解　説

エンドユーザコンピューティング
——ソフトウェア危機回避のシナリオ†——

中　所　武　司††

1. はじめに

迫りくる新しい世紀に向かって激動する今の時代を象徴するキーワードとして，グローバル化とパーソナル化をあげることができる．

グローバル化は，経済活動の 24 時間化と現地化という意味を超えて，あらゆる分野で進行している．東からは日米構造協議による日本的経済機構の排除，西からは欧州統合と東欧諸国の急激な社会変革，南からは経済難民や不法就労者の流入による国境の無意味化，という形でのグローバル化の波が押し寄せている．これらはすべて社会システムのオープン化と標準化を促す動きである．

一方，パーソナル化は，物のレベルで進行している．家庭でのテレビや自動車の一人一台化，オフィスでの電話やコンピュータの一人一台化は，これらの物の使い勝手や感性を重視した個性化を促している．

このような「システムのグローバル化」と「物のパーソナル化」の動きは，進展の著しい最近の情報化社会の核となる計算機システムの世界も例外ではない．ビジネス分野の情報システムは，従来の業務の合理化に加えて，最近では経営戦略を実現する戦略的情報システム SIS や統合製造システム CIM へと発展している．そのシステム構成はメインフレーム主体からワークステーション主体の分散システムへと発展しており，計算機システムのオープン化と標準化が進みつつある．

このような変化にともなって，計算機の利用方法も大型機の共同利用や小型機のグループ共用からワークステーションやパソコンの個人専用が中心となり，さらに，ブック型コンピュータの出現により，オフィス用，家庭用，携帯用の一人三台化へと向かっている．

このような状況下でハードウェア市場の年率 10% 程度の伸びに対し，ソフトウェア市場は 30% 前後の伸びを示している[1]．しかるにハードウェア技術の進歩に比べ，ソフトウェア技術の遅れが目立つ．ハードウェアの性能が 10 年に 100 倍の率で向上しているのに対し，ソフトウェアの生産性は 10 年に 2 倍ないし 3 倍程度である[2]．ソフトウェア危機は，「規模」と「量」と「質」の面からますます深刻になっている．

本解説では，2. で果てしないソフトウェア危機について分析し，3. で危機回避のシナリオを示す．4. でその主要なブレークスルー技術と思われるエンドユーザコンピューテング技術について述べる．

2. ソフトウェアの問題

2.1 ソフトウェア危機の歴史

（1） 1970 年代——「規模の問題」

最初にソフトウェア危機が言われたのは，1970 年ごろである．当時はハードウェアが非常に高価であり，グロッシュの法則に従ってどんどん大型化されていった．それにともなって，図-1[3] に示されるようにソフトウェアも大規模化していったが，このような大規模なソフトウェアを開発するための技術がなかったため，早晩ソフトウェアがハードウェアの大型化に対応できなくなるのではないかという心配があった．

そのため 70 年代は，二つのアプローチがとられた．その第一は，「構造化プログラミング」に代表されるプログラミング方法論と言語に関する研究である．プログラムの記述容易性よりも理解容易性のほうが重要視され，良い構造のプログラムを作るための構造化設計法や構造化プログラミン

† End-user Computing—Scenario for Overcoming Software Crisis by Takeshi CHUSHO (Systems Development Laboratory, Hitachi Ltd.).

†† (株)日立製作所システム開発研究所

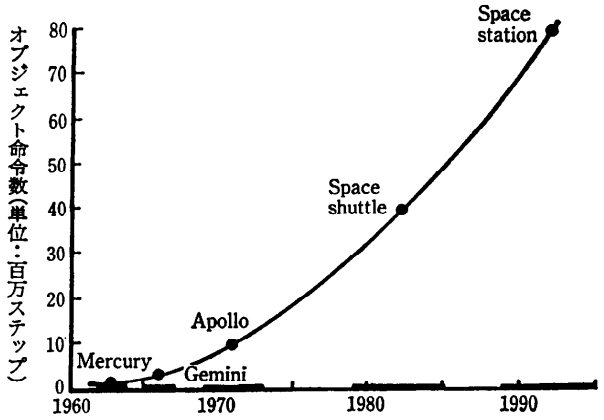


図-1 ソフトウェア規模の増大：米国の有人宇宙飛行プログラム
(IEEE と著者の許可を得て文献 3) より引用, © 1987 IEEE)

グ技法とその言語が開発され，普及していった．

その第二は，「ソフトウェア工学」に代表される研究であり，ソフトウェア開発工程のあらゆる局面，すなわち，要求定義，設計，製造，検査，保守運用を対象に方法論，技法，ツールの開発が行われてきた．しかし，解決すべき課題の多くが80年代に持ち越された．

(2) 1980年代——「量の問題」

規模の問題の後に量の問題が続いた．計算機システムの普及とともに，開発すべきソフトウェアの量が増大し，情報処理技術者の不足が第二のソフトウェア危機をもたらした．通産省では，1990年には情報処理技術者が60万人不足（需要：160万人，供給：100万人）という推定に基づき，1985年には Σ プロジェクトを発足した．さらに2000年には97万人不足（需要：215万人，供給：118万人）という推定が1987年に出されている．この現象は，かつての電話交換手の場合と同じく，“一億総プログラマ化”の不安をかり立てている．このような情報処理技術者の不足は，必然的にその平均的能力の低下という二次的な問題をも招いている．開発すべきソフトウェアがより複雑化し，情報処理技術者により高い能力が要求されるにもかかわらず，その平均的能力が低下している現象を報告者はセマンティクギャップになぞらえて「教育的ギャップの拡大」[4] と呼んでいる．

そのため，ソフトウェア工学の分野では，開発者の作業効率向上や個人差の縮小を目的とした統合的ソフトウェア開発支援環境や部品化再利用を目的としたパッケージ化技術が研究され始めた．さらに，ソフトウェア基礎論の分野では，従来の手続き的プログラミングに代わるさまざまの新しいプログラミングパラダイムの研究が行われてきたが，いまだ実用的成果をあげるに至っていない．

一方，官庁では情報処理技術者養成のために，労働省のハイテクカレッジ構想，通産省と労働省共同の地域ソフトウェアセンター構想，文部省の情報処理教育拡大策などの施策が取られている．

(3) 1990年代——「質の問題」

量の問題が未解決のまま，さらに質の問題が重要になってきている．質の問題は，グローバル化の観点での信頼性の問題とパーソナル化の観点での使い勝手の問題が大きい．前者では，社会的に重要な役割をになうメガステップオーダの大規模ソフトウェアの信頼性がますます重要になってきている．1990年にソフトウェアが原因で生じた計算機システムの事故の例をいくつか以下にあげるが，いずれも社会的影響の大きいものである．

- 銀行オンラインシステムのダウン
- 証券取引所売買システムのダウン
- 水道局システムの料金請求誤り
- 座席予約システムのダウン
- 電話交換システムのダウン（米国）
- 航空管制システムのダウン（英国）

また一方では，ハードの急速な進歩と低廉化により，計算機は情報処理の専門家に限らず，幅広く一般の人たちに利用され始めている．このような大衆化へ対応するためには，だれでも簡単に使えるソフトウェアの開発が必須である．さらにエンドユーザ自身が自分の使うソフトウェアを簡単に作れるようなエンドユーザコンピューティング技術が重要になっている．これらの課題が克服できなければ，ハードがいかに安くなっても宝の持ち腐れになってしまうという意味で第三のソフトウェア危機と言える．

2.2 現在の社会動向

情報化社会に対応して，これら「規模」と「量」と「質」に関するソフトウェア生産技術への要求は急速に増大している．その典型的なものとして，以下の二つをあげることができる．

- 「大規模高信頼ソフトウェアを早く作って，長く利用できる」技術

● 「使い勝手の良いソフトウェアを簡単に作って，どこでも利用できる」技術

第一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．企業における従来の計算機システムは，業務効率改善という合理化が主たる目的であったので，対象とする業務対応に比較的にまとまりのある中規模程度（100 ks オーダ）のアプリケーションソフトウェアを個々に開発してきた．しかし，最近，ビジネス分野の SIS，産業分野の CIM の構築にみられるように，「人」，「物」，「金」に続く第四の資源としての「情報」を経営に利用するための情報システムが注目され始めている．

SIS は，自社の競争優位性を確保するための企業戦略/経営戦略を具現するものである．顧客情報や商品販売情報の収集，分析，管理，あるいは新商品開発や生産計画，生産体制の実行管理などを迅速に行い，経営トップの意思決定機構と現場の業務機構が一体となって経営戦略を遂行できるような情報システムであることが重要である．

CIM は，売れる商品を早く作って，タイムリに顧客に提供するために，販売，物流，生産，開発の各システムを統合したシステムである．パーソナル化の観点では顧客のニーズの多様化に対応した多品種生産を可能とし，グローバル化の観点では生産，開発の現地化などの国際化戦略を促進することになる．

したがって，これらの計算機システムとしては，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．

第二の使い勝手の良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ともなって，これらの一人一台化が進行している．この傾向は，ブック型コンピュータに代表されるポータブル型やラップトップ型の低廉化とともに加速され，オフィス，野外，家庭での一人三台化の時代になりつつある．このような分野では，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，使い勝手の良い 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．パソコンの“パーソナル性”が追求され，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．

さらに大型機によるホスト集中システムから中小型機による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移植性が重要になる．ワークステーションを中心とした小型の分野では，OS，ネットワーク，データベース，グラフィカルユーザインタフェースなどのソフトウェアの標準化の動きが活発になっており，ソフトウェアの作り方が変わってくる．これらの標準の上で動くパッケージの利用や簡易言語を中心としたエンドユーザコンピューティングの傾向が強まり，オープンシステム化を加速している．

2.3 ソフトウェア生産技術の問題点

ソフトウェアがあらゆる分野で計算機システム普及のキーテクノロジになってきている．しかしながら，現在のソフトウェア生産技術は，職人芸的生産技術および労働集約型生産形態から脱皮していないため，上記のようなソフトウェアの大規模化，量の増大および機能の複雑化に対応できていない．「欲しいソフトウェアを早く，安く入手できる」ようにはなっていないが，自前で新規開発する場合は以下のような問題がある．

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

3. 危機回避のシナリオ

3.1 基本的解決方法

ソフトウェアの問題の具体的な形態は多種多様であり，解決方法も条件によって異なることが多い．メーカ視点（作る立場）では，ソフトウェアの生産性向上に関して，以下に示す 5 項目の一般的な解決方法が考えられる．

①開発対象（ソフトウェア）の標準化
②開発工程（プロセス）の定式化，自動化
③開発手段（ツール）の高機能化
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

これらは，新規開発量の抑制，工数削減，作業効率の向上，労働の質の向上などの形で直接，間接

表-1 ユーザ視点でのソフトウェア危機回避シナリオ

解決手段	ユーザのソフトウェア入手方法
標準化	適正価格の市販パッケージを購入
自動化	要求仕様を提示し，自動生成
情報処理技術者の自由業化	大金を払って優秀な技術者に依頼
エンドユーザコンピューティング	自分で作成

に「規模」,「量」,「質」の問題の解決に寄与する.

3.2 解決へのシナリオ案

これらの解決方法をユーザ視点（使う立場）で見直すと，表-1 に示すような4種類のドラスティックなソフトウェア危機回避シナリオ案を描くことができる.「標準化」シナリオは 3.1 の解決要因の①,「自動化」シナリオは②と③,「情報処理技術者の自由業化」シナリオは④,「エンドユーザコンピューティング」シナリオは⑤に対応する.

（1）「標準化」シナリオ

「標準化」は古くて新しい問題である．標準化の基本は共通化による共有化であろう．ネジの標準サイズの設定や，フロッピディスクの標準仕様の設定などの利点は言うまでもない.

ソフトウェアの標準化の利点も明白である．ユーザにとっては，入手が容易で，その標準ソフトウェアを用いて作成したデータの互換性やアプリケーションソフトウェアの移行性が保証される．メーカにとっては，生産物の価値を n 倍化するものである．n カ所で使用されればステップ単価は $1/n$ になる.

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る.

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と③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，基本ソフトやミドルソフトを共通化するものである．そのうちの②は，自社内のスーパ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狙っている．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図っている.

一方，③は，アプリケーションソフトウェアアーキテクチャの各階層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するものである．古くから行われてきたプログラミング言語のほかに，通信手順 (OSI) などが国際標準化されている．また，UNIX*，GUI，ウィンドウ，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.

最後の④については，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パッケージもある．このようなアプリケーションパッケージは，業務の知識と情報処理の知識を一体化して作成するという意味で，知識集約型産業といえる.

90年代のシステム構成は，大まかには，応用ソフト/基本ソフト/ハードの3階層で，かつ上位の応用ソフトと下位のハードの種類は豊富になり，真中の基本ソフトの選択が限定されたワイングラス型になる．したがって，ユーザは図-2 のようなソフトウェアシステムの構築方法になると思われる.

このように標準化が進めば，ユーザは，自分の業務に必要な市販のアプリケーションパッケージを適正な価格で買ってくることができる.

（2）「自動化」シナリオ

「自動化」，すなわち「自動プログラミング」[5]

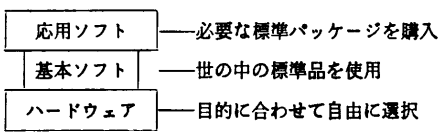


図-2 「標準化」シナリオにおけるオープン指向システム構成

* UNIX オペレーティングシステムは UNIX System Laboratories, Inc. が開発し，ライセンスしている.

は，フォンノイマン型計算機の発明以来のソフトウェア技術者の夢である．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，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．最近では，対象分野を限定した問題向き言語や4GLによる自動化率の向上が実現している．

現在は，仕様記述言語と仕様からのプログラム自動生成技術の研究が活発に行われている．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，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．その後のツール化は下流中心に行われた[6]．

しかしながら，ソフトウェア開発の真の難しさは上流工程にあるという認識から，最近再び上流のツール化の試みが積極的に行われ始めた．このような上流工程支援を含む統合開発環境を統合CASEと呼ぶ[7]．CASEの目的は，基本的には「思考の道具」と「自動化ツール」を提供することであろう．

最終的には，ユーザは，自分の業務の用語で要求定義を記述するだけで，プログラムを自動生成することができる．

（3）「情報処理技術者の自由業化」シナリオ

現在，情報処理技術者の社会的待遇がほかに比べて優位にあるということはないが，今後，急激な変化がありえる．受注ソフトのバックログが増加しており，市場価格は高騰するはずである．そして，情報処理技術者の高給化が実現するはずである．

もちろん，この業務は個々の技術者の能力差が大きいので，「ソフトウェアの良し悪しを見分けるもの差し」が不可欠である．そして建築士などの専門家と同じように個人の能力に応じた収入を得られるように自由業化する．銀行オンラインシステムや電話交換システムなど，一般の人たちの日常生活と深く関わるシステムの数が増加するので，社会的な影響の大きいソフトウェア事故が頻発し，優秀な情報処理技術者の高収入は保証されるであろう．また，自分の作ったアプリケーションパッケージやゲームソフトがベストセラーになれば大金持ちになれるというビジネスチャンスが広がる．

その結果として，多くの人々が優秀な情報処理技術者になるために自らすすんで努力をするようになり，ユーザは，お金さえ払えば，優秀な技術者が作る良質のソフトウェアをタイムリに入手できる．

（4）「エンドユーザコンピューティング」シナリオ

ユーザ業務を大まかに定形業務と非定形業務に分けて考えると，定形業務については，「標準化」シナリオあるいは「自動プログラミング」シナリオによって解決する．一方，非定形業務については，自分の業務に固有の部分のソフトを作成する必要がある．今後急激な増大が予想されるこのようなソフトはユーザ自身が自分で開発できることが理想である．ビジュアルプログラミング，日本語プログラミングなどはその方向を目指している．知識工学応用として実用化が進んでいるエキスパートシステムも，ルールをユーザが記述し，保守も可能という意味において，ルール指向の簡易プログラミングと考えられる．

エンドユーザコンピューティングが発展すれば，ユーザは，“プログラミング”を意識することなく，自分の業務をメタファベースで，あるいは業務の言葉を用いて簡単に計算機化できる．

3.3 ソフトウェア産業論

このようなドラスティックなシナリオ案に対応する技術の位置付けを明確にするために，ソフト産業の進化過程を表-2のような図式でとらえる．この分類では，必ずしも世の中一般の用語の定義と一致しない部分もあるが，あえて単純な図式を作ってみた．

ハードウェア産業の華々しさとは対照的に，ソフトウェア産業はいまだに未成熟である[8]．ソフトウェアの生産技術がハードウェアのそれとのア

表-2　ソフトウェア産業の進化過程

ソフト産業の形態	主要な技術職	主要技術
労働集約型産業	プログラマ	自動化（CASE）
知識集約型産業	設計者	標準化（パッケージ）
知恵集約型産業	業務専門家	エンドユーザコンピューティング

ナロジで語られることが多いが，ハードに比べてソフトは見えない部分が多いため，定量的なもの差しが少ない点が決定的な違いである．ソフトウェア産業を真に“産業”たらしめるために，そして，ソフトウェア工学を真に“工学”たらしめるためにも，第一にソフトウェアの生産に関与する諸要因を観測可能にすること，第二にその観測データを科学的根拠のあるもの差しを用いて評価可能にすることが望まれる．ソフトウェアメトリックスの対象はあまりにも多い．

（1） 労働集約型産業

現在のソフトウェア産業の大部分に相当する労働集約型産業においては，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る．ここで最も重要なメトリックスは，ステップ数/人月あるいはステップ単価である．メーカは，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いる．しかし，この尺度の致命的欠陥は，ソフトウェアの価値（質）を規模（量）ではかることである．

しかし，実際には，多種多様なソフトウェアに対して多種多様な生産性の定義があるはずである．不良品のステップ単価がどんなに安くても嬉しくはないし，信頼性を2倍にするためにステップ単価が世間相場の10倍してもよいものもある．1カ所でだけ稼働するメガステップオーダのオンラインプログラムと100万本売れるキロステップオーダのゲームソフトを同じ生産性の尺度で計ることは意味がない．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．中でも最も重要なメトリックスは，「ソフトウェアの良し悪しを見分けるもの差し」であろう．

（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

「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．今後，アプリケーションソフトウェアのビジネスは，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，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） 知恵集約型産業

情報処理システムが，業務の効率化ではなく，経営戦略の実現に用いられるようになると，ソフトウェア開発も，効率よりも効果が重要視される．SIS や BIS では，何を作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に作っていくためには，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．

4. エンドユーザコンピューティング

エンドユーザコンピューティングは，**3.** で述べたように情報化社会の発展過程で必然的に生じるニーズであり，それがソフトウェア危機回避のためのブレークスルー技術でもある．米国では，すでにオフィスにおけるエンドユーザコンピューティングのための投資が情報処理システム全体の約 40% を占めると言われている．

ここでは，エンドユーザがソフトウェアを自ら作り，自ら使い，自ら保守拡張するための現実的な技術について述べる．

4.1 プログラミングパラダイムの転換

4.1.1 プログラミング言語のトレンド

プログラミング言語は，当初からソフトウェア生産性向上に大きな役割を果たしてきた技術である．プログラミング言語の研究開発に関するこれまでのマクロなトレンドは**表-3** のように要約できる．

最初の 1960 年代は，高価な計算機を効率良く

表-3 プログラミング言語の発展過程

時 期	目 的	内 容
1960 年代	量的高級化	記述水準（機械語への展開率）の向上
1970 年代	質的高級化	プログラミング方法論（構造化技法）の導入
1980 年代	パラダイム転換	宣言的記述（手続き型から非手続き型へ）
1990 年代	エンドユーザコンピューティング	脱プログラミング

利用する必要のあったコンピュータ利用の初期の段階に，プログラムの実行効率をあまり低下させないで生産効率をあげるために，アセンブリ言語から FORTRAN, COBOL, PL/I などの高級言語への移行が行われた．これは，機械語への展開率の向上という意味で「量的高級化」と言える．

次に 1970 年代は，ソフトウェアの大規模化という第一次のソフトウェア危機への対応として，高信頼性と保守性を重視したプログラミングスタイルの研究が行われ，構造化技法などによりプログラムの理解容易性を実現する「質的高級化」[4]が追求された．

1980 年代には，ソフトウェアの生産性，信頼性，保守性に関する諸問題は，本質的に手続き型のプログラミングパラダイムに起因するという考えから，論理型，関数型，オブジェクト指向型[9]などの新しいプログラミングパラダイムの研究が盛んに行われるようになった．

さらに最近では，情報処理の専門家でなくてもソフトウェアを開発できるように，“脱プログラミング”を実現するエンドユーザコンピューティングの研究が盛んである．

4.1.2 エンドユーザコンピューティング技術

本項では，エンドユーザがプログラミング言語を用いないで業務の計算機化を行えるための現実的な技術として，以下の 4 項目について述べる．

①業務向け簡易言語
②日本語プログラミング
③ビジュアルプログラミング
④AI（ルール，ファジィ，ニューロ）

いずれも脱プログラミングのコンセプトに基づいているが，実際には“脱プログラミング言語”のレベルであり，プログラミングの概念やセンスが不要というわけではない．自ら作って使うという意味では，エンドユーザコンピューティングというよりもユーザオウンコンピューティングと言ったほうが良い．ユーザが作らないで使うだけの OA 用標準パッケージなどはここでは述べないが，ハイパテキストやデータベース検索，意思決定支援などの分野もユーザオウンコンピューティングの観点で注目していく必要がある．

4.2 業務向け簡易言語

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．James Martin[10] は，4 GL を 13 カ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，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 より一桁少ない，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 の現状分析は文献 11）に詳しいが，それによれば，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．

4.3 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[12]．特にアプリケーションプログラムの保守性に不可欠の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分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が使える．

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を

克服する必要がある.

①業務で使用している用語をそのまま使用するための業務用語辞書作成保守機能.

②日本語記述レベルで閉じたシステムとし,それから変換されるプログラミング言語の知識を不要とする開発保守環境.

③日本語表現がもつ本来的なあいまい性を避け,かつ日本語としての自然な表現を保持した文法規則.

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,データベース検索用自然語インタフェースHNLDB[13]では,

「箱根の宿泊先を費用の安い順にだせ」

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.本システムでは,上記の課題を克服するために

①入力例文集から近いものを選んで修正して使用,

②システムの解析結果(検索条件,対象)の修正,

③あいまい語,同義語の登録,

などの機能をユーザに提供して,日本語インタフェースを使いやすいものにしている.

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COBOLやPL/Iなどのプログラミング言語で開発してきたが,次期のシステムは,以下のような理由で従来方式では無理と言われている.

①開発管理が不可能な規模になる.

②計画段階から稼働までの期間が長くなると,数年先をみた計画が不可能になる.

③開発中および稼働後の機能変更や機能拡張が頻繁に発生するが,タイムリな対応が不可能になる.

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,上記の問題は軽くなる.われわれは,その一つの試みとして,金融向け業務仕様記述言語システムSPECTRUMの研究をしている.文型を用いた限定日本語の仕様記述言語を開発し,メニュー選択方式でプログラムを誘導する方式を取っている.画面の例を図-3に示す.画面の左上と左下のウィンドウが日本語仕様の記述例である.本システムは以下の特徴がある.

①日本語文型はシステム組込の21種類の組合せで十分記述可能であるが,ユーザ定義も可能.(画面の上中央に文型例)

②限定日本語文の目的語をオブジェクト,動詞をメソッドとみなし,表形式のオブジェクト指向プログラミング言語に内部変換することにより,意味を厳密に規定.(画面の右上に例)

③この表形式のオブジェクト指向プログラムを業務用語辞書やプログラム部品ライブラリを用いてPL/Iプログラムに変換.(画面の右下に例)

4.4 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難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.

最近のビットマップディスプレイ付きワークステーションの普及とともに,以下の三つの観点からの「プログラムの視覚化」の研究[14)~17)]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

第一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.代表的なシステムとしてSmalltalk-80*がある[18)].

第二の視覚的編集は,本来テキスト表現を基本に作られたプログラミング言語を用いてプログラムを作成する場合,通常はテキストエディタを使用する.しかし,プログラムを単なる文字列とし

* Smalltalk-80は米国Xerox社の登録商標である.

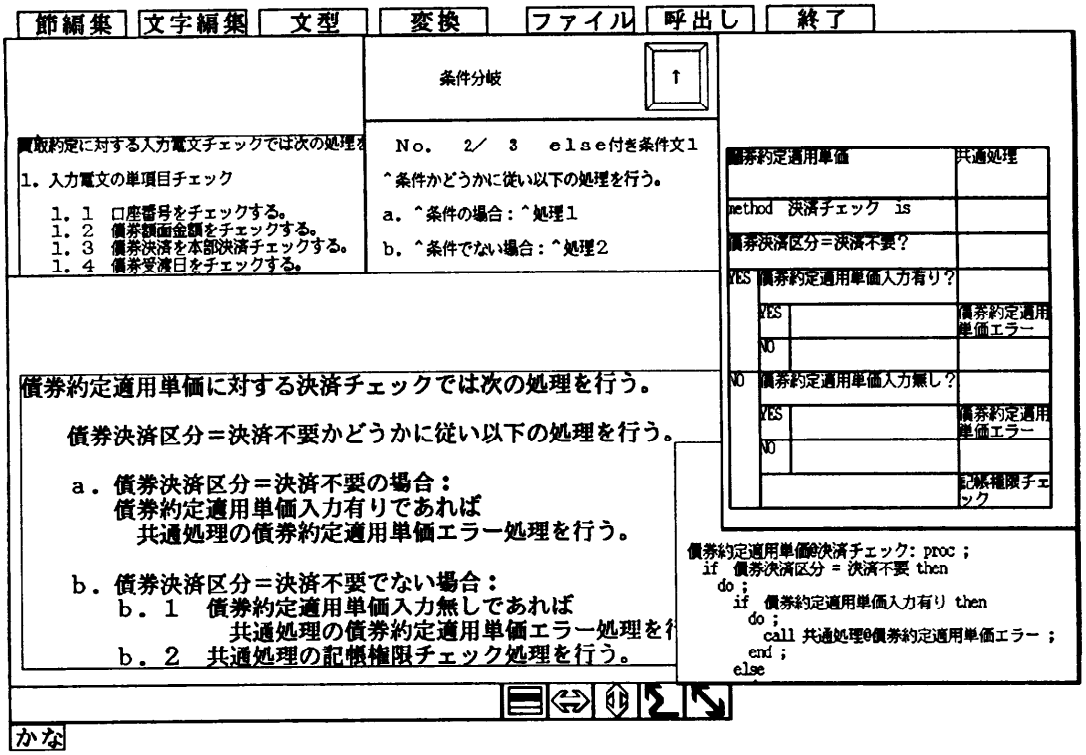


図-3　日本語仕様記述言語システム SPECTRUM の画面例

て扱うため，文法的エラーを作り込みやすい．そこで，プログラミング言語の文法規則をシステム側が知識として保有しておき，テンプレートに穴埋めさせる方式で文法的に正しいプログラムを誘導する構造エディタが普及し始めている．報告者らも構造エディタ PARSE[19] と構造エディタジェネレータ[6]を開発した経験がある．

第三の視覚的言語は，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的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[20]．

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．先に述べた SPECTRUM システムの表形式オブジェクト指向言語もその一例である．限定日本語による仕様記述のほかに，図-3 の右上のウィンドウに示すような表形式で直接仕様を記述するためのエディタを用意している．

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，制御フロー図，ブロック図，状態遷移図などがある．実行可能な仕様記述言語[21]の例として，Statechart[22] がある．これは，分かりやすさの利点をもつ視覚性と厳密さの利点をもつ形式性をあわせもつ視覚的形式性という概念に基づき，状態遷移図を拡張したものである．図-4 にディジタル腕時計の 4 種類のボタンを押したときの表

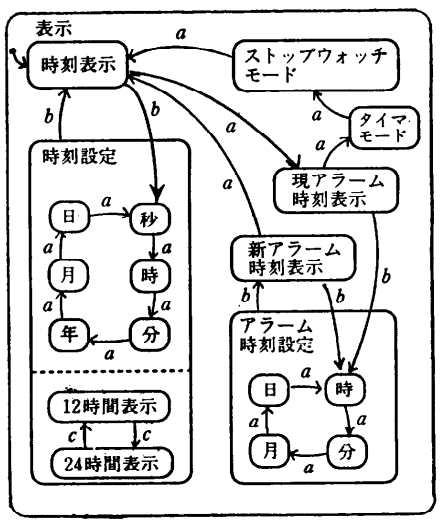


図-4　Statechart の例（ディジタル腕時計の仕様の一部）

示機能を Statechart で記述した例を示す.

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．この例として，広島大学の視覚的プログラミング環境 HI-VISUAL[23] や GUI 構築ツールである CMU の Garnet[24], NeXT コンピュータのインタフェースビルダなどがある．

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．ユーザが記述したい業務内容に本質的に必要な部分にのみプログラミングの概念が用いられるのが理想であるが，当面は脱プログラミングではなくて，脱プログラミング言語が目標となろう．

なお，ここでは視覚的技術の説明に A. Ambler[16] および西川ら[20] の分類を用いたが，新しい研究分野の常として，まだ用語の定義は定まっていない．

4.5 AI（ルール，ファジィ，ニューロ）

プロダクションルールによる知識表現をベースとしたエキスパートシステム[25]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．先に述べた日本語プログラミングの場合と同様に，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，日本語プログラミングの場合は通常のプログラミングと同様の手続き的概念が必要であるが，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．

図-5 に報告者らが開発したエキスパートシステム構築ツール ES/X 90 の例[26]を示す．図の（a）は，ES/X 90 の知識テーラ（専用知識エディタ作成システム）を用いて作成した自動車購入相談エキスパートシステム用の簡易知識エディタである．自動車販売担当者は，

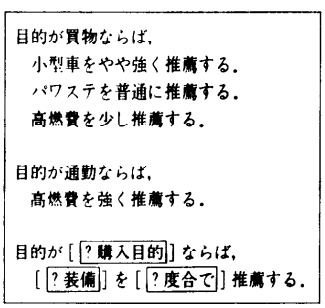
目的が［？購入目的］ならば，

［？装備］を［？度合で］推薦する．

という文型の［？］の部分にあてはめる用語を選択して，販売ノウハウを入力すれば，自動的に図（b）のプロダクションルールが生成される．このような方法により応用分野の専門家自身が知識を入力，管理できる．

ルール表現の一つの問題は，真偽が明確でないあいまいな知識を表現するために，ルールに確信度を付加したり，ルールの条件部に真偽判定の可能な表現を用いてやる必要があるが，この数量化が難しい．この問題は，ルールの条件部や結論部にファジィ表現を用いることで解決する．

さらにニューラルネットワークの学習機能を用いれば，ルールの確信度やファジィのメンバシップ関数の設定が不要になる．ただし，この場合はうまく学習させることがプログラミングに代わる課題となる．Learning is programming といわれるゆえんである．



（a） 自動車購入相談エキスパートシステム用簡易知識エディタ

```
if 購入者＃目的＝買物
  then +(小型車, 0.6),
       +(パワステ, 0.4),
       +(高燃費, 0.2);

if 購入者＃目的＝通勤
  then +(高燃費, 0.8);
```

（b） 内部表現（ルール形式）

図-5 エキスパートシステム構築ツール ES/X 90 における日本語知識表現例

5. おわりに

本解説では，「規模」，「量」，「質」の問題をかかえて果てしなく続くソフトウェア危機を解決するシナリオとして，自動化，標準化，自由業化，エンドユーザコンピューティングの 4 種類について考察した．特にエンドユーザコンピューティングについては，ソフトウェア危機回避のブレークスルー技術として現在の技術動向についても述べた．

エンドユーザコンピューティングの分野で脱プログラミングを達成するためには，ソフトウェア科学，ソフトウェア工学，ソ

フトウェア産業，エンドユーザの協力が必須となろう．しばしばテクノロジトランスファ（技術移転）の難しさが指摘されるが，その前にお互いのパーセプションギャップ（問題認識の溝）を解消することが重要であり，テクノロジトランスファとは逆方向のパーセプショントランスファ（問題認識の移転）が鍵となるように思われる．

参 考 文 献

1) 情報処理市場展望 ダウンサイジングの影響出始め，構造変革期に突入：日経コンピュータ 1991. 3. 25 号，62-83 (Mar. 1991).
2) 大野 豊：ソフトウェア工学の背景と展望，情報処理，Vol. 28, No. 7, pp. 845-852 (July 1987).
3) Boehm, B. W.: Improving Software Productivity, IEEE Computer, Vol. 20, No. 9, pp. 43-57 (1987).
4) 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 (June 1983).
5) 原田，福永（編集）：大特集：自動プログラミング，情報処理，Vol. 28, No. 10, pp. 1261-1411 (1987).
6) 野木，中所：プログラミングツール，p. 202，昭晃堂 (Apr. 1989).
7) 大箸，川越（編集）：特集：CASE 環境，情報処理，Vol. 31, No. 8, pp. 1012-1085 (1990).
8) 自立への道みえぬ向上心乏しいソフト業界，日経コンピュータ，1990. 12. 31 号，pp. 30-45 (1990).
9) 中所武司：使いやすいソフトウェアと作りやすいソフトウェア―オブジェクト指向概念とその応用―：電気学会雑誌，Vol. 110, No. 6, pp. 465-472 (June 1990).
10) Martin, J.: Fourth Generation Languages, Prentice-Hall (1985).
11) 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).
12) 高橋延匡：日本語プログラミング環境，情報処理，Vol. 30, No. 4, pp. 363-372 (1989).
13) Kinukawa, H.: A Natural Language Interface Processer Based on the Hierarchical-tree Structure Model of Relation Tables, Journal of Information Processing, Vol. 11, No. 2, pp. 83-91 (1988).
14) Glinert, E. P. (Ed.): Visual Programming Environments: Paradigms and Systems, p. 661, IEEE Computer Society Press (1990).
15) Glinert, E. P. (Ed.): Visual Programming Environments: Applications and Issues, p. 687, IEEE Computer Society Press (1990).
16) 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 (Oct. 1989).
17) Chang, J.: Visual Languages: A Tutorial and Survey, IEEE Software, Vol. 4, No. 1, pp. 29-39 (1987).
18) 横手靖彦：オブジェクト指向言語のプログラミング環境，情報処理，Vol. 30, No. 4, pp. 334-346 (1989).
19) Chusho, T., Watanabe, T. and Hayashi, T.: A Language-Adaptive Programming Environment Based on a Program Analyzer and a Structure Editor, Proc. the 9 th World Computer Congress IFIP '83, pp. 621-626 (Sep. 1983).
20) 西川，寺田：視覚的プログラミング環境，情報処理，Vol. 30, No. 4, pp. 354-362 (1989).
21) 大槻 繁：制御用ソフトウェアの仕様化技法，計測と制御，Vol. 29, No. 1, pp. 982-988 (1990).
22) Harel, D.: On Visual Formalism, Comm. ACM, Vol. 31, No. 5, pp. 514-530 (1988).
23) Hirakawa, M., Tanaka, M. and Ichikawa, T.: An Iconic Programming System, HI-VISUAL, IEEE Trans. Software Eng., Vol. 16, No. 10, pp. 1178-1184 (1990).
24) Myers, B. A. et al.: Garnet Comprehensive Support for Graphical, Highly Interactive User Interfaces, IEEE Computer, Vol. 23, No. 11, pp. 71-85 (1990).
25) 上野晴樹（編集）：特集：エキスパート・システム，情報処理，Vol. 28, No. 2, pp. 146-236 (1987).
26) 中所武司：ビジネス業務をマルチエキスパートシステムでモデル化，日経エレクトロニクス，1988. 3. 21 号，pp. 143-151 (1988).

（平成 3 年 3 月 22 日受付）

中所 武司（正会員）

1946 年生．1969 年東京大学工学部電子工学科卒業．1971 年同大学院修士課程修了．工学博士（東京大学）．1971 年(株)日立製作所入社，現在同社システム開発研究所主任研究員，東京工業大学非常勤講師．入社以来，プログラミング方法論，構造化技法と言語，テスト技法，構造エディタ等のソフトウェア工学およびオブジェクト指向言語，知識処理言語，エキスパートシステム構築ツール等の知識工学の研究に従事．1982 年度情報処理学会論文賞，1986 年度大河内記念技術賞受賞．著書「プログラミングツール」，「人工知能」（昭晃堂，共著），電子情報通信学会英文論文誌 D 編集委員，日本ソフトウェア科学会編集委員，人工知能学会，IEEE Computer Society 各会員．