）で表わし、その判定要因となるフロー（制御フロー）はＣＦＤ（Ｃｏｎｔｒｏｌ　Ｆｌｏｗ　Ｄｉａｇｒａｍ）上で示すようになっている（図2.10-6）。ＣＳＰＥＣでは、読みやすさの点を配慮した有限状態マシン（Ｆｉｎｉ

入力		出力		プロセス					
入力１	入力２	出力１	出力２	1	2	3	4	5	6
0	don't care	オン	オフ	1	0	0	0	0	0
1	オフ	オフ	オン	0	1	0	0	0	0
	オン	オン	オフ	0	1	2	3	0	0
2	オン	オン	オフ	0	0	0	1	0	0
	オフ	オフ	オン	0	0	0	1	1	2

プロセス起動テーブル



状態遷移図

現在のステート	イベント	アクション	次のステート
ステートＡ	入力１＝０	出力１＝オン	ステートＢ
	入力１＝２	プロセス４起動	ステートＣ
ステートＢ	入力１＝１	出力１＝オフ	ステートＣ
	入力２＝オン	出力２＝オフ	ステートＡ
ステートＣ	入力２＝オフ	プロセス３起動	ステートＡ

状態遷移テーブル

	ステートＡ	ステートＢ	ステートＣ
ステートＡ		入力１＝０ 出力１＝オン	入力１＝２ プロセス４起動
ステートＢ	入力２＝オン 出力２＝オフ		入力１＝１ 出力１＝オフ
ステートＣ	入力２＝オフ プロセス３起動		

状態遷移マトリックス

図2.10-7　ＣＳＰＥＣを構成する代表的な仕様書

te State Machine）を実現するため、いくつかの表現形式（プロセス起動テーブル、状態遷移図、状態遷移マトリックス、および状態遷移テーブルなど）を採用している（図2.10-7）。

Hatleyのリアルタイム SAは、DeMarcoらの構造化分析の良さと、リアルタイム・システムへの対応力を兼ね備えているため、あらゆる分野のシステムに幅広く適用することが可能である。

2.10.4 その他

CASEについて論じる場合、CASEツールそのものについての話と、そのツールが支援する方法論についての話を分ける必要がある。CASEツールとは、ある特定の方法論を使って開発するユーザを、単に側面から支援するためのものにすぎない。そのため、「基礎になるメソドロジを理解しないでCASEツールを買っても効果は出ない。すなわち、CASEツールはメソドロジをソフトウェアとして実現しただけのものであり、メソドロジを理解しないでいくらやってもダメ」[7]なのである。また、「とのツールも手法を的確に支援しているように見えるが、現実はそうではない」[8]ので、CASEツールを評価する場合、そのCASEツールがどれだけ特定の方法論を忠実に支援できているかをよく見ておく必要がある。

[参考文献]

1) Tom DeMarco, Structured Analysis and System Specification, Prentice-Hall/Yourdon Press, 1978.
（高梨・黒田訳、構造化分析とシステム仕様、日経ＢＰ社、１９８６年）

2) Meilir Page-Jones, The Practical Guide to Structured Systems Design, Prentice-Hall/Yourdon Press. (2nd Edition, 1988)
（久保・丸山訳、構造化システム設計、近代科学社、１９９０年〈予定〉）

3) Paul T. Ward and Stephen J. Mellor, Structured Development for Real-Time Systems, Prentice-Hall/ Yourdon Press, 1985.

4) Paul T. Ward, The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, IEEE Transactions on Software Engineering, Vol. SE-12, No.2, February, 1986, pp.195-210.
（立田訳、ワード氏のリアルタイムＳＡ手法－変換図：データフロー図の拡張により、制御とタイミングを記述、ｂｉｔ誌、Ｖｏｌ．２０、Ｎｏ．５、１９８８年５月号）

5) Derek J. Hatley, The Use of Structured Method in the Development of Large Software-based Avionics Systems, AIAA/IEEE 6th Digital Avionics Systems Conference, December, 1984.
（立田訳、ハトレー氏のリアルタイムＳＡ手法－大規模なソフトウェア主体の航空システムを構造化手法で開発、ｂｉｔ誌、Ｖｏｌ．１９、Ｎｏ．１４、１９８７年１２月号）

6) Derek J. Hatley and Imtiaz A. Pirbhai, Strategies for Real-Time System Specification, Dorset House Publishing, 1987.
（立田訳、リアルタイム・システムの構造化分析、日経ＢＰ社、１９８９年８月）

7) 日経コンピュータ、１９８９年１０月９日号、ｐｐ.５２－５３.

8) Derek J. Hatley, Criteria for the Automation of Real-Time Requirements Specifications, The Third Annual Excelerator User Conference, 1987.
（立田訳、リアルタイム構造化分析支援ツールの規格、日経データプロ・ソフト、NS2－108－006～017、1988年8月）

3. 海外事例調査報告

3. 海外事例調査報告

3.1 調査目的と方法

代表的なＣＡＳＥツールであるExcelerator(Index Technology社) やＩＥＷ(Knowledge ware社) 等は世界で10,000セット以上販売されていると言われている。

しかし、ある調査によれば、実際に使われているＣＡＳＥツールは全体の20％であり、７％のツールは導入後拒絶されている。また、30％のユーザは、いかにメーカの宣伝文句を読むかが最も重要な事項であるとしている。こうしたことはＣＡＳＥツール自体の評価が定まっていないこと及びＣＡＳＥツールは単に導入すれば効果を発揮するといったものでないことを意味している。そこで、欧米における先進ユーザを対象に、いかにＣＡＳＥツールを導入し、その効果はどのようなものであったか等について調査を実施することにし、英国ＯＶＵＭ社に調査を依頼した。ＯＶＵＭ社に依頼した内容は、表3.1－1に示すとおりである。

表3.1－1　質問表

Questionnaire

1. Company name
   size (turnover)
   (staff)
   (systems development staff)
   type of business
   what Case products do they use?
   (functions, documentation)

2. Use of Case
   2.1 Why were Case tools introdued/developed ?
   - to address a backlog ?
   - to speed up development?
   - skills shortage?
   - growth in system size/complexity ?
   - other reasons?

   2.2 What was the timescale for their introdution?

   2.3 What problems have been experienced:
   a) with the introduction of the tools
   - difficulty in selecting which to use
   - not as efficient as hoped
   - organisation was not prepared sufficiently
   - product not compatible with co's working practices
   - expense of tools
   - insufficicient hardware
   - user resistance
   - others
   b) with the tools themselves ?

   2.4 What criteria were used to select the tools ?
   - benchmarking
   - comparison of product literature
   - short-term trial
   - recommendation from others
   - others

   2.5 How were the tools introduced into your organisation?
   - on a pilot project
   - phased-in over a period
   - clean sweep approach
   - others

   2.6 What software development procedures do the tools cover ?

   2.7 What type of training accompanied the introduction of the tools ?
   - vendor presentation
   - consultancy training
   - others

2.8 If the results have not matched expectations, to what do you attribute this?
- mismatched to user's needs
- lack of vendor support
- difficult to install and maintain
- cumbersome user interface
- lack of interfaces to other of tools
- lack of experience in the use of tools
- others

2.9 How will the use of Case evolve in your organisation?
- will you extend the use of upper Case ?
- will you extend the use of lower Case ?
- will you employ reverse engineering products?
- Ipse?
- others?

2.10 Do you have a wishlist for Case vendors ?
- what about : interface standards
- openness of the system
- use of AI and other advanced techniques
- better productivity/quality achievement

3. Benefits

3.1 What benefits has the use of Case brought in terms of :

3.1.1
- Productivity
- at the system analysis phase
- at the system design phase
- at the program desin phase
- at the testing phase
- at the maintenance phase

3.1.2
- Quality
- better ease of use in the systems produced
- better integrity
- improved efficiency
- correctness
- reliability
- maintainability
- testability
- reusability
- flexibility
- portability
- interoperability
- others

3.2 What are your expectations for quality improvement in five years' time ?

3.3 What has been the effect of using Case tools :
- development time, standards enforcement, documentaion
- user feedback/involvement
- staff numbers, program lifetimes
- project control

3.2　調査結果の分析

3.2.1　調査対象の概要

(1)　調査対象企業

事例調査先の選択に当たっては次の基準を設けて行った。

・１年以上ＣＡＳＥツールの評価又は利用を行っていること。

・進行中又は完了したＣＡＳＥを用いた開発プロジェクトがあること。

・ＣＡＳＥをよく理解していること。

・ＡＷＢ(Analysis Work Bench) ツールを含んでいること。

・ＣＡＳＥ利用に積極的であること。

・航空、財務、製造、政府関係であること。

こうした観点から調査を行った企業は次のとおりである。

＜アメリカ＞

・Shearson Lehman Hutton　（証券）

・ＡＴ＆Ｔ　（通信）

・Federal Home Loan Mortgage　（証券）

＜イギリス＞

・Inland Revenue　　（ＵＫ税処理）

・Norwich Union　（保険）

＜フランス＞

・Grédit Agricole de L'ile-de-France（金融）

・Electricité de France　（電力）

・Rhône Poulenc　（化学）

なお、航空関係並びに米国の製造関係の調査は戦略的な問題から拒否された。

(2)　適用していた開発方法論

今回の調査で最も多く使われていた開発方法論は、ＬＳＤＭ(LBMS Structured Development Methodology)とその英国政府推選版ＳＳＡＤＭ(Structured System Analysis and Design Methodology)、Merise（仏で有名）、Method/1、Yourdon Structured Method 、Information Engineering 及びBackman であった。表3.2－1に方法論とそれを使っていた企業の対応を示す。

表3.2－1　適用していた方法論

	方　法　論	適　用　会　社
データモデル	Bachman	Shearson Lehman Hutton
	Information Engineering	Federal Home Loan
	SSADM/LSDM	Inland Revenue Norwich Union
	Merise	Rhône Poulenc
	Corig	Crédit Agricole
プロセスモデル	Yourdon ／ DeMarco	Federal Home Loan
	SSADM ／ LSDM	Inland Revenue Norwich Union
アクティビティ分解	Information Engineering	Federal Home Loan
その他	Matrix diagraming	ＡＴ＆Ｔ
	Entity life history	Inland Revenue Norwich Union

なお、ＬＳＤＭ／ＳＳＡＤＭ及びＣｏｒｉｇ／Ｍｅｒｉｓｅはヨーロッパで開発されたものであり、日本であまり馴染がないため、ここで簡単な説明を行う。

① ＬＳＤＭ

ＬＳＤＭは、構造化システム分析と設計のための方法論である。この方法論はデータモデリングに重点を置いており、最終的な製品は、ある特定のＤＢＭＳ又は第４世代言語の上に構築されることを想定している。

この開発方法論は次の３つの技法から構成されている。

・エンティティモデル…… システムのデータ構造を記述する。

・データフロー図 …… システムの情報の流れを記述する。

・エンティティ・ライフヒストリ …… システムのイベントとそれらのシーケンス及び関連する処理を記述する。

なお、これらは、Learmonth and Burchett社のAutoMateツールに実装されている。

② ＳＳＡＤＭ

ＳＳＡＤＭは英国政府関係の機関で標準として用いられているシステム分析、設計の開発方法論であり、英国政府のＣＣＴＡ(Central Computer and Telecommunications Agency)で開発された。この方法論は、システム分析、要求仕様、論理設計、データとプロセスの物理設計からプログラム設計迄をカバーしている。ＳＳＡＤＭはＬＳＤＭによく似ている。

ＳＳＡＤＭは、実現可能性調査(Feasibility Study)から始まって次の６つのステージがある。

・現行システムの分析

・要求システムの仕様

・データ設計

・プロセス設計

・詳細物理設計

各ステージはいくつかのステップに分解され、各ステップは更にタスクに分解される。ＳＳＡＤＭ reference マニュアルは、National Computing Centre 、Oxford Road、Manchester Englandから入手可能である。

③ ＭｅｒｉｓｅとＣｏｒｉｇ

Ｃｏｒｉｇ (Conception、Organization et Realisation en Informatique de Gestion) はＣＧＩ社により開発された方法論である。ＣｏｒｉｇはＣｏｒｉｇ－Ａ、Ｃｏｒｉｇ－Ｂ、Ｃｏｒｉｇ－Ｃから成っている。

Ｃｏｒｉｇ－Ａは全体的なビジネス計画作成、Ｃｏｒｉｇ－Ｂは要求分析、Ｃｏｒｉｇ－Ｃはアプリケーション開発のためのものである。マニュアル類はＣＧＩ社から入手可能である。

Ｍｅｒｉｓｅはフランスで広く使われており基本となるアイディアはＣｏｒｉｇからきている。これらは、情報システム開発の実質上の標準(de facto standard) になっている。

Ｍｅｒｉｓｅの概要は、Merise ou l'Informatigue avec Methode (CGI/Natham Nouvelle Librarie SA (ISBN:2 86479 959 6) より知ることがができる。

(3) 適用していたツール

ＣＡＳＥとはComputer-Aided Software Engineering のことであり、ここで、Software Engineeringは、ソフトウェア開発の全工程即ち、要求定義からシステムの開発、導入迄を含んでいる。また、保守や導入後の改善・改良迄も含むことができる。定義からいくとＣＡＳＥツールとは、これらのすべてのステージを支援するものとなる。しかし、しばしば、ソフトウェア開発過程の分析と設計ステージを支援するものとして捉えている場合がある。今回の調査先で使用していたツールは次のカテゴリーに分類される。

・アナリストワークベンチ（Fundation Method/1, Automate, Excelerator）

・コードジェネレータや４ＧＬ（Pacbase ）

・リエンジニアリングツール（Bachman Re-Engineering Product Set）

・戦略計画や企業モデルのためのツール（PC Prism）

・ＩＰＳＥ　（統合化プロジェクト支援環境）（Maestro, Project Manager's Workbench）

数ユーザはＡＷＢ（Analyst Work Bench）とコードジェネレータを組み合わせて利用していたが、それらの結合はまだ自動化されていなかった。プログラミングに関していくつかのＡＷＢは、プログラミング用のテンプレートを生成していたが、多くの場合は手作業を主体に行われていた。また、多くのユーザはライフサイクルツール（又はＩＰＳＥ）を使用していた。ライフサイクルツールとは、プロジェクト管理、構成管理、品質管理、ドキュメント管理用のツールのことである。リバース・エンジニアリングについては、今回の調査では１社のみであったが、将来利用したいという声が５社からあった。

(4) ＣＡＳＥ利用の文化的相異

今回の調査を通して、興味深い文化的な相異がみられた。これらはＣＡＳＥツールの適用に影響を与えていると考えられる。

1) フランス

フランスは、ソフトウェア開発に“知的”な取り組みを行っていることで知られている。彼らは、短期的な利益のためにエレガントでないツールを組み合せるといった方法よりもむしろ全体的に調和のとれた形での開発に興味をもっている。これは、言語的な問題から米国製のツールがフランスに簡単には入り込めないといった原因や、フランス人の物の考え方によっていると思われる。

2) イギリス

イギリスもフランスと同様にソフトウェア開発プロセスをカバーする方法論に多くの力点を置いている。ヨーロッパにおいては、イギリス、スウェーデン、オランダが方法論や技法の利用について進んだ国だといえる。ライフサイクルツールは英国のマーケットにおいて成功を納めてきており、今回調査した2つの会社でも利用してきた。

米国製のツールは、言語が同じことからイギリスでも広く使われている。しかし、イギリスのユーザは一般にアメリカのユーザよりも費用対効果に厳しくツール購入がより慎重であったり、購入決定を遅らせる傾向にある。

3) アメリカ

アメリカにおいては、彼らはＣＡＳＥを戦略上の武器の一つとみている。そうした意味で、今回調査したＡＴ＆ＴやShearson Lehman Huttonでは正当な動機を持ている。即ち、ＡＴ＆Ｔでは電話クレジットカードサービスという戦略上の目的で、また、Shearson Lehman Huttonでは生産性を上げて新しいサービスをより速く提供するといった目的でＣＡＳＥツールを利用していた。

Shearson Lehman Huttonは開発方法論を持たないでツールをつかうといった点で米国の典型である。同社では、Bachman 社の製品を導入する迄は、データベース開発に標準手順を持っていなかった。

3.2.2 ＣＡＳＥツールの導入と適用

ここでは調査結果を基にＣＡＳＥツールの導入と適用について分析してみる。

(1) ＣＡＳＥツールの導入又は開発した理由

今回の調査では7社がＣＡＳＥツールを購入し、自社開発をしていたのは1社のみであった。市販のツールを導入した理由としては、その開発方法論が従来用いていた方法論にマッチしていた又は開発環境に一致していたことを挙げていた。

Rhône Poulenc では全社的にソフトウェア開発の標準化を図りソフトウェア開発を“工業化”しようとしていた。Federal Home Loan は、Norwich Union がＬＳＤＭで、また Crédit AgricoleがCoring／Meriseで行ったような、全社的な開発方法論の標準を望んでいた。ＡＴ＆Ｔでは共通ツールセットや、ビジネス要求に基づいた開発ライフサイルを導入しようとしていた。Shearson Lehman HuttonがＣＡＳＥツールを導入した主な理由はＩＤＭＳからＤＢ２への移向

であった。英国の政府機関であるInland Revenueでは英国政府の開発方法論に基づいてシステムを開発する必要からＳＳＡＤＭを持つツールを導入していた。

共通システム開発環境は次のような利点をもたらすと考えている。

① システムインタフェースの共通化により、異った開発チームで開発されたシステム同志がお互いに会話ができる。

② ドキュメントが標準化されて作成され、すべての人に理解し易くなる。

③ システムが主尾一貫した方法で開発されるため信頼性が増す。

④ 開発方法論の導入やＣＡＳＥツールの利用により開発の遅れを減少させることができ、保守が容易になる。これらにより生産性及び品質の向上が期待できる。

Electricité de France （ＥｄＦ）は、市場に適合したツールがないとのことからツールを自社開発した。同社では原子力発電所のコントロール用にツールを開発している。この目的としては、より大きく、より複雑なシステムの開発のスピードアップを図ることを挙げていた。

(2) 導入スケジュール

調査結果から全体的な導入スケジュールの一般的な傾向をみることは難しい。すべてのユーザは違ったスケジュールでＣＡＳＥを導入していた。

ＡＴ＆ＴやInland Revenueでは管理部門からＣＡＳＥツールの導入が支持されており迅速に導入しているし、一方、Rhône Poulenc ではツールの導入に５年を要していた。Federal Home Loanでは管理部門から完全な支持を得たとはいえず導入計画を誤ったものにしていた。即ち、社内的な体制がとられるよりも早くツールを導入してしまった。

ＣＡＳＥを導入するスケジュールを考える上での主要な要因は次のとおりであった。

- ツールの必要性（システムのバックログの増大、エンジニアの不足等）
- 管理部門からの支持の度合
- ニーズに合ったツールの出現時期
- ユーザからの反発の度合

(3) ツール導入・利用上の問題点

ＣＡＳＥツールを導入する上での障害として６つの会社がユーザの抵抗を挙げていた。この内、２つの会社は深刻な状態であった。

Electricité de France ではツールを自社で開発しているため、そのツールが社内で正当に理解されない(市販ツールのような評価がない)という問題を持ったいた。Federal Home Loan

での抵抗は、試用時における不幸な経験に根ずいていた。

共通的に指摘された問題としては、ツールを開発現場へ適合させて行く上での難しさや、ユーザが開発方法論を理解していないことなどがあった。

Rhône Poulenc では、ツールとしては良いが自社の開発方法論とマッチしていないツールを導入したため、両者のミスマッチを経験していた。Crédit Agricole も同様に自社へのPacbaseツールの導入の困難さを指摘していた。

ツール自体への不満としては、ツール提供者が言う機能を満たしていないことが挙げられていた。もっとも今回調査したユーザはＣＡＳＥ等に十分な経験を持ち、低めの予想を持っていたため、むしろ予想よりはよく出来ていることに驚いていた。導入担当者は、職場にツールを広める役割りを持っており、利用し易いツールを望んでいた。特にＡＴ＆Ｔでは Excelerator がニーズにあっていないと不満をのべていた。しかし、そのツールを替える様子はなかった。

いくつかのツール、例えばBachman 社のリエンジニアリング製品やＥｄＦのGenesia は大変複雑で、利用を広めるのに困難を感じていた。例外なく初期のツールには欠陥がみられた。しかし、その後のリリースで改善されたり改良されているため現在では特に問題となっているものではない。いったんツールが現場に導入されるとユーザは欠点が何であろうとほとんど変更しようとはしない。

それ故、ユーザのツール選択はツール提供者が、ユーザの要求に合った開発を行えるか否かによって導入後迄大きく影響を与える。他の不満としてはコスト（特にBachman 社のツールは競争相手がいないため非常に高価である）やマルチユーザ環境でのサポートの不備があった。

この調査から次のことが言える。

- ユーザは、既存の環境にツールが適合するかよく検討すべきである。さもなくば、導入前に十分な教育を行うべきである。これらがなければ混乱や反抗を招くだけである。
- ツール提供者がユーザ要求に応じてツールの変更を行ったり新しい機能を付け加えてくれるか検討すべきである。

(4) ツールの選定規準

ＣＡＳＥツールの選択に当たっては、ＣＡＳＥのカタログ、デモンストレーション、ツールに対する知識、評価コピー、コンサルタントからの意見を参考にしていた。

Norwich Union 、Rhône Poulenc 、Inland Revenueは、ツールに対する要求仕様リストを持っていた。しかし、これらのユーザは、ツールが彼らの要求を満たしていないことに不満をも

っていた。Inland Revenueは彼らの要求仕様に合うものとしてMaestroを、またRhône Poulencは評価後ツールを選定していた。Norwich Union では最終候補として３つのツールを選んでいた。

ＡＴ＆Ｔでは選択に対して一貫した過程をもっていないことを認めていた。またInland Revenue とFederal Home Loan はスタッフによる内部的な推薦に従っていた。

多くの所では、購入前にツールの実践的評価を行っていたことを強調しておく。

(5) ＣＡＳＥツールの導入方法

８ユーザの内５ユーザ（Rhône Poulenc、Crédit Agricole、Federal Home Loan、Shearson Lehman Hutton、Norwich Union ）ではＣＡＳＥツールをまずパイロットプロジェクトで適用していた。

面白いことに素早くツールを導入したInland RevenueとＡＴ＆Ｔでは導入に対してほとんど抵抗がなかった。これらは、ユーザにツール選択の余地がなかった、管理者の介入が強かった、ツールを気入らなかった人は会社を去ったことによると考えられる。

パイロットプロジェクトでの適用の内、４ユーザは本番システムで適用していた。一方Shearson Lehman Huttonは実験的にツールを使っていた。Electricité de France の場合はユニークである。即ち、Genesia の開発者自身が、ＥｄＦ内の種々の部門のプロジェクトを遂行していた。この”デモンストレーション”プロジェクトの目的はそのツールが本番システムでも利用できることを証明することであった。このような方法でツールの価値を認めた後、初めてこれらの部門で利用し始めることになる。

(6) ツール導入に伴う教育

トレーニングの形態は導入したユーザにより大きく異る。Inland RevenueやNorwich Union は社内に専門的なトレーニング部門を設けていた。Norwich Union では、特別なツールに捕われることなくＬＳＤＭ開発方法論を教育するために自社教育を行っていた。

Crédit AgricolとShearson Lehman Huttonではツール提供者の教育に依存していた。Shearson Lehman Huttonのケースでは、Bachman 社のツールが複雑で、それらを利用するための知識を持ち合わせていなかったことによる。大半のユーザは社内、社外のミックスした教育パッケージをもっていた。

(7) ツールがカバーする開発工程

多くのユーザは分析並びに設計用のツールを持っていた。これは調査自体が設計用ツールに

焦点を当てたことによる。表3.2－2に調査先とツールによって支援されている工程を示す。

表3.2－2　ツールがカバーしている工程

調　査　先	ツールにより支援されている工程
ＡＴ＆Ｔ	Business analysis, requirements specification, systems design, implementation, documentation, project management
Federal Home Loan	Business analysis, systems design
Shearson Lehman Hutton	Requirements specification, systems design, implementation, maintenance
Norwich Union	System design, implementation, project management, quality management
Inland Revnue	Implementation, project management, documentation management
Rhône Poulenc	Systems analysis, systems design
Crédit Agricole	Systems analysis, systems design, documentation, implementation
Electricité de France	Systems design, implementation

(8)　期待した効果を上げていない理由

ユーザは一般にツールが予期した効果を上げているか否か、また、何が成功又は失敗に導いたかについて明確な見解を持っているが、ＡＴ＆ＴとInland Revnue を除いて成功又は失敗のメジャーを得ることができなかった。他のユーザからのコメントは主観的であり、また多くの期待値は期待はずれに終らないためにかなり低く見積っており、ほとんど何の期待も持っていなかったようにみえる。

Norwich Union 、Rhône Poulenc とFederal Home Loan は期待外れであったと述べている。Norwich Union の場合は過剰な期待があったものと考えられる。Rhône Poulenc は現実的な期待を持っていた。即ち彼らは、彼らの選んだツールが必ずしも満足した結果を出すとは思って

いなかったが、ドキュメント作成への支援や方法論支援上の問題に不満を持っていた。Federal Home Loanではユーザインタフェースの悪さを期待外れの最大の原因にしていた。Electricité de Franceでは社内で開発しているため期待した効果を上げていると述べている。Crédit Agricoleでは当初の期待が何であったかを忘れるほど長くPacbase を使っていた。

Inland Revnue では、生産性に関して期待した効果を上げていたが、更に、Maestro で向上させようとしていた。ＡＴ＆ＴとShearson Lehman Huttonのみが期待以上であったと述べている。ＡＴ＆Ｔでは測定の基礎として開発期間を用いており、当初システムの概念設計から開発迄に２年かかると予測していたものが１８ケ月に短縮されたと述べている。Shearson Lehman Huttonでは実験的にツールを利用し始め、結果として生産性向上が予測を上回っていたと述べている。

(9) 今後の計画

今後の計画としては共通的な傾向が見られる。４つの共通的な傾向は次のとおりである。

- リバースエンジニアリング（５ユーザ）
- リポジトリ／ツールの統合（４ユーザ）
- 新しい方法論のサポート（３ユーザ）
- よい良いコード生成（４ユーザ）

リポジトリ／ツールの統合のためのリポジトリの候補としてはＩＢＭのＡＤ／Cycle とSoftlab社のMaestroがあった。ＩＢＭのリポジトリは多分多くのツール提供者が使用するde facto標準となろう。

ＡＴ＆ＴではＩＰＳＥをリポジトリの上に構築しようとしていた。

(10) ＣＡＳＥベンダーへの要求

この質問は(9)の質問にクロスする。しかし、要求の多くはツールの機能的な面であった。ユーザは、より強固で、より良いユーザインタフェースを持ち、方法論をサポートし、カスタマナイズでき、開放型でしかもユーザ要求に対応がとれるツールを望んでいた。２ユーザは特にＯＳ／２のサポートを望んでいた。他はＵｎｉｘ、マルチユーザ、ネットワーク化を挙げていた。 また、アセンブラーの生成（Crédit Agricole ）、オブジェクト指向の設計、プログラム作成の支援（ＡＴ＆Ｔ）、プロジェクト、品質管理機能（Norwich Union ）等もあった。更にエキスパートシステム開発への支援といった要求もあった。

3.2.3　ＣＡＳＥツールの導入効果

(1)　現状における生産性の向上

先にも述べたとおり、生産性と品質の測定方法は主観的であり、どの程度ＣＡＳＥがソフトウェア開発に影響を与えているかは感覚的なものであった。

これは、生産性及び品質の共通的な測定尺度がないことに起因している。表3.2－3は、今回の調査で得られた結果をまとめたものである。定量化できたものは、その値も示している。

(2)　現状における品質の向上

品質向上の測定尺度を示したのはＡＴ＆Ｔのみであった。ＡＴ＆Ｔでは品質を製品完成後の変更の度合に置いている。この観点から９７％の品質向があったとのべている。即ち、変更が９７％減少した。

他のユーザは、信頼性、保守性、互換性、正確度において向上が見られたが、その度合を示すことは出来ないとしていた。

(3)　５年後の生産性向上の予測

ここでは予測であるから期待が多く語られた。しかし、これを証明することは多くのデータ収集を必要とすることから困難であろう。事実、そのような活動は時間がかかるものとして省略されている。多分、最も信頼できる予測はＡＴ＆Ｔのものであろう。彼らは、システム開発コストを少なくとも４０％また、開発期間を少なくとも２５％減少させようとしている。Norwich UnionはCrédit Agricole の場合と同様に５０％の削減を望んでいた。

Rhône Poulenc の期待はもっと穏やかなもので現状における３０％～４０％の開発期間の伸びを１０％にしたいとしていた。

(4)　５年後の品質の向上

この質問に対する回答はより曖昧なものであった。多くの回答者は品質向上に意欲は持っているものの、その尺度については明確なものを持っていなかった。

ＡＴ＆Ｔ、Norwich Union 、Shearson Lehman Huttonでは、よりよい要求仕様を作りその仕様に適合したシステムを作ることにより変更を減少させようとしていた。

表3.2－3　生産性の向上

調　査　先	生産性向上分野	向上の期待値(%)
AT&T	systems specification, testing	20%
Federal Home Loan	management of diagrams, data	'minimal'
Shearson Lehman Hutton	systems design	no metrics
Inland Revnue	coding management	15－20% no metrics
Norwich Union	Systems analysis programming	15% 200%
Crédit Agricole	Systems analysis, program generation, maintenance	40%
Electricité de France	program design, testing, maintenance	no metrics
Rhône Poulenc	Systems analysis	no metrics

付　録　　参考文献一覧

付録　参考文献一覧

(1) 単行本

1. E. J. Chikofsky : IEEE Computer Society Press Technology Series : Software Development : Computer-Aided Software Engineering (CASE), IEEE, 0-8186-1917-1, 1989
2. Rosemary Rock-Evans : CASE Analyst Workbenches : A Detailed Product Evaluation (VOL.1), OVUM, 0-903969-41-6, 1989
3. Rosemary Rock-Evans : CASE Analyst Workbenches : A Detailed Product Evaluation (VOL.2), OVUM, 0-903969-41-6, 1989
4. John Cameron : JSP & JSD : The Jackson Approach to Software Development (SECOND EDITION), IEEE, No-8186-8858-0, 1989
5. Richard E. Fairley : Software Engineering Concepts, McGRAW-HILL, 0-07-019902-7, 1985
6. Roger S. Pressman : Software Engineering : A Practitioner's Approach (SECOND EDITION), McGRAW-HILL, 0-07-050783-X, 1987
7. Capers Jones : Programming Productivity, McGRAW-HILL, 0-07-032811-0, 1986
8. Roger S. Pressman : Making Software Engineering Happen : A Guide for Instituting the Technology, PRENTICE-HALL, 0-13-547738-7, 1988
9. Tom Gilb : Princples of Software Engineering Management, ADDISON-WESLEY, 0-201-19246-2, 1988
10. Watts S. Humphrey : Managing the Software Process, ADDISON-WESLEY, 0-201-18095-2, 1989
11. James Martin / Joe Leben : Strategic Information Planning Methodologies (SECOND EDITION), PRENTICE-HALL, 0-13-850538-1, 1989
12. Michael W. Evans : The Software Factory - A Fourth Generation Software Engineering Environment -, JOHN WILEY & SONS, 0-471-01192-4, 1989
13. Alan S. Fisher : CASE : Using Software Development Tools, JOHN WILEY & SONS, 0-471-63747-5, 1988
14. Pearl Brereton : Software Engineering Environments, JOHN WILEY & SONS, 0-471-

21022-2, 1988

15. CASE : The Potential and the Pitfalls, QED, 0-89435-285-7, 1989
16. The Annual Case Survey, 1988, QED, 0-98435-286-5, 1989
17. James Martin / Carma McClure : Structured Techniques : The Basis for Case, PRENTICE-HALL, 0-13-854936-2, 1988
18. Carma McClure : CASE is Software Automation, PRENTICE-HALL, 0-13-119330-9, 1989
19. James Martin / Carma McClure : Diagramming Techniques for Analysts and Programmers, PRENTICE-HALL, 0-13-208794-4, 1985
20. Vaughan Merlyn : The Strategic Impact of CASE, CASE RESEARCH CORP., 0-39435-292-X, 1989
21. Martin E. Modell : A Professional's Guide to Systems Analysis, McGRAW-HILL, 0-07-042632-5, 1988
22. Brian Dickinson : Developing Quality Systems - A Methodology Using Structured Techniques - (SECOND EDITION), McGRAW-HILL, 0-07-016803-2, 1989
23. David A. March / Clement L. Mcgowan : SADT : Structured Analysis and Design Technique, McGRAW-HILL, 0-07-040235-3, 1988
24. Sally Shlaer / Stephen J. Mellor : Object-Oriented Systems Analysis : Modeling the World in Data, YOURDON PRESS, 0-13-629023-X, 1988
25. Meilir Page - Jones : The Practical Guide to Structured Systems Design (SECOND EDITION), YOURDON PRESS, 0-13-690769-5, 1988
26. Chris Gane : Computer - Aided Software Engineering : The Methodologs, The Products, and The Future, PRENTICE-HALL, 0-13-176231-1, 1990
27. Derek Nicholls : Introducing SSADM - The NCC Guide -, NCC, 0-85012-628-2, 1987
28. ｼｭｰﾏﾝ(著) 松本正雄 / 小泉正(訳) : ｿﾌﾄｳｪｱ工学講座(1)ﾌﾟﾛｸﾞﾗﾑの設計ﾂｰﾙと技法, ﾏｸﾞﾛｳﾋﾙ出版(株), 4-89501-090-2, 1989
29. ｼｭｰﾏﾝ(著) 菊地豊彦(訳) : ｿﾌﾄｳｪｱ工学講座(4)ｿﾌﾄｳｪｱの開発管理技術, ﾏｸﾞﾛｳﾋﾙ出版(株), 4-89501-093-7, 1987
30. ｼｭｰﾏﾝ(著) 寺本雅則 / 上村松男(訳) : ｿﾌﾄｳｪｱ工学講座(3)ｿﾌﾄｳｪｱの信頼性, ﾏｸﾞﾛｳﾋﾙ出版

(株), 4-89501-092-9, 1989

31. Tom Demarco(著) 渡辺純一(訳) : - 品質と生産性を重視した - ｿﾌﾄｳｪｱ開発ﾌﾟﾛｼﾞｪｸﾄ技法, 近代科学社, 4-7649-0133-1, 1989

32. Derek J. Hatley / Intiaz A. Pirbhai(著) : ﾘｱﾙﾀｲﾑ･ｼｽﾃﾑの構造化分析, (株)日経BP, 4-8222-7075-0, 1989

33. 佐藤正美 : CASEﾂｰﾙ - 機能解説と活用のﾉｳﾊｳ -, (株)ｿﾌﾄ･ﾘｻｰﾁ･ｾﾝﾀｰ, 4-915778-02-9, 1989

34. 原田実(編著) 大野豊(監修) : 自動ﾌﾟﾛｸﾞﾗﾐﾝｸﾞﾊﾝﾄﾞﾌﾞｯｸ, ｵｰﾑ社, 4-274-07539-7, 1989

(2) 論文及び雑誌記事

1. Michael Lucas Gibson : Implementing the Promise, Datamation, Feb.1, 1989, P65-67
2. Michael Lucas Gibson : A Guide to Selecting CASE Tools, Datamation, Jul.1, 1988, P65-66
3. Lori Weitz : Not Cannibalism, But Coexistence with CASE, Software Magazine, Oct., 1988, P43-59
4. Charlie Bachman : A CASE for Reverse Engineering, Datamation, Jul.1, 1988, P49-56
5. Gary McWilliams : Users See a CASE Advance in Reverse Engineering Tools, Datamation, Feb.1, 1988, P30-36
6. Jack M. Davis : "Ipo-Ipo" = PROGRESS, Software News, Dec., 1987, P68-71
7. John Desmond : Expert Systems in Software Development, Software News, Jul., 1987, P44-47
8. John Voelcker : Automating Software : Proceed with Caution, IEEE SPECTRUM, Jul.,1988, P25-27
9. David Barstow : Artificial Intelligence and Software Engineering, Exploring, Artificial Intelligence, AAA, P641-669
10. Software in the Far East, IEEE Computer, March, 1989
11. Murat M. Tanik : Rapid Prototyping in Software Development, IEEE Computer, May, 1989
12. Software Development, Datamation, Vol.1, No.1, 1989
13. V. Karakostas : Requirements for CASE Tools in Early Software Reuse, ACM SIGSOFT, Software Engineering Notes, Vol.14, No.2, Apr., 1989, P39-41
14. Lin Zucconi : Selecting a CASE Tool, ACM SIGSOFT, Software Engineering Notes, Vol.14, No.2, Apr., 1989, P42-44
15. Carma McClure : Methodology in Path from Art to Science, Software Magazine, June, 1989, P33-41
16. Ellen Berman : A Best CASE Scenario, System Integration, June, 1989, P49-66

17. Michael Puttré : Repositories Get Down to Business, Information Week, May 29, 1989, P12-13
18. Luqi : Rapid Prototyping Languages and Expert Systems, IEEE EXPERT, Summer, 1989, P2-5
19. Rommert J. Gasimir : Forth Generation Problems, SIGPLAN NOTICES, Vol.24, No.5, 1989 P83-86
20. J. M. Carey 他 : The Prototyping Conundrum, Datamation, June 1, 1989, P29-33
21. T. A. Corbi : Program Understanding : Challenge for the 1990s, IBM Systems Journal, Vol.28, No.2, 1989, P294-306
22. L. Cleveland : A Program Understanding Support Environment, IBM Systems Journal, Vol.28, No.2, 1989, P324-344
23. O. De Troyer : RIDL* : A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints, Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data, SIGMOD RECORD, Vol.18, No.2, June, 1989
24. Marcus Loh, R. Ryan Nelson : Reaping CASE Harvests, Datamation, July 1, 1989, P31-34
25. George Schussel : Why Software is IBM's Most Important Business, Software Mgazine, July, 1989, P82-85
26. Barbara M. Bouldin : Culture Has Impact on Data Processing, Software Magazine, June, 1989, P73-77
27. Michael Lucas Gibson : The CASE Philosophy, Byte, April, 1989, P209-218
28. Ken Orr 他 : Methodology : The Experts Speak, Byte, April, 1989, P221-233
29. Carma McClure : The CASE Experience, Byte, April, 1989, P235-246
30. Charles Rich 他 : The Programmer's Apprentice : A Research Overview, IEEE COMPUTER, Nov., 1988, P10-25
31. Roberto Bisiani 他 : A Tool to Cordinate Tools, IEEE EXPERT, Nov., 1988, P17-25
32. Gail E. Kaiser 他 : Intelligent Assistance for Software Development and Mentenance, IEEE Software, May, 1988, P40-49

33. Gail E. Kaiser 他 : Database Support for Knowledge-Based Engineering Environments, IEEE EXPERT, Summer, 1989, P18-32

34. Charles Rich : Automatic Programming : Myths and Prospects, IEEE COMPUTER, Aug., 1988, P40-51

35. Sridhar A. Raghavan 他 : Diffusing Softwave - Engineering Methods, IEEE Software, 1989, P81-90

36. Sudha Ram 他 : An Automated Tool for Relational Database Design, Information Systems, Vol.14, No.3, 1989, P247-259

37. Barbara M. Bouldin : What are you measuring ? Why are you measuring it ?, Software Magazine, August, 1989, P30-39

38. Damian Rinaldi : Getling Beyond Drawings, Software Magazine, April, 1989, P51-58

39. Tom DeMarco : Looking for Lost Keys, Software Magazine, April, 1988, P58-62

40. Edith Myers : Back-End Tools Seek Front-End Companions, Software Magazine, September, 1988, P49-68

41. John Desmond : A Visionary and a Motivator Deliver the CASE Message, Software Magazine, 1988, P95-104

42. Warren Harrison : PDSS : A Programers Decision Support System, Data & Knowledge Engineering 4, 1989, P115-123

43. David Stamps : CASE vs 4GLs, Datamation, Aug.15, 1989, P29-32

44. Joseph Fiksel 他 : Knowledge Systems for Planning Support, IEEE EXPERT, FALL, 1989, P16-23

45. Ali Hazzah : Improvements in Tools Leading to Second Look, Software Magazine, Sept., 1989, P31-47

46. Chi Y. Lin 他 : Computer-Aided Software Development Process Design, IEEE, Trans. on Software Engineering, Vol.15, No.9, 1989, P1025-1037

47. R. P. Ten Dyke 他 : Obiect-Oriented programming, IBM Systems Journal, Vol.28, NO.3, 1989, P465-478

48. Allen L. Ambler 他 : Influence of Visual Technology on the Evolution of

Language Environments, IEEE COMPUTER, Oct., 1989, P9-22

49. Ali Hazzah : Everything is in the Names for Coming "CAPTURE" Tools, Software Magazine, Oct., 1989, P40-50
50. Jeff Moad : Workstations Win a Big Blue CASE, Datamation, Oct.1, 1989, P37-39
51. Jeff Moad : Cultural Barriers Slow Reusability, Datamation, Nov.15, 1989, P87-92
52. T. Capers Jones : Why Choose CASE?, BYTE, Dec., 1989, P80IS-3-80IS-10
53. The BYTE Staff : Making a Case for CASE, BYTE, Dec., 1989, P154-171
54. Won Kim : A New Database for New Times, Datamation, Jan.15, 1990, P35-42
55. Ralph Carlyle : Is Your Data Ready for the Repository ?, Datamation, Jan.1, 1990, P43-47
56. Elliot J. Chikofsky 他 : Reverse Engineering and Design Recovery : A Taxonomy, IEEE Software, Jan., 1990, P13-17
57. Mary Alice Hanna : Move is on to Tie Vision to Information Systems, Software Magazine, Jan., 1990, P39-45
58. Jeff Moad : The Software Revolution, Datamation, Feb.15, 1990, P22-30
59. Alexander J. Polack : Practical Applications of CASE Tools on DoD Projects, ACM SIGSOFT, Software Engineering Notes, Vol.15, No.1, Jan., 1990, P73-78
60. Veri-Pekka Tahuanainen 他 : An Annotated CASE Bibliography, ACM SIGSOFT, Software Engineering Notes, Vol.15, No.1, Jan., 1990, P79-92

平成２年３月発行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園３丁目５番８号
機　械　振　興　会　館　内
ＴＥＬ０３（４３２）９３７２

印刷所　株式会社　タ　ケ　ミ　印　刷
東京都千代田区神田司町２－１６
ＴＥＬ０３（２５４）５８４０

01-R016