5章 開発環境

5・1 概　説

■ 5・1・1 目　的 ■

開発環境とは，狭い意味では，ソフトウェアの開発のためにコンピュータ上で利用するツールの集合であるが，実際にはそれらのツールが提供する開発技法に加えて，それらのツールの操作対象となるデータの管理，それらのツールを有効活用するための開発方法論やプロセス管理，さらには開発担当者まで含めたプロジェクト管理も関連している．むしろ開発方法論の実現手段として，あるいはプロセス管理やプロジェクト管理の一環として開発環境ないしツールがあると考えるべきである．

したがって，開発環境の目的は，一般には「良いものを早く安くつくること」すなわち高品質ソフトウェアの高効率生産を促進することである．

これまでのソフトウェア生産は，職人芸的生産技術および労働集約型生産形態に依存してきたため，実際にソフトウェアを開発する場合は以下のような問題があった．

(1) 開発する人がいない ← 情報処理技術者の不足
(2) 開発に時間がかかる ← バックログ増大
(3) 開発コストが高い ← 人件費増大
(4) 保守コストが高い ← ソフト資産増大
(5) 品質が不十分である ← 大規模化と複雑化

このような課題に対して一般的には以下のような解決方法がある．

(1) 開発手段（ツール）の高機能化
(2) 開発工程（プロセス）の定式化，自動化
(3) 開発対象（ソフトウェア）の標準化
(4) 開発者（プログラマ）の技術力向上
(5) 業務専門家（ユーザ）による開発可能化

開発環境は，主に(1)～(3)による生産効率向上と品質向上を支援する．

■ 5・1・2 歴　史 ■

開発環境をツール史の視点からみると次のようにまとめられる．その概要を図5・1に示す．

〔1〕 1950～1960年代

ソフトウェアツールの歴史は，1949年のプログラム内蔵方式のコンピュータの実用化に始まる．「自動化」，すなわち「自動プログラミング」は，その当初からのソフトウェア技術者の夢であった．かつては，アセンブラやコンパイラが自動プログラミング技術と呼ばれた時代もあった．プログラムの記述形式は，機械語からアセンブラ，さらにFORTRAN, COBOLなどの高級言語へと発展し，機械語への展開率の向上という意味での「自動化」を実現したが，プログラムを手続き的に記述するというスタイルは本質的に変わっていない．

この時期には，コンパイラやプログラムのソースファイル操作用のユーティリティなどのツールがバッチ処理形態で使われていた．

〔2〕 1970年代前半

ソフトウェア工学の研究初期の1970年代は，上流工程に注目した要求定義技法や設計技法の研究が盛んに行われ，多くの技法が提案されたが，コンピュータによる支

年　代	特　徴
1950年代 ～ 1960年代	●バッチ型プログラミングツール ・高級言語コンパイラ ・ソースファイル操作ユーティリティ
1970年代 前半	●対話型プログラミングツール ～バッチ型処理から対話型(TSS)処理へパラダイムシフト～ ・上流工程：ドキュメント作成支援 ・下流工程：ツールボックス（エディタ，デバッガなど）
1970年代 後半 ～ 1980年代 前半	●統合プログラミング環境 ～開発手段(ツール)の高機能化から 開発工程(プロセス)の定式化へパラダイムシフト～ ・ユーザインタフェース共通化 ・プログラミング用データベース共通化 ・特定プログラミング言語向き環境(構造エディタなど)
1980年代 後半	●統合開発環境 ～製造工程(how-to-make)中心から 計画・分析・設計工程(what-to-make)中心へパラダイムシフト～ ・CASEツール ・GUIによるユーザインタフェース統合 ・リポジトリによるデータ統合
1990年代	●オープンシステム ～開発工程(プロセス)の定式化から 開発対象(プロダクト)の標準化へパラダイムシフト～ ・開発環境のプラットフォームの共通化 ・分散環境化 ・ミドルウェアの共通化

図5・1 開発環境の変遷とパラダイムシフト

援という観点では，ドキュメント作成支援程度の自動化しか実現できなかった．その理由は，技法の適用プロセスの自動化がむずかしいためと思われる．

そのため，下流工程で利用するプログラミングツールの個別開発が中心に行われた．UNIXに見られるような，いわゆるツールボックス型といわれるものである．この時期には，ツールの利用形態が，バッチ処理形態から対話処理（TSS処理）形態へと移っていった．

〔3〕 1970年代後半～1980年代前半

その後，1970年代に開発されたプログラミングツールの統合化，すなわちプログラミング環境の開発が進んだ．この時期に従来の開発手段（ツール）の高機能化から開発工程（プロセス）の定式化へのパラダイムシフトが生じ，環境という表現が一般化した．当時の統合プログラミング環境は，ユーザインタフェースとプログラミング用データベースの共通化によるツール統合が基本であった．

ツール間の連携は，特定のプログラミング言語を対象にした言語指向プログラミング環境が先行する形で発展した．先駆的なINTERLISPのほか，Adaプログラミング環境APSE，Smalltalk-80などがある．

上流工程に関しては，構造化技法が定着し始めた時期であり，データフロー図やプログラム構造図などの作図やドキュメンテーションが支援されるようになった．同時に構造化図式を対象としたエディタも利用され，ビジュアル化が重視された．

〔4〕 1980年代後半

ソフトウェア開発の真のむずかしさは上流工程にあるという認識から，再び上流工程の方法論や技法のツール化が積極的に試みられ始めた．how-to-makeが主体の製造工程からwhat-to-makeを重視する計画，分析，設計工程へと関心が移っていった．このような上流工程支援を含む統合開発環境を統合CASE（computer aided software engineering）と呼ぶ．

10年前の統合プログラム環境に比べた特徴として，ユーザインタフェースに関しては，ワークステーションの進歩により使い勝手のよいGUI（graphical user interface）が一般化した．データの管理に関しては，リポジトリによるライフサイクル支援がある．

〔5〕 1990年代

最近では，情報処理技術者の不足（または技術不足）とアプリケーションソフトウェアの複雑化の挟み撃ちにあって，CASEへの期待が高まっている．特にオープンシステムの普及とともに，CASEのソフトウェアアーキテクチャをできるだけ階層構造にしておくことにより，個別ツールの追加と拡張，ツール間の連携処理，ユーザインタフェースの統一，データやファイルの共有と統合管理，CASE自身の移植性，ヘテロな分散環境下での共同開発などが容易になってきた．

1980年代に従来の開発手段（ツール）の高機能化から開発工程（プロセス）の定式化へのパラダイムシフトが生じ，環境という表現が一般化したことに対応させると，1990年代には，CASEを含めた基本的なソフトウェアに関して，開発工程（プロセス）の定式化，自動化から開発対象（ソフトウェア）の標準化へのパラダイムシフトが生じ，オープンシステムという表現が一般化したといえる．

5·1·3 現状と展望

情報化社会の進展に対応して，アプリケーション開発のために求められているソフトウェア技術の典型的なものとして以下の3項目がある．

(1) 大規模高信頼ソフトウェアを早くつくって，長く利用する技術

(2) 使い勝手のよいソフトウェアを簡単につくって，どこでも利用できる技術

(3) 機器制御用のソフトウェアをハードウェア技術者が簡単につくれる技術

(1)の大規模高信頼ソフトウェアはグローバル化への対応である．最近のビジネス分野の情報システムや産業分野のCIM（computer integrated manufacturing）の構築に見られるように，広域ネットワークを前提に，従来のアプリケーションを統合し，分散コンピューティングやノウダウン，ノンストップのオンライントランザクション処理を実現する，より複雑で大規模かつ高信頼のソフトウェア開発が必要である．

(2)の使い勝手のよいソフトウェアはパーソナル化への対応である．オフィスにおけるワークステーション，パソコン，ワープロなどの普及に伴って，情報処理の専門家ではない一般の人がエンドユーザとなるため，使い勝手のよいOAソフトやエンドユーザ自身が簡単にプログラムを作成できるツールが必須である．処理対象も数値データからマルチメディアデータへ拡大してくる．さらに分散システムへの移行に対応して，アプリケーションソフトウェアの接続性や移植性が重要になる．

(3)の機器制御用ソフトウェアはインテリジェント化への対応である．工作機械や計測機器から自動車，家電に至るまでマイクロコンピュータが埋め込まれ，熟練技術者のノウハウのプログラム化による知能化が進んでいる．このような分野は多岐にわたるため，ハードウェアとソフトウェアの両方のわかる技術者を確保することは容易ではない．そのため，高性能でコンパクトな機器制御用ソフトウェアをハードウェアの専門家が簡単につくることができる技術が必要とされている．

このような情報化社会のトレンドの中で，分散コンピ

ューティング，オープンシステム化，エンドユーザコンピューティングの進展とともに，アプリケーションソフトウェアのつくり方に関して標準化重視へとパラダイムシフトが起きつつある．それに対応して，開発環境の目標が以下のようにプロセス重視からプロダクト重視へと変わってくる．

(1) エンドユーザの視点では，業務モデルの仕様の形式化を可能とする分野別アプローチをとる．

(2) ツールの視点では，アプリケーションソフトウェアの標準アーキテクチャと部品ライブラリを用意し，業務モデルから部品合成によりソフトウェアを生成する．

(3) 産業形態の視点では，この部品ライブラリを業務の知識に基づく標準パッケージのライブラリとすることにより，知識集約型産業にシフトしていく．

従来のプロセス重視のパラダイムである生産手段（ツール）の高機能化と生産工程（プロセス）の自動化による生産効率の追求だけでは労働者の技術水準が低いという労働集約型産業の弱点を克服できない．プロセスイノベーションよりもプロダクトイノベーションの効果が大きい．効果的な生産物の開発というプロダクト重視のパラダイムにより，生産対象（ソフトウェア）そのものの標準化を目ざす知識集約型産業にシフトしていく．この標準化のパラダイムは地球規模の生産性につながるものである．

5・2 CASE

5・2・1 役 割 と 種 類

CASEツールの目的は，基本的には，ソフトウェアのライフサイクルを通じて，ソフトウェア開発にかかわる種々の側面をコンピュータにより支援することである．CASEは，ソフトウェア開発の真のむずかしさは上流工程にあるというソフトウェア工学の原点の再認識から生まれた用語であるが，適用工程を明確にするために上流工程支援を含む統合開発環境を統合CASEと呼び，上流工程支援主体のものをアッパーCASE(上流CASE)，下流工程支援主体のものをローワーCASE（下流CASE）と呼ぶこともある．

5・2・2 主 要 技 術

統合CASEの主要な技術課題としては以下のようなものがある．

〔1〕 ソフトウェア開発プロセスモデル

ソフトウェアのライフサイクルを支援するためには，まず，そのモデル化が必要である．ソフトウェアのライフサイクルモデルとして次のようなものがある．

(a) ウォータフォールモデル（滝モデル） これは一般の工業生産物と同じように，要求分析，設計，製造（コーディング），検査(テスト)，運用の順にトップダウンに開発していくものである（1章図1・1参照）．このモデルの欠点は，ソフトウェア開発の最初の段階でつくり込んだ誤りが最後の段階にならないと検出できないということである（3章図3・3参照）．

(b) プロトタイピングモデル これは，ウォータフォールモデルの欠点を回避するために，最終的な製造に入る前に，要求定義や設計上の特に重要な部分についてプロトタイププログラムをつくって検証する方法である．たとえば，開発の初期段階でユーザインタフェースの確認や応答性などの性能の確認を行うことにより，最終段階での仕様の変更などによる大幅な手戻りを防ぐことができる．実際の適用では，プロトタイププログラムを使い捨てにする場合やそれを核として拡張していく場合などいろいろである．

(c) 操作的アプローチ これも，ウォータフォールモデルの欠点を回避する方法で，要求分析や概略設計の段階で実行可能な仕様を記述し，それを実行することにより，詳細設計やコーディングの前に仕様を検証する．実行は通常インタプリタによるシミュレーション方式で行う．

(d) プロセスプログラミング さらに，このような開発プロセスそのものを「プログラム化」して，洗練していく方法も研究されている．

〔2〕 開 発 方 法 論

定式化された個々の工程で用いる道具の高級化だけでは単なるツールボックスである．統合開発環境を構築するためには，特に工程間の関連づけが重要であり，具体的にどのようにソフトウェアを開発していくかという方法論の確立が必要である．

方法論は，ソフトウェアの開発を情報システムの開発という観点で広くとらえると，情報システム構築技法を含む．この場合は，情報技術と経営戦略の関係にまで及ぶ．1990年ごろの戦略的情報システムSIS（strategic information system）では，「経営戦略，即，情報」という観点から情報システムを経営戦略にどう生かすかというアプローチがみられ，成功例は必ずしも多くなかった．最近のビジネスプロセスリエンジニアリングでは，「顧客の視点から業務プロセスの抜本的見直し」という観点で業務革新に情報技術をどう生かすかというアプローチがとられている．

ここでは，要求分析段階での業務モデルの表現方法として，多くのCASEツールで支援している代表的な図式モデルをあげておく．

(a) データフローモデル データフローモデルの表現に用いられるデータフロー図は，図5・2に示すよう

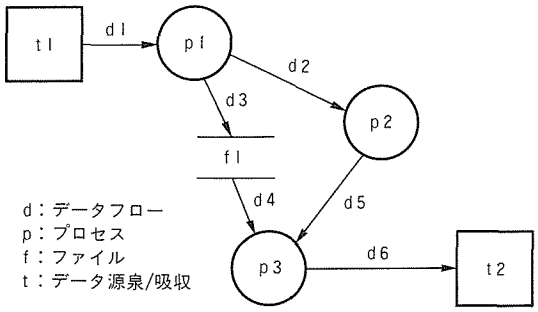

図5・2 データフロー図

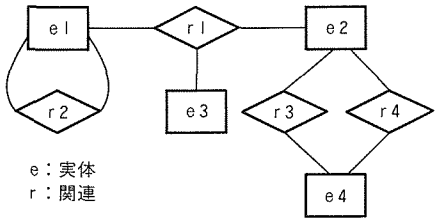

図5・3 実体関連図

に4種類の基本要素から構成される．まずデータの処理の単位となるプロセスは円，データの流れは矢印，ファイルやデータベースのようなデータ蓄積部分は平行な2本の線分，データの源泉と吸収部分は矩形で表す．複雑なシステムを表現するときは一つのプロセスをさらに詳細なデータフロー図で階層的に表現できる．

1970年代後半に開発された構造化分析技法（SA；structured analysis）では，このデータモデルを用いて業務モデルを要求仕様として記述する．最近のオブジェクト指向分析，設計技法でもオブジェクトの機能仕様の記述にこのデータフロー図が用いられている．

(b) 実体関連モデル　これはデータモデルとして開発されたもので，E-Rモデル（entity-relationship model）とも呼ばれ，図5·3に示すようなER図を用いる．実体を矩形で表し，実体間の関連を菱形で表す．

1970年代中頃の先駆的な要求定義支援システムPSL/PSA（米国ミシガン大学）では，ERモデルに基づく要求仕様定義言語が開発されている．最近のオブジェクト指向分析，設計技法でもオブジェクト間の関係の定義にERモデルの拡張モデルが用いられている．

(c) 有限状態機械モデル　このモデルでは状態遷移図が用いられる．図5·4に示すように状態をノードで表し，ノード間の状態遷移をアークで表す．アークに状態遷移条件を記述する．

このモデルは制御用実時間システムや電子交換システムなどで用いられている．最近のオブジェクト指向分析，設計技法でもオブジェクトのふるまいの記述にこの状態遷移モデルが用いられている．

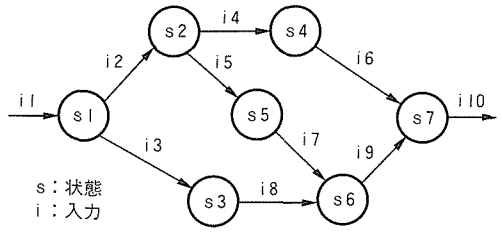

図5・4 状態遷移図

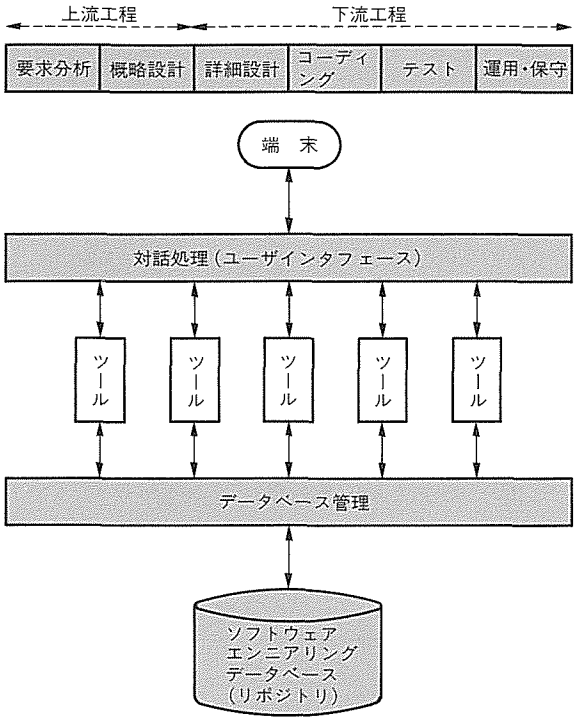

図5・5 開発環境の基本構成

(d) オブジェクトモデル　現在，種々のオブジェクト指向分析・設計技法が提案されており，それぞれ用いる図の記法が異なるが，一般に複数の記法を組み合わせて用いる場合が多い．よく用いられるものは，すでに上に述べたER図，データフロー図，状態遷移図，およびその拡張したものである．クラス階層図（is-a関係，kind-of関係）やインスタンスの包含関係（has-a関係，part-of関係）などのオブジェクト関連図はER図を拡張したものが利用される．

〔3〕 アーキテクチャ

ツールボックスから統合プログラミング環境，さらに統合CASEへと開発環境が発展してきたが，そのフレームワークを与えるものとしてソフトウェアアーキテクチャがある．ツール統合化の一般的なアーキテクチャは，図5·5に示すようにユーザインタフェースをつかさどる

対話処理部分をフロントエンドプロセッサとして，また設計仕様やプログラムなどの生産物を管理するデータベース管理をバックエンドプロセッサとしてツール間で共通にすることである．

一般に，統合化には次のような観点がある．

(1) データ統合：開発環境で用いられるあらゆる情報が統一的に管理され，ツール間で利用される．

(2) 制御統合：開発環境の中で，ツールが提供する機能（サービス）がプロセス管理やプロジェクト管理に対応して柔軟に組み合わせてツール間で利用される．

(3) 表示統合：ユーザインタフェースをツール間で一様にすることより，使い勝手をよくする．

(4) プロセス統合：定義されたプロセスに従ってツールが利用される．あるプロセスの実行結果に依存して次のプロセスが選択される．

統合CASEでは，さらにこの傾向が進み，標準化の促進により，個別ツールの追加と拡張，ツール間の連携処理，ユーザインタフェースの統一，データやファイルの共有と統合管理，CASE自身の移植性，ヘテロな分散環境下での共同開発などの処理を容易にしている．

〔4〕 情 報 管 理

仕様書やプログラムなどの生産物に関する情報，それらの管理情報および生産技術に関する情報などの統合管理のためには，図5･5に示すように，それらの情報を保管するSEDB（software engineering database）またはリポジトリ，およびそれらの情報と各種ツールとの仲介をするデータベース管理システムの構築が必要である．特に，このような情報管理は，貴重なソフトウェア財産を有効活用するためのリエンジニアリング技術にも必須である．

情報管理のための代表的なデータモデルとして，ERモデル，オブジェクトモデル，ハイパーテキスト/リンクベースモデルなどがある．ERモデルおよびオブジェクトモデルは，前記〔2〕の要求仕様のところで説明したが，これらのモデルはリポジトリのモデルとしても一般的である．さらに，ソフトウェア開発時に種々の情報を利用する場合，各データを個別に参照するほかに，関連するいくつかの情報を同時に見たいことがある．たとえば，データフロー図を見ながらあるプロセスに対応するモジュール仕様書とその入力データのデータ構造仕様書を見るなどである．このように，情報を渡り歩いていく場合やシステム側が積極的にそれをナビゲートする場合に適したモデルとして，ハイパーテキストシステムあるいはリンクベースシステムといわれるものがある．

5･3 標 準 化

5･3･1 概 要

生産性向上の手段の変遷をおおまかにまとめると，まずツールの高級化に始まり，次に工程の定式化（プロセスの自動化）を行い，現在は対象の標準化が活発である．

もちろん標準化の基本は共通化による共有化であり，古くて新しい問題である．これまでソフトウェアの標準化には次のようないくつかの段階があった．

(1) 同一組織内での部品化，再利用

(2) 同一組織内，複数機種間での共通化

(3) 複数組織間，複数機種間での共通化

(4) アプリケーションパッケージの業界標準化

このうちの(1)は，従来からプログラムの部品化，再利用技術として研究がなされてきたものである．知識工学を応用したものやオブジェクト指向概念をベースにしたものがある．次の(2)は，アプリケーションソフトウェアの異機種間の移行性を高めるために，アプリケーションソフトウェアの標準的なアーキテクチャを設定し，特定メーカのスーパーコンや大型機からワークステーションやパソコンまでのどの機種ででも同じアプリケーションソフトウェアが実行できることをねらっている．そのために，プログラミング言語，ユーザインタフェース，通信管理などの外部仕様の統一を計っている．このアプローチは1980年代後半にメーカ側から提案されたものであるが，次に述べるその後の急激なオープンシステム化の動きの中でアプリケーションの移行性よりも異機種相互接続性に重点が移っている．

次に，(3)はオープンシステムといわれるもので，アプリケーションの移行性だけでなく，ネットワーク上の異機種間の接続性も実現している．オープンシステムについては次項で詳細に述べる．最後の(4)は，特定用途のアプリケーションをパッケージ化して複数のユーザに利用してもらうやり方である．これらの中からベストセラーになったものが業界標準になっていく．OAソフト，RDBソフト，4GLなどでその傾向が強い．銀行システムではメガステップオーダのバッケージもある．

開発環境に関しては，以下に示す二つの標準化の観点がある．

(1) 標準化動向に対応したソフトウェアを開発する．

(2) 開発環境自身が標準化される．

いずれにせよ，類似のソフトウェアが数多く開発され，おのおのが限られた数のユーザに利用されるという人的資源のむだ使いの時代は終わり，質の高い標準品を多数のユーザが利用するという地球規模のソフトウェア生産性を実現する時代になっている．

5·3·2 オープンシステム

現在は，次のような特徴に代表されるオープンシステムの時代であり，開発環境も例外ではない．

(1) 異機種/異種プラットフォーム間でのアプリケーションの移行性

(2) 異機種/異種プラットフォーム間の接続性/相互運用性

1990年代のシステム構成は，大まかには，図5·6に示すように，応用ソフトウェア，ミドルウェア，基本ソフトウェア，ハードウェアの4階層で表現される．上位の応用ソフトウェアと下位のハードウェアの種類は豊富になり，中間のミドルウェアと基本ソフトウェアの選択が限定された形になる．ミドルウェアや基本ソフトウェアとアプリケーションとの間のAPI (application programming interface) をできるだけ業界標準や国際標準にして，異なるメーカの機種間でもアプリケーションソフトウェアの移行性を確保したり，相互接続を可能とするものである．特にミドルウェアの役割は大きく，プラットフォームと呼ばれることもある．

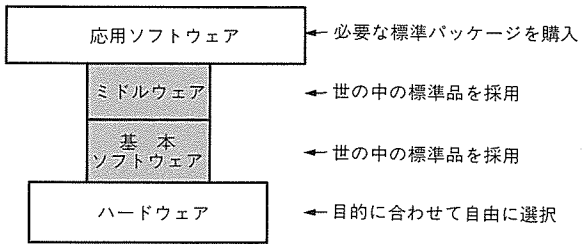

図5·6 オープンシステムの基本構成

長い標準化の歴史を有するプログラミング言語のほかに，通信手順としてOSI (open systems interconnection) などが国際標準化されている．また，UNIX, GUI (graphical user interface)，オブジェクト指向データベース，オブジェクト管理などに関して国際標準化，業界標準化の動きが活発である．

開発環境に関しては，以下のような標準化関連事項がある．

〔1〕 移　行　性

(1) 異機種/異種プラットフォーム間でのCASEツール自身の移行性

(2) 同機種/同種プラットフォーム間での異種CASEツールが扱う情報の互換性

(3) 異機種/異種プラットフォーム間での同種CASEツールが扱う情報の互換性

〔2〕 分散環境下の相互運用性

(1) 異機種/異種プラットフォーム上の同種CASEツール間の相互運用性およびCASEツールが扱う情報の互換性あるいはリポジトリ共有

(2) 同機種/同種プラットフォーム上の異種CASEツール間の相互運用性およびCASEツールが扱う情報の互換性あるいはリポジトリ共有

(3) 異機種/異種プラットフォーム上の異種CASEツール間の相互運用性およびCASEが扱う情報の互換性あるいはリポジトリ共有

ユーザ視点でみれば，CASEツールを用いて開発および保守を行い，さらにリエンジニアリングの対象にもなるアプリケーションが，環境の変化に対応しながら長く有効活用されていくためには，前記〔2〕の(3)の実現が望ましい．常に時代の最先端の生産技術を用いてアプリケーションの開発保守を行うことができるからである．

これは，同時にCASEツールやプラットフォームあるいはハードウェアの開発者にとっても，いいものをつくれば必ず普及することが保証され，技術の進歩が促進される．

5·3·3 開発環境に関する標準化

このようなCASEツールに関連した移行性，互換性，相互運用性の観点で，そのプラットフォームおよび情報に関する標準化活動の具体例について述べる．

〔1〕 PCTE

1980年代前半に，ヨーロッパを中心にデータ統合とツールの移植性を可能とする，CASEツールのプラットフォームあるいはフレームワークとしてPCTE (portable common tool environment) が検討され，現在，欧州電算機製造業者協会ECMA (European Computer Manufacturers Association) の標準案としてISO (International Organization for Standardization) に提案されている．

PCTEの特徴として，オブジェクト管理システムがある．PCTEではデータ結合の実現のためにERモデルに基づくオブジェクトベースを基本にしている．このオブジェクトベースはオブジェクトおよびオブジェクト間のリンクからなり，これらのオブジェクトおよびリンクは属性をもつことができ，オブジェクト間の関係はリンクの対で表現される．

異機種ネットワーク上に分散したリポジトリを統合管理するようなネットワーク透過性を支援しており，分散開発環境に対応している．

〔2〕 トースタモデル

CASE環境のフレームワークを与える参照モデルとして，図5·7に示すようなトースタモデルをECMAと米国商務省の標準技術局NIST (National Institute of Standards and Technology) が制定している．この参照モデルは標準化やオープン性の観点で種々のCASE環境をお互いに比較する基準となりうるものである．

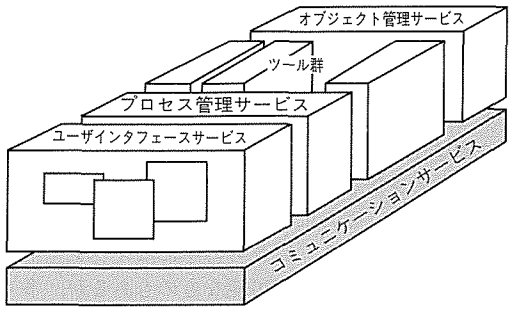

図5・7 トースタモデル

図5・7において，オブジェクト管理サービスがデータ統合を実現し，プロセス管理サービスとコミュニケーションサービスが制御統合を実現し，ユーザインタフェースサービスが表示統合を実現する．ツール群は必要に応じて追加されたり，差し替えたりできる．

〔3〕 CORBA

オブジェクト指向技術の標準化団体OMG（Object Management Group）は，ソフトウェアの相互運用性を図るために，オブジェクト管理プラットフォームの参照モデルとしてCORBA（common object request broker architechture）を制定している．その概要を図5・8に示す．アプリケーションに固有の仕事をするオブジェクトは，通信用のインフラであるORB（object request broker）を介して一般性のあるアプリケーション間共通の機能を果たすオブジェクトと低レベルのオブジェクトサービスを依頼する．

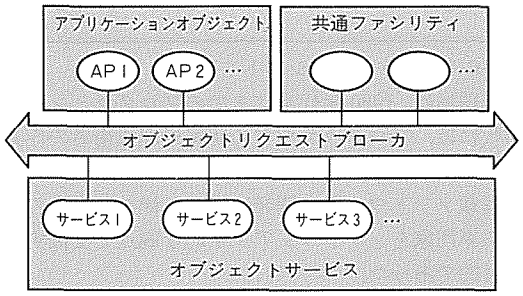

図5・8 CORBA（OMG）の概要

一方，アプリケーションの移行性と相互運用性を実現する共通オープンソフトウェア環境を開発するための業界標準制定活動として，米国のUNIX関連企業が中心になって進めているCOSE（common open software environment）プロセスがある．ここでは，共通デスクトップ環境，ネットワーキング・分散システム，オブジェクト指向技術などの共通化を進めており，オブジェクト指向技術では，このCORBAを採用している．

〔4〕 CDIF

米国電子産業協会（EIA；Electronic Industries Association）では，CASEツール間のデータ交換の標準形式としてCDIF（CASE data interchange format）の暫定規格を1991年に制定しており，さらに検討中である．その主目的は，ソフトウェア開発にかかわる情報を異種CASE間で交換可能とすることにより，ソフトウェアのライフサイクル管理を容易にしたり，分散開発を容易にする．すなわち，ソフトウェアの開発，保守が特定のCASEツールに束縛されないことにより，ユーザとベンダーの双方に利益をもたらす．

データフロー図やER図を正確に伝えるために，意味情報と表示情報を分けて送る．このCDIFは，現在，ECMAおよびISOにおいて標準化の一候補として検討されている．

〔5〕 そ の 他

前記のほかにも種々の標準化活動がある．CDIFと同じくCASEツール間でのデータ交換を目的としたものに，IEEEのCASEツール相互接続参照モデルとデータ転送用言語STL（semantic transfer language）がある．STLは，上流工程で使われる仕様情報に関して，アクション，データなどの概念情報，その部分集合に対応する視点情報，表現形式に関する表現情報などを規定している．

情報共有の観点でリポジトリの標準化に関連するものに，情報資源辞書システムIRDS（information resource dictionary system）がある．これは，ANSIで検討が始められ，後にISOでも取り上げられている．型とインスタンスの概念を拡張してデータレベルを規定している．

5・4 支 援 機 能

開発環境が提供する個々の機能については，本書のほかの章で詳細に説明がなされているので，ここでは機能項目をあげることにとどめる．

〔1〕 分析，設計支援機能

(1) 仕様書作図・編集・印刷
(2) 仕様書図表間関連管理とナビゲーション
(3) 仕様記述言語の入力・編集・検証
(4) 仕様の実行・検証

〔2〕 プログラミング支援機能

(1) 仕様からのプログラム自動生成
(2) プログラム変換
(3) 部品合成
(4) エディタ
(5) コンパイラ
(6) デバッガ

(7) インタプリタ
(8) プログラム管理
〔3〕 検査，検証支援機能
(1) テスト項目作成
(2) テストデータ生成
(3) テスト手続き
(4) デスト網羅率測定
(5) プロフィール解析
〔4〕 保守・リエンジニアリング支援機能
(1) リバースエンジニアリング
(2) 再構造化
(3) 影響範囲解析
〔5〕 ライフサイクル支援機能
(1) プロジェクト管理
(2) プロセス管理
(3) 進捗管理
(4) 構成管理
(5) 品質管理
(6) ライブラリ管理
(7) 文書管理

5・5 分散環境

5・5・1 分散実行環境

パソコン，ワークステーションの普及とコンピュータネットワークの普及により，情報システムといえば分散システムが普通になっている．したがって，開発環境の観点では，開発の対象が分散システムであるという面と開発環境自身が分散システムであるという面がある．

前者は後者を含むので，まず分散システムの標準化動向を述べておく．ユーザ視点では，異機種ネットワーク上でのアプリケーションシステムの相互運用性を確保するためには，先に図5・6に示したミドルウェアのレベルで実現するのがわかりやすい．その一例として，UNIXの標準化団体OSF（Open Software Foundation, Inc.）が開発中の分散コンピューティング環境DCE（distributed computing environment）がある．DCEは，クライアント/サーバモデル構築のためのリモートプロシジャコール機能，ディレクトリ，タイム，スレッド，セキュリティなどの基本サービスや，エンドユーザが直接利用可能なデータ共用サービスを提供する．

5・5・2 分散開発環境

ソフトウェアの分散開発の形態として，開発管理の観点では次のようなレベルが考えられる．

(1) 分散環境下でのデータの受け渡し
(2) リポジトリの共有ないしリポジトリの分散管理
(3) 協調作業（グループウェア）

これらのレベルは地域的な分散の範囲がLANの範囲で，開発者間の直接的なコミュニケーションが簡単にできるか，あるいはもっと離れているWANの範囲かということとも関係する．

先に標準化のところで述べた種々の活動は，このような分散開発の支援を目的としており，データの互換性，ツールの移行性，さらにはネットワーク透過性や相互運用性を実現しつつある．これらの中にはオペレーティングシステムのレベルで実現可能なものもあるが，多くはミドルウェアのレベルで実現される．

特に今後はCSCW（computer-supported cooperative work）の観点でグループウェアが重要である．分散開発の観点では，電子会議，電子メール，電子掲示板などのコミュニケーション機能やバグ票管理などのワークフロー支援機能のように，距離と時間のギャップを克服する機能があればよい．さらに，グループウェアの観点では，たとえ共同開発者がそばにいても，そのシステムを利用することにより仕事の質が向上するような機能が必要である．たとえば一例として，米国のMCCで開発されたハイパーテキストツールgIBIS（graphical issue-based information system）がある．これは，複雑なシステムの共同設計時において，途中で生じた個々の問題を議論しながら解決していくプロセスの重要性に着目し，それをハイパーテキストとして保存，管理する処理をGUIツールで支援する．たとえば，問題およびそれに対する見解，論拠などをリンクをはって保存する．

5・6 エンドユーザコンピューティング

5・6・1 パラダイムシフト

開発環境は，1980年代半ばまでのプログラミング環境の時代には，プログラマなどの情報処理技術者の作業効率の向上が主目的であった．その後のCASEでは，大規模高信頼ソフトウェアの開発に携わる種々の立場の情報処理の専門家をソフトウェア工学的な観点で支援するのが主目的であった．

しかし，高度情報化社会を迎えた現在では，情報システムは，すでに業務の効率化といった従来の目的を越えて，社会生活の中に入り込んできている．そして，そのような情報システムの利用に関して，従来の情報システム部門に対比したエンドユーザ部門という範囲を越えて，あらゆる人達がエンドユーザになりつつある．このような状況は以下のような理由によるところが大きく，今後は，エンドユーザコンピューティングを支援する環境が重要になろう．

(1) ハードウェアの低廉化とともに，ブック型コンピュータ，パソコン，ワークステーションなどの普及

がめざましく，OA分野を中心にエンドユーザが急増している．「物のパーソナル化」が「情報のパーソナル化」を促進している．

(2) コンピュータの利用形態が大型機によるホスト集中型からパソコン，ワークステーション主体のネットワーク型へと移行するとともに，分散コンピューティングの普及がめざましく，エンドユーザがコンピュータを直接操作する機会が急増している．「システムのグローバル化」が「情報のグローバル化」を促進している．

(3) コンピュータの利用目的が，従来の業務の合理化から経営戦略の実現へと広がるにつれて，非定形業務が増加し，さまざまな業務担当者がエンドユーザとしてコンピュータを利用し始めている．

オフィスにおけるエンドユーザの業務を大ざっぱに分類すると，従来は定形業務と非定形業務に分けられた．定形業務はビジネス分野の帳票処理に代表されるような企業の基幹業務にかかわるものである．そのためのアプリケーションプログラムは，通常は，エンドユーザの要求をききながら，企業内の情報システム部門が開発あるいは導入して，エンドユーザに提供してきた．保守，拡張も情報システム部門が行い，エンドユーザは定められたユーザインタフェースに従ってコンピュータを使用するだけであった．一方，非定形業務については，表計算に代表されるような市販のOAソフトウェアを購入して利用してきた．

しかし，コンピュータの利用目的が，従来の業務の合理化から経営戦略の具現化へと広がるにつれて，基幹業務と個人の非定形業務の連携が必要になってきている．この傾向は，コンピュータシステムが分散コンピューティングの方向に進展するにつれていっそう加速されている．その結果，従来の表計算ソフトウェアと基幹業務データベースをつないだり，データベース検索ソフトウェアにデータの加工，編集機能を追加したりして，意思決定支援システムをつくり上げることが重要になっている．さらに，CSCWのように，別々の場所にいるグループが協力しあって意思決定をしていくためのグループウェアが求められている．

このような状況下でタイムリーに必要なソフトウェアを手にいれていくためには，もはや情報システム部門が提供するシステムや市販のソフトウェアパッケージをたんに利用するだけでなく，業務担当者は，積極的に自ら使うソフトウェアの開発に関与していく必要がある．そのために，開発環境は，従来の手続き型パラダイムに基づくプログラミングの支援ではなく，脱プログラミングを可能とする新しいパラダイムを提示する必要がある．

5·6·2 ソフトウェア産業論

エンドユーザコンピューティングは，ソフトウェア産業の発展過程でとらえるならば，労働集約型産業，知識集約型産業に続くものである．

〔1〕 労働集約型産業

これまでのソフトウェア産業は労働集約型産業であり，生産コストあたりの生産量という生産性の効率向上が重要であった．ステップ数/人月あるいはステップ単価を改善するために，CASEツールなどを用いて自動化率を向上させ，人海戦術からの脱皮の努力をしてきた．しかし，この生産性の尺度の致命的欠陥は，ソフトウェアの価値（質）を規模（量）で計ることである．ソフトウェア産業の未熟さは，ソフトウェアの価値をステップ単価や工数（人月）でしか計算できないというソフトウェアメトリックスの未熟さに起因している．そのため，ソフトウェアの受託開発において，生産コストを発注者がすべて支払う場合は，ステップ数が多いほど価格(価値)が上がることになってしまい，受注者側の技術力向上の努力がおろそかになってしまう．このような矛盾は，バブル経済崩壊とともにソフトウェア産業が不況業種の代表になってしまったことによく現れている．

〔2〕 知識集約型産業

ソフトウェアの生産性は本来以下のように定義されるべきである．

ソフトウェアの生産性＝生産物の価値/生産コスト

この「生産物の価値」はユーザの視点で決まるべきものであり，高品質のソフトやベストセラーのアプリケーションパッケージの価値は高い．このようなアプリケーションパッケージの開発には，業務の知識と情報処理技術の双方が必要である．

いま，大まかに現状を把握するために，生産物の価値の代わりに売上高に着目し，生産コストの代わりにその大部分を占める人件費に対応する従業員数を用いると，従業員1人当りの売上高という指標が得られる．この指標で過去10年間を振り返ると，売上高が約7倍に対し，わずかに2倍弱程度である．売上の伸びが従業員の伸びで支えられているという労働集約型産業の実態がよくわかる．

今後，ソフトウェアのビジネスは，アプリケーションパッケージの開発とシステムインテグレーションの方向に発展していくと思われるが，この場合，種々の分野対応の業務の知識と情報処理の知識をノウハウとして蓄積することが必要である．

〔3〕 ポスト知識集約型産業

情報処理システムが，業務の効率化ではなく，経営戦略の具現化に用いられるようになると，ソフトウェアの開発においても効率よりも効果が重要視される．何をつ

くるかが最も大事であり，それを決めるのは業務専門家である．業務専門家が知恵をしぼって意思決定支援などの非定形業務用ソフトウェアをタイムリーにつくっていくためには，エンドユーザ自身が開発でき，かつ保守拡張ができる必要がある．

このようなエンドユーザコンピューティングを実現するためには，そのためのツールや環境を提供しなければならない．自動化ツールや標準パッケージなどの情報処理技術を統合したうえに，きめ細かく分類された応用分野対応（domain-specific, application-oriented）の業務の知識に基づいたアプリケーションフレームワークの構築が必須である．

潜在ユーザとしての業務の専門家の数を考えると，市場規模は現在の情報サービス産業のそれ（数兆円規模）を大きく上回ることになる．ポスト知識集約型産業は，知恵集約型産業ともいえる．

5·6·3 主 要 技 術

〔1〕 エンドユーザコンピューティングツール

エンドユーザコンピューティングを指向したツールで基本となるのはエンドユーザ言語である．その要件は，以下の3種類の観点でみると，右側に示した特徴を有するものである．

(1) 手続き型言語 vs. 非手続き型言語
(2) コンピュータ言語 vs. 対象世界記述言語
(3) システム記述言語 vs. 応用記述言語

まず，(1)はパラダイムによる分類である．現在広く普及している手続き型のパラダイムは，対象とする問題の記述には適切でないことが多い．このギャップを埋めるためにプログラムを非手続き的あるいは宣言的に記述する言語でなければならない．

次に，(2)は言語の役割による分類である．言語は本来的にコミュニケーションの道具であるが，コンピュータ言語の場合は，コミュニケーションの相手がコンピュータであるため，これまで従来のコンピュータアーキテクチャと相性のよい手続き型言語が用いられてきた．これに対し，対象世界記述言語は，対象世界を抽象的に表現するモデリング技術に基づいた問題記述言語でなければならない．

最後に，(3)は言語の用途，すなわち記述対象による分類である．システム記述言語は，OS，データベースなどの基本ソフトウェアの開発に使用するため，実行効率が重視され，アセンブラや手続き型言語が用いられてきた．これに対し，応用記述言語はアプリケーション対応に特定分野/業務向きの言語でなければならない．

これまでエンドユーザ言語として，データベース検索や表計算のアプリケーションの中でマクロ言語やスクリプト言語が用いられ始めているが，前に述べた条件を満たしたものではなく，プログラミングの概念は必要である．

以下では，エンドユーザが業務のコンピュータ化を行えるための現実的な技術の例として，① 第4世代言語，② 日本語プログラミング，③ ビジュアルプログラミング，④ 人工知能技術，について簡単に説明する．

〔2〕 第4世代言語

エンドユーザコンピューティングを指向した業務向け簡易言語として第4世代言語（4 GL）がある．第4世代言語という名前は，第1世代言語(機械語)，第2世代言語（アセンブラ），第3世代言語（コンパイラ言語）に続く言語という意味で付けられた．ジェームス・マーチン（James Martin）は，4 GL を13か条の特徴で定義したが，その主なものは，非職業的プログラマが使用可能であること，アプリケーションプログラムの記述ステップ数と作成工数がCOBOLより1桁少ないこと，非手続き的記述形式の採用，などである．

4 GL の普及は米国が進んでいるが，現在，日本では数十種類の4 GL が流通していると思われる．4 GL の現状調査によると，4 GL ユーザの98％がエンドユーザだけでのソフトウェア開発可能性を否定している．その理由は，現在の4 GL が，COBOL と比較した展開率の向上という「量的高級化」アプローチに基づいて設計されており，プログラミング技術を必要としているためと思われる．

今後，4 GL のような業務向け簡易言語をエンドユーザコンピューティングのツールとしていくためには，業務の言葉で要求仕様を記述でき，業務知識のデータベースを用いてプログラムを自動生成できるようにする必要がある．

〔3〕 日本語プログラミング

職業的プログラマでないエンドユーザにとって，プログラムの日本語表現は書きやすさ，読みやすさの点で魅力的である．特にアプリケーションプログラムの保守性に不可欠であるプログラムの理解容易性の向上には，プログラム構造がきれいであることとともに，プログラムの各文の意味がわかりやすいことが重要である．

プログラムの日本語表現には次のようなレベルがある．

(1) 既存のプログラミング言語でデータ名や手続き名に日本語を使う．
(2) 既存のプログラミング言語の if，end などのキーワードも含め，すべて日本語で記述する．
(3) 既存のプログラミング言語相当の記述レベルで独自の文法規則をもつ日本語プログラミング言語で記述する．
(4) 仕様記述言語として日本語を導入し，プログラム

を自動生成する．

現在の実用レベルは(2), (3)が主で，4 GL に組み込まれているものが多い．今後は(4)が主流になっていくと思われるが，次のような技術課題がある．

(1) 業務用語の使用に対応した業務用語辞書作成保守機能

(2) 日本語記述レベルですべて処理するための開発保守環境

(3) 日本語表現のあいまいさを回避する文法規則

日本語化が進んでいる分野として，データベース検索のコマンドインタフェースがある．たとえば，あるデータベース検索用自然語インタフェースでは

「箱根の宿泊先を費用の安い順に出せ」

という問合せ文を入力すると，システムが

select 宿泊先テーブル. 宿泊先 from 宿泊先テーブル
where 宿泊先テーブル. 所在地＝'箱根'
order by 宿泊先テーブル. 費用 desc

という SQL 文に変換して実行してくれる．

日本語プログラミングのニーズが大きいもう一つの分野は，第3次銀行オンラインシステムのようなメガステップオーダの大規模高信頼ソフトウェアの開発である．これまでは，ユーザの情報システム部門，ソフトウェアハウス，メーカの SE 部門が協力して，COBOL や PL/I などのプログラミング言語を用いて開発してきたが，これまで以上の大規模なシステムは，以下のような理由で従来方式での開発は無理である．

(1) 計画段階では数年先の稼働時の機能設計は不可能

(2) 開発管理が規模的に不可能

(3) 開発中や稼働後の機能変更への迅速な対応が不可能

これに対し，日本語プログラミング技術を適用すれば，プログラムの仕様が業務仕様書レベルになり，エンドユーザがプログラムを開発したり，保守できるので，前記の問題は軽くなる．

〔4〕 ビジュアルプログラミング

従来のプログラミング言語のむずかしさを回避する方法として，日本語プログラミングでは記述の形式性を緩和することによって記述容易性と理解容易性を実現しているが，ビジュアルプログラミングでは記述の線形性の緩和によってそれを実現する．これまで次の三つの観点からプログラムの視覚化の研究が活発に行われてきた．

(1) 視覚的ユーザインタフェース

(2) 視覚的編集

(3) 視覚的言語

(1)の視覚的ユーザインタフェースは，プログラミング環境を用いてプログラムを開発するときに，システム側から提供される種々の情報をグラフィカルに表示し，ユーザ側からの入力もそのグラフィカルな画面上で行えるようにするものである．

(2)の視覚的編集は，従来のテキストエディタに代わり，プログラミング言語の文法規則の基本構造をテンプレートとして表示し，文法的に正しいプログラムを誘導するものである．構造エディタや構造化図式を用いた図式エディタがすでに使われている．

(3)の視覚的言語は，本来的に視覚的表現を基本とした言語であり，その視覚化に用いる図式の表現形式の違いにより，フォーム指向言語，グラフ指向言語，アイコン指向言語に分類される．

フォーム指向言語は，表形式を基本とするもので，OA 分野のスプレッドシートを用いたフォームベースプログラミングが代表的である．グラフ指向言語は，ノードとアークで構成される有向グラフ表現を基本とするもので，データフロー図，状態遷移図などがある．アイコン指向言語は，ビジュアルオブジェクト（アイコン）とその関係表現を基本とするものである．基本的な処理単位をアイコンとして用意しておき，画面上でアイコンとアイコンを結ぶことによってプログラムを作成していく．

以上のビジュアルプログラミング技術のうち，視覚的ユーザインタフェースと視覚的編集はプログラミング言語の概念から開放されていないので，エンドユーザコンピューティングという観点では，不十分であろう．視覚的言語に関しては，プログラミング言語の概念は不要であるが，プログラミングの概念は必要である．

〔5〕 人工知能言語

知識ベースシステムやエキスパートシステムが種々の分野で実用になっているが，その理由は，“推論機能”というよりはむしろ“簡易プログラミング”としての魅力にある．職業的プログラマでない業務専門家が業務の言葉で表現できる利点が大きい．さらに，最もよく用いられている知識表現であるプロダクションルールの場合は記述形式が「もし～ならば，～せよ」という一つのパターンに限定されており，かつ基本的には記述順序が自由であるので，業務知識の記述，追加，修正が容易である．

従来の手続き型プログラムが，N. Wirth の図式で

プログラム＝アルゴリズム＋データ

と表現されたのに対応させると，知識ベースシステムでは

知識ベースシステム＝推論エンジン＋知識ベース

となるが，アルゴリズムに対応する推論エンジンはすでに用意されている．したがって，ユーザは，データに対応する知識だけを記述すればよい．

分散コンピューティングの世界では，グループウェアを経由して，このようなルールベースシステムがエージェントとしてお互いにメッセージをやりとりして分散協

調システムを形成することになる．このときの一つの課題は，各エージェントにどのように適応機能をもたせることによってシステム全体の適応機能を実現するかということである．そのためには，エージェントにマルチパラダイムの知識表現のみならず，メタ知識やリフレクション機能が必要となろう．（中所　武司）

参考文献

[青山, ほか 1992] 青山, 坂下（編）："特集：分散開発環境", 情報処理, Vol. 33, No. 1 (1992)

[鯵坂 1993] 鯵坂恒夫（編）："特集：ソフトウェア生産環境", コンピュータソフトウェア, Vol. 10, No. 2 (1993)

[大筆, ほか 1990] 大筆, 川越（編）："特集：CASE 環境", 情報処理, Vol. 31, No. 8 (1990)

[古宮 1990] 古宮誠一："事務処理ソフトウェア開発用簡易言語（第4世代言語）の現状と分析", 情報処理, Vol. 31, No. 9, pp. 1257-1269 (1990)

[竹下 1990] 竹下亨：CASE 概説, 共立出版 (1990)

[中所 1992] 中所武司：ソフトウェア危機とプログラミングパラダイム, 啓学出版 (1992)

[中所 1983] 中所武司："プログラミング言語とその会話型支援環境", 情報処理, Vol. 24, No. 6, pp. 715-721 (1983)

[野木, ほか 1989] 野木, 中所：プログラミングツール, 昭晃堂 (1989)

[Ambler, et al. 1989] Ambler, A. L. and Burnett, M. M.: "Influence of Visual Technology on the Evolution of Language Environments", IEEE Computer, Vol. 22, No. 10, pp. 9-22 (1989)

[Chang 1987] Chang, J.: "Visual Languages: A Tutorial and Survey", IEEE Software, Vol. 4, No. 1, pp. 29-39 (1987)

[JIPDEC 1993] 日本情報処理開発協会(編)："情報化白書 1991", コンピュータ・エージ社 (1993)

[JISA 1993] 情報サービス産業協会（編）："情報サービス産業白書 1993", コンピュータ・エージ社 (1993)

[Martin 1985] Martin, J.: Fourth Generation Languages, Prentice-Hall (1985)

[Norman, et al. 1992] Norman, R. J. and Chen, M. C. (Ed.): "Special Issue on Integrated CASE", IEEE Software, Vol. 9, No. 2 (1992)