第３章

危機回避のシナリオ

3.1 ソフトウェア工学の誕生

1968年にドイツのガルミッシュで開催されたNATO会議において，ソフトウェア危機を克服するための技術開発の必要性から，はじめてソフトウェア工学(software engineering)という言葉が登場した。これ以降，ソフトウェアの開発方法は，これまでのように職人芸的ないし手工業的観点を脱し，工学的観点で研究されるようになっていった。

1975年には米国の電気電子学会（IEEE：The Institute of Electrical and Electronics Engineers, Inc.）においてソフトウェア工学の論文誌（TSE：Transactions on Software Engineering)の発行が開始された。同じ年にソフトウェア工学国際会議(ICSE：International Conference on Software Engineering）が始まり，1982年の第6回の会議は東京で開催された。

今日まで，要求定義から設計，プログラミング，テスト，保守にいたるソフトウェアライフサイクル全般にわたり，技術と管理の両面から研究が行われ，多くの成果を生み出してきた。

3.2 生産性の定義

ソフトウェア工学は，他の従来の工学と大きく異なり，物理法則や化学法則などの科学的根拠を持たない。自然現象を対象とした物理の世界とは正反対の論理の世界を対象としているため，他の工学よりも技術体系の構築が難しい面

がある。たとえば，プログラムという「もの」は作っているが，「物」は作っていないため，可視性に欠け，メトリックスが弱い。

そこで，ソフトウェアの生産技術について考察する前に，ソフトウェアの生産性について考察しておく。従来からよく「あるツールを利用するとある工程の生産性がP倍になる」という表現が用いられることが多いが，これは主に同じソフトウェアを作るのに工数が1/pで済むということ，あるいはステップ数/人月がp倍になることを意味している。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ヶ所でだけ稼動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。

ここでは，まずマクロな観点から以下のように生産性を定義する。

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。一方，「生産コスト」は，人件費のほかに開発環境やオフィス環境などへの設備投資を含む。したがって，生産性向上の努力としては，

- ●同じ価値のソフトウェアをつくるための生産コストをできるだけ少なくする技術，
- ●同じ生産コストでつくれるソフトウェアの価値をできるだけ大きくする技術，
- ●生産コストを積極的に増やして，ソフトウェアの価値をそれ以上に増やす技術，

などがある。特に，従来は一番目の生産技術が中心であったが，今後は二番目，三番目の技術も重要である。

さらに，ソフトウェアの生産性に関しては，個人差の問題も重要である。あるデータによれば，ツールの良し悪しによる差が1.5倍なのに対し，個人差は4倍以上であるといわれている。同一人物でも種々の環境条件で効率に大きな

差がでると思われる。したがって，

●個人差を吸収できる生産技術，

●個人差を活かした生産技術，

という観点も重要である。

3.3 基本的解決方法

ソフトウェアの問題は，第2章で述べたように多種多様であり，何を開発するか，誰が使うか，誰が開発するか，等々の条件によって問題が異なることが多い。

ここでは，まずメーカ視点（作る立場）から，ソフトウェア生産性に関する一般的な解決要因として以下の5項目を考える。

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，間接に「規模」，「量」，「質」の問題の解決に寄与する。

次に，これらの解決要因をユーザ視点（使う立場）で見直すと，**表3.1**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」シナリオは上記の解決要因の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。

表3.1　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

3.4 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，フロッピー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箇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幾つかの段階がある。

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計っている。

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他に，通信手順（OSI：Open Systems Interconnection）などが国際標準化されている。また，UNIX，GUI(graphical user interface)，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，知識集約型産業と言える。

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 3.1 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。

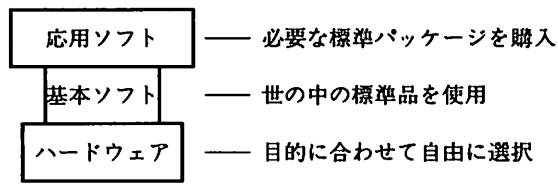

図 3.1 「標準化」シナリオにおけるオープン指向システム構成

3.5 自動化シナリオ

「自動化」，すなわち「自動プログラミング」は，フォン・ノイマン型コンピュータ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 4 GL 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。その理由は，技法の適用プロセスの自動化が難しいためと思われる。その後のツール化は下流中心に行われた。

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 (Computer Aided Software Engineering) と呼ぶが，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。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) 方法論

統合開発環境を構築するためには，まず開発工程の定式化が重要である。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。工程の定式化では特に工程間の関連付けが重要であり，プロセスプログラミングなど，開発保守工程全体を通した方法論の確立が必要である。

(2) アーキテクチャ

CASEのソフトウェアアーキテクチャは，できるだけ単純な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，等の処理を容易にする必要がある。筆者も，ユーザインタフェース管理をフロントエンド，プログラム管理をバックエンドとし，その間にツールを埋め込むサンドウィッチ型ソフトウェアアーキテクチャを開発し，統合プログラミング環境PERFECT (Programming EnvirRonment For Editing, Compiling and Testing) に適用した経験がある。(文献94)

(3) 情報管理

仕様情報，仕様書，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のSEDB (software

engineering database)あるいはリポジトリ，さらにそこでの管理情報と各種ツールとのインタフェースとしてのオブジェクト管理プラットフォームなど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。

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。思考の道具は開発者の助手，自動化ツールは開発者の代理／代行者といえる。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。

3.6　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他に比べて明らかに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，受注ソフトの伸びは，需要ではなく，供給で決まる状態になりつつあり，市場価格は高騰するはずである。そして，現在のソフトウェア生産技術では，その供給の伸びは，情報処理技術者数の伸びで決まることになるから，情報処理技術者の高給化が実現するはずである。

勿論，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さし」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ーに入手できる。

3.7 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「自動化」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ウェア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ウェア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コンピュータ化できる。