ソフトウェアエンジニアリングのすすめ　―原理と潮流と予測と課題―

(株)日立製作所 システム開発研究所
大槻 繁

はじめに

製品としてのソフトウェア開発に携わっておられる方々は、現世のご利益というか、適切なコードを効率よく開発するための支援環境に興味があると思いますが、ここでは、大局的な視点からソフトウェア生産技術について眺めてみようと思います。

組み込みシステムの分野においても、ソフトウェアの占める役割は、ますます重要になって来ています。図１はLSIの集積度と開発コード量の生産性に関する状況を単純化して示したものです。

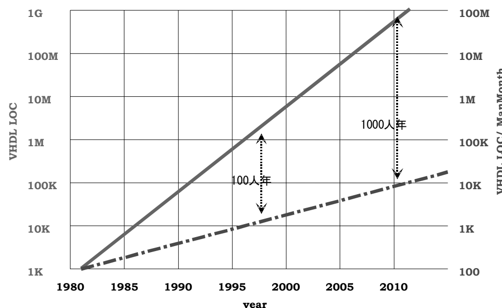

図1 LSIの集積度と開発コード量の推移

LSIの集積度は、2000年現在で10^7～10^8 Logic Tr／Chipにまで達しています。これはVHDL記述で数MLOC（百万ステップ）に相当する分量です。設計の生産性は、RTLレベルの記述からLSIレイアウトデータまで、種々のツールを駆使したとしてもせいぜい1KLOC（千ステップ）／人月であるから数百人年の工数がかかっていることになります。

集積度の向上は、2010年には10^9 ～10^{10} Logic Tr／Chip程度になると予想されており、これは年率で58%の伸びになります。一方、生産性は、記述の抽象度を上げることによって過去には年率で21%の伸びを示して来ましたが、最近では配線遅延がゲートの遅延を越えてしまうという論理合成上の問題があり、このままでは、せいぜい10KLOC／人月程度にとどまってしまうと言われています。

エンジニアリングというのは、常に原理的な限界を見極め、その中で、製品開発というミッションを満たすために解を見いだす活動ですから、こういった規模に関する推移は、時々、確認しておくことをおすすめします。

ソフトウェアエンジニアリングとは

ソフトウェアエンジニアリングとは、ソフトウェア開発に関わる広義の言語活動のことを示しています。

図２に示すように、ソフトウェア開発では、記述(プログラミング言語,仕様記述言語のみならず,図式や,ユーザインタフェース上の規約等）を媒体として用いるということは普遍的であり、これに関わる諸活動を体系化し、技術として蓄積することがソフトウェアエンジニアリングの目的と言えるで

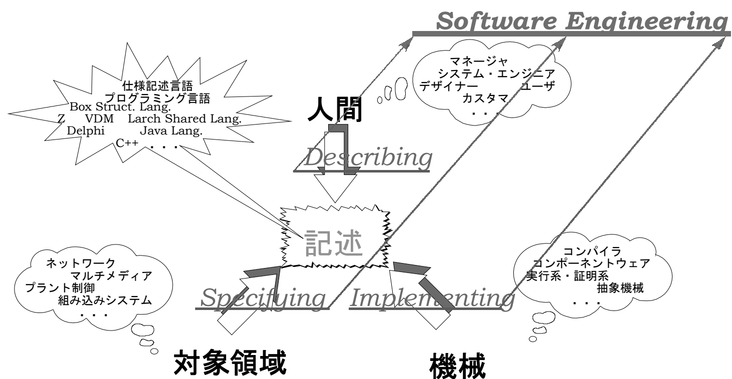

図2 ソフトウェアエンジニアリングに関する3つの世界

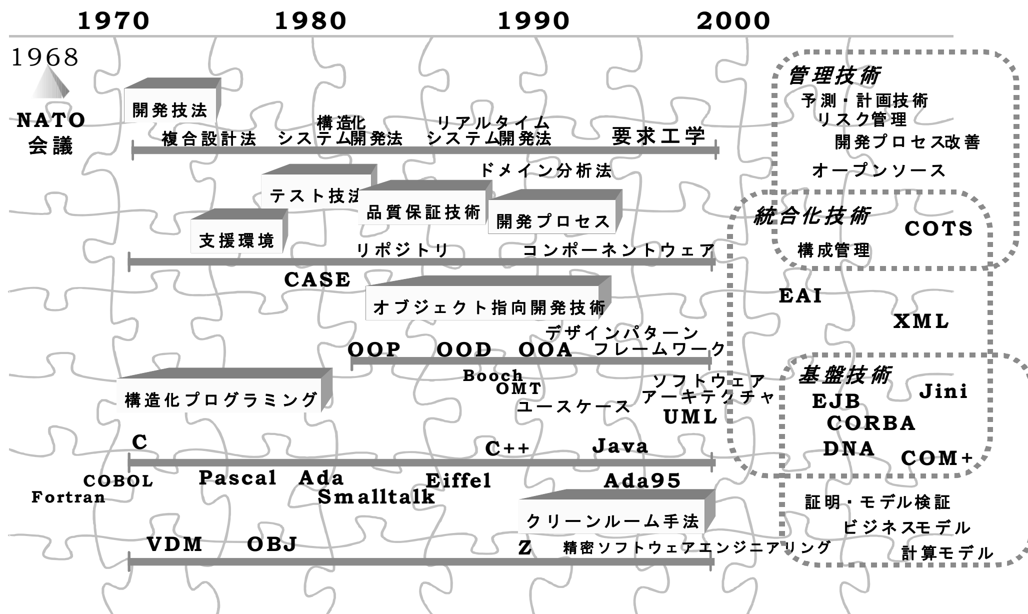

図3 ソフトウェアエンジニアリングの変遷

しょう。

(1)対象領域:Specifying/Modeling(言語設計論)

対象領域の問題をモデル化し、それを記述としてまとめることに相当しています。アルゴリズム、シミュレーション、科学技術計算、事務処理、制御分野といった問題領域を見据えてプログラミング言語は設計されてきました。計算機による実行特性がない言語では、伝達を目的とした図式言語、厳密な検証を目的とした仕様記述言語などもあります。言語を設計することは、対象領域の特性を見極め、それをモデル化してバランスよく仕上げる創造的活動です。

(2)機械:Implementation/Compiling(実装論)

言語を機械（計算機）上で処理・実行・検証等を行えるように実現することに相当しています。狭義の計算機実装論は、いわゆるコンパイラ技術である言語の構文解析技術や最適化技術を示しています。この領域で重要な事柄は、計算機の構造や処理方式、アーキテクチャを考慮し、それ等の資源をどのように有効に扱うかということです。対話型や複数のツールから構成される統合環境、さらには、ソフトウェアの構造や部品、ライブラリなども考慮する必要があります。

(3)人間:Description/Programming(言語用法論)

人間が言語を使って記述（伝達・定義）することに相当しています。初期のプログラム書法に始まり、最近の仕様記述法に至る技術で、狭義のソフトウェアエンジニアリングとして位置付けられる領域です。人間は考えをまとめるため、それを伝えるため、計算機に司令を出すため、出来上がったプログラムを保守するためなどに言語記述を使用します。ある言語の記述を得るために前提となる記述もあり、こういった複合的な記述手順として、種々の開発技法があみ出されてきました。

三つの世界（対象領域, 機械, 人間）の構造は全く異種のものであり，これ等を同時に扱わなくてはならないところにソフトウェアエンジニアリングの原理的な難しさがあります。

対象領域というのは、機械も人間もない世界として一般的には成立しています。物理現象、装置、

物体などから構成され、そこに存在・機能しています。機械の世界というのは、主としてフォンノイマン型の演算装置と記憶装置とからなる世界で、これも対象領域も人間もなくても成立しています。人間の世界もまた、その思考や認識の仕方、社会的な形態として有史以来存立しています。

ソフトウェアエンジニアリングの歴史

ソフトウェアエンジニアリングは、図3に示すように、その発祥より高々30 年たらずですが、概ね３つの世代に分けてとらえることができます。

第１世代：

1970 年代は、高レベルプログラミング言語が数多く提唱されるとともに、プログラムの規模の問題を扱った複合設計法や段階的詳細化法に基づく技術が整備されました。この時代に、ソフトウェアの構成単位である抽象データ型やモジュールの概念も整備されました。

第２世代：

1980 年代は、ここではオブジェクト指向開発技術に代表されるように生産パラダイム自身が見直され要求定義やプロトタイピング技術が提唱されました。ソフトウェアというものが単純なプログラムという対象から、より複合的なシステムという捉え方に変わり、この時代でようやく「仕様」という概念が実務レベルでも定着しました。顧客・ユーザの論理と、開発者との論理とを繋ぐ基本的な技術か確立したと言えるでしょう。

第３世代：

1990 年代以降は旧来の生産技術は熟成の域に達するとともに、分散システム向きの新しい計算モデルや、コンポーネントウェアに代表される広い意味での知識の再利用技術が台頭して来ています。実際の開発プロジェクトでも従来のウォータフォール型の生産パラダイムにとらわれつつも、チーム制やインクリメンタル開発等を導入し急変するソフトウェア市場に対応しようとしています。これ等の動向は、標準化やオープン化といった社会的な仕組みが急変し、ビジネスに直接影響を及ぼすようになったところに技術が集約されているとみることができるでしょう。

プロセス／プロダクトの技術開発の重点化という視点で言うと，低粒度から高粒度へと移行しつつ、開発プロセスの局面とプロダクト（中間成果物）の局面とが繰り返し波打つように発展してきているように思えます。

産業構造の変化

産業論上の経緯からみますと、図４に示すように、電子デバイス（ハードウェア）の領域とユーザ（顧客）領域とのギャップが年々広がってきているとみなせます。すなわち、モノリシックで未分化であったソフトウェア全体が、分化して来たと同時に、ユーザがやっていたことをソフトウェアが代替えするようになったということであり、社会の中でのソフトウェアの役割が拡大してきたことを意味していると言えるでしょう。

初期：

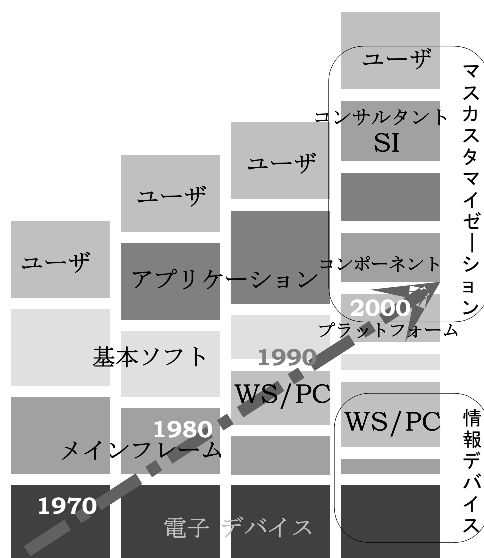

図4 産業構造の変遷

かつての大型機（メインフレーム）の時代では、電子デバイス（ハードウェア）とユーザ（顧客）との間には、メインフレームのOS（オペレーティングシステム）と基本ソフト（コンパイラやデータベースシステムなど）があればよく、これ等の開発に注力していました。この時代では、ハードウェアの知識を持つ者と、ソフトウェア（あるいはプログラミング）の知識を持つ者が協調する程度で済んでいました。

中期：

時代とともにユーザの要求も多様化し、アプリケーション（ユーザの要求する機能をそのドメイン／分野で実現するためのソフトウェア）が成長しました。その後、WS（ワークステーション）やPC（パソコン）が台頭し、旧来の汎用機／メインフレームの役割は小さくなりました。

近年：

最近では、ユーザの要求はますますエスカレートし、要求を満たすシステムを構築するためには専門のコンサルタントやSI（システム・インテグレータ）がいなくては対処できなくなって来ました。また、これと同時に、より大規模で複雑なシステムを効率よく構築するために、プラットフォーム（Windows, Unix, MacOSなど）が整備され、その上でコンポーネント（部品）を組み合わせる技術が台頭して来ています。1990年代に入ってからのインターネットの急速な普及とオープン化が技術開発を促進しており、1980年代に提案されたオブジェクト指向開発技術やオブジェクト管理技術が実用段階に入り、部品のコンポーネント化も急速に進んできていると言えます。このユーザとプラットフォームの間の技術は、一言で言うならば、マスカスタマイゼーション（ユーザの多様な要求を個別に・迅速に満たす技術）と呼ぶべき領域であり、これはいわゆるミドルウェアと呼ばれている領域に相当しています。

ビジネスの視点からユーザ領域の変化も激しく、価値観も多様化してきています。ユーザの世界にある問題を計算機や周辺機器等のハードウェアを使って解いたり、支援したりするのがシステムの目的ですから、ユーザ領域にある要求や問題の構造と、ハードウェアの構造とを原理的につなぐためにソフトウェアの仕組みがますます重要になってきています。ここで言うマスカスタマイゼーションは、構造変換を迅速に行なうための系統的な方法を確立するための一つの方向として今後も重要と思われます。

一方、電子デバイスも高機能化し、WSやPCの機能、さらには、インターネットの機能などが埋めこまれた装置（デバイス）が台頭して来ています。これは、旧来の組み込み型のシステムや情報家電といった分野の発展形としてとらえることができます。この電子デバイスの発展分野は一言で言うならば、情報デバイスの分野と呼ぶことができます。特に、こういった装置産業に直結した領域は日本の得意とする領域であり、諸々のインパクトを与えることができる領域です。

技術の難易度

ソフトウェアそのものの技術も図５に示すように、当初の簡単な科学技術計算のレベルからより大規模で複雑なものへ、さらに、社会的な意味での重要性も大きくなってきています。

静的構造：

プログラムの制御構造（ブロック構造、条件分岐構造、くり返し構造）やデータ構造（構造体、配列構造、リスト構造）にだけ注意をはらえば、そこそこのソフトウェアが作れるという時代がかつてありました。

情報システム：

計算機というものが科学技術計算だけでなく、情報を扱う機械という位置付けが定着し、これに伴って情報システムの構築方法（組織のミッションや、扱っている情報から計算機システムを構築する方法）が確立されました。

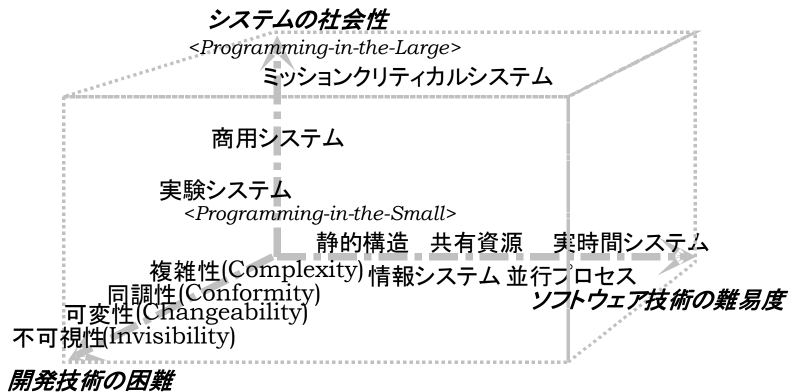

図5 システムへの要求と技術課題の変遷

共有資源：

計算のメカニズムもより複雑になり、同時にいくつかの資源（記憶領域や周辺装置など）を共有したり、これ等の資源を扱うプロセスを排他制御する機構が確立されました。

並行プロセス：

いくつかの計算や処理を行なうプロセスが同時に複数実行される並行プロセスの研究が進み、より複雑な問題を扱えるようになって来ました。

実時間システム：

対話型システムや、オンラインシステム、さらには，制御システム（FA：Factory Automation や航空管制，航空機制御）といった時間に依存した概念も直接扱えるようになり、より広い範囲の問題を扱えるようになって来ました。

こういったソフトウェア技術の難易度に関して、ここ数十年の間にわたって背景となる基礎理論の整備が進められてきていますが、並行プロセスや分散処理、実時間システムを基礎づける理論はまだこれからの課題であると言えるでしょう。

こういった技術展開を代表していると言えるのが天才M.A.Jackson の仕事です。Jackson は、1975 年にJSP：Jackson Structured Programming を発表し、構造化プログラミングの３つの基本構造（連接、選択、繰り返し）に基づく、実行単位としてのプログラムの仕様と実現との関係を明らかにしました。次に、1982 年には、JSD：Jackson System Development という開発技法を提唱しています。JSD は、JSP の発展させたもので、プログラムの総体としてのシステムを開発するための技法です。この技法によって動的な側面（振る舞い）についての仕様化を基本に置いており、基礎的なモデルは CSP:Communicating Sequential Process という並行プロセスモデルを使っています。さらに、1995 年に『問題フレーム』という数学における問題の解き方をソフトウェアに援用する方法を提唱しています。これは、近年のデザインパターンの考え方に通じるものがあります。

抽象度・粒度と再利用技術

システムは、広い意味で、抽象的な機械とみなすことができます。計算機というハードウェアは、機械語という命令体系を提供している汎用の機械です。より多くの高レベルの機能を提供する機械を構築するためには、抽象度の高い命令体系を持った抽象的な機械を積み重ねて行く必要があります。プログラミング言語も命令体系を提供している低レベルの抽象機械です。非常に単純化して述べるならば、プログラミング言語の言語要素をまとめて記述することによってオブジェクトが得られ、さらに、オブジェクトをまとめたものがコンポーネントとすることができます。システムのアーキテクチャは、コンポーネントの総体としてまとめることができます。いわゆるミドルウェアと呼ばれている領域は、フレームワークやコンポーネントといった中間層の抽象機械に相当しており、より大規模で複雑なシステムを開発するためには、洗練化されたものを

開発しておく必要があります。

オブジェクト指向技術(OOT:Object-Oriented Technology)は，ソフトウェアの再利用性や理解容易性を高める技術として近年急速に発展してきたものです。開発工程の中でもプログラミング工程に対応したオブジェクト指向プログラミング（OOP:Object-Oriented Programming)、設計工程に対応したオブジェクト指向設計、（OOD: Object-Oriented Design)、分析工程に対応したオブジェクト指向分析（OOA:Object Oriented Analysis)というように全体に渡って技術開発が進んできています。オブジェクト指向開発技法は，多くの流派がありますが，最近では徐々に淘汰され，Rational 社の提案した表記方法（UML: Unified Modeling Language）がデファクトになりつつあるようです。

システムの開発支援環境の視点は、システムの構成要素の抽象度や粒度とともに発展してきています。旧来のプログラミング言語レベルでは、コンパイラ（言語の処理系）を中心とした支援環境が中心でした。コンパイラは、エディタやライブラリやデバッガ等を一体化した統合環境に移行し、さらに、モジュールの内部ロジックと外部インタフェースとを分離して管理できるような設計支援環境へと発展して行きました。旧来このコンパイラが単体で提供されていた。最近では、この統合環境の考え方をさらに発展させ、ライブラリそのものを開発したり、ライブラリや部品を組み合わせてさらに大きな部品を開発・利用できる環境が整備されて来ています。再利用可能な部品としてのコンポーネントとして最初は GUI(Graphical User Interface) で使い易いものが普及しました。その後、データベースやクライアントサーバシステム、さらには、分散システム向きのコンポーネントを搭載した開発環境が提案されてきています。コンポーネントの標準化や信頼性を上げる努力も地道に行われて来ており、旧来の個人的で小さなソフトウェアを対象としたものから、企業システムレベルで利用可能なものも整備されて来ています。

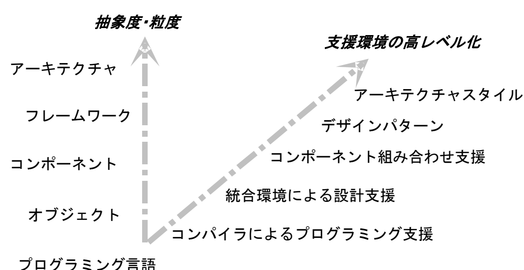

図6 再利用の単位と抽象度

また、設計や開発という本来の人間の知的創造活動を焦点にあてたアプローチとしてデザインパターンの考え方が有効です。これは、オブジェクトの組み合わせ方の雛形を、設計過程で現れる問題とその解という図式でとらえるものです。デザインパターンをより高粒度のレベルで行なうアプリケーションフレームワークやアーキテクチャスタイルといった開発ノウハウも、ドメインを特化した形で実用化されて来ています。

システムの構成単位の抽象度や粒度という見方と、支援環境のレベルとは密接に関係しており、最近では、OS やプラットフォームレベルでも従来のプログラミング言語レベルでの仕掛け（コンパイラの機能）が組み込まれてきています。

パターン技術の系譜

パターン技術に対する取組みは、広範囲にわたっており図７に示すように、理論面から実践面まで、汎用のものから特定の領域を対象としたものまで、さまざまなアプローチが試みられています。この図では、オブジェクト指向技術を中心に据えており、建築分野の C. Alexander の『パターンランゲージ』は対象領域が異なるものの特定領域の実務知識の集大成として位置づけることができます。「技術の難易度」の節の最後で述べた M.A. Jackson の『問題フレーム』は、パターン技術の伝道師の人々はあまり言及しなませんが、ソフトウェアとそれを基礎づけている数学の世界とを結びつけ、かつ、豊富なコンサルティングの経験や哲学的な思索も加えて一般的ではあるものの実用性の高い方法を提案しています。並行プロセスやリアルタイムシステ

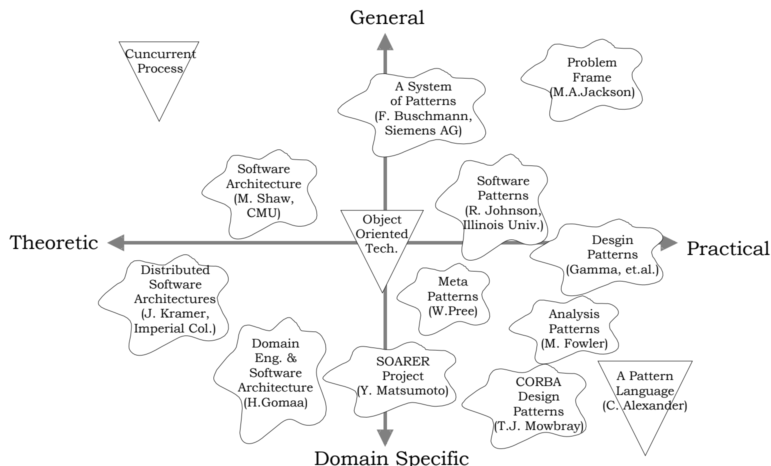

図7 パターン技術の系譜

ムに関する技術は、未だ学術領域での理論開発が盛んな分野であり、現状では一般的で理論レベルでの提唱に止まっています。

M.Shaw 等による『ソフトウェアアーキテクチャ』やR. Johnson の『ソフトウェアパターン』は、パターン技術の思想的な基礎付けを行ない、GoF の『デザインパターン』は、自らの実務上のコンサルティングの経験からオブジェクト指向に関する 23 個のパターンを具体的に提案しています。これ等は、パターン技術の先駆的でかつ啓蒙的な業績と言えるでしょう。オブジェクト指向より一般的なパターンについては、その粒度の概念とともに整理をしたF. Buschmann 等の『パターン体系』があります。パターンそのものを導出する原理を示したものは未だ多くはありませんが、W. Pree の『メタパターン』の《ホットスポット》、《フローズンスポット》の考え方は説得力があります。

オブジェクト指向を中心としたパターン技術は、大きく２つの方向性が出てきています。一つは実装面、すなわち、Gof の『デザインパターン』に代表されるようなオブジェクト指向に基づく開発環境やコンポーネント、プラットフォームに特化した開発者の論理の中でパターンを開発して行くアプローチです。もう一つは、顧客との関係を主体にした『アナリシスパターン』の領域です。M. Fowler の『アナリシスパターン』では、ビジネス情報系についてオブジェクト指向の手法を使って業務分析を行ない構築すべきシステムの実務世界でのあり方を《ビジネスモデル》として構築して行く方法を提唱しています。

特定分野向けのパターン技術としては、ビジネス情報システムを主対象とした T.J. Mowbray 等の『CORBA デザインパターン』、リアルタイムシステムの分野を対象としたH. Gomaa の『開発技法』や、その技法や組み込みシステムの製品開発から設計知識を抽出して基盤整備を進めている『IPA-SOARER(SOftware Architecture Repository for Embedded and Realtime systems) プロジェクト』等があります。特に、リアルタイムシステムや分散システムに関しては、理論整備とともにアーキテクチャの提案をして行く必要があり、J. Kramer 等の「分散アーキテクチャ」の研究は並行プロセスを中心にすえた実務的なパターン整備を狙っています。

技術展望と課題

正しい技術が社会の中で普及するとは限らないですし、その発展が連続的であるという保証もありません。しかしながら、趨勢と大局的な方向性という視点からの課題を整理すると、ソフトウェアエンジニアリングに関する課題は次のように整理することができます。

(1) 高度再利用技術＝規模の問題：

LSI の集積度が年率 50～60％で上がる一方で、ソフトウェア開発の生産性は年率 10％程度の伸びに留まっています。規模の問題を解決するためには、原理的にソフトウェア開発時の抽象度を上げ、再利

用率を高める解決方法しかあり得ません。コンポーネント技術という視点からもより大きな粒度、あるいは、抽象度の高いものを部品として蓄積する技術（コンポーネントを作る技術）が必要ですし、部品を適確に組み合わせて顧客にソリューションを提供するマスカスタマイズの手法（コンポーネントを組み合わせてアプリケーションを構築する技術）も、人間のコミュニケーションや企業モデルを扱うことができるより要求工学的なアプローチを取り入れ、さらに、ハードウェア・ソフトウェア、ないし、プロダクト・プロセスを統合化したもの（コデザインプロセス）に進化して行かなくてはなりません。

⑵ 高度高品質化技術＝信頼性の問題：

ユビキタスコンピューティングを始めとして社会のあらゆる分野にソフトウェアが普及し、組込まれて行くと、従来のミッションあるいはセイフティクリティカルなシステムのみならず、あらゆる分野のソフトウェアに対する社会的な責任も重要になってきます。現在の技術では、そのソフトウェアの品質を保証する技術も未成熟であり、理論的な整備もフォンノイマン型の計算モデルの上でさえ完全とは言えません。ソフトウェアの規模の増大に伴って、ソフトウェアの構成部品や提供サービス一つ一つにかけることのできる費用も限度があり、開発プロセスや品質そのものの考え方に対してもブレークスルーが要求されています。形式的仕様記述や証明技術、計算の理論についてもソフトウェアエンジニアリングの視点から総合的に見直す必要があると言えるでしょう。私はこの領域のことを『精密ソフトウェアエンジニアリング(Precise Software Engineering)』と呼んでいます。

セキュリティやプライバシなどの基盤技術，標準化と標準に対するコンフォーマンスの問題などもその知識蓄積の上で取り扱い、ノウハウ（パターン）としてまとめて行かなくてはならない重要な課題です。

⑶ 多次元オープン化＝社会性の問題：

メインフレームレベルの技術から、組み込みシステムのレベルまで一貫したプラットフォーム整備を図る技術は、産業構造における水平・垂直型の開発役割モデルの視点から有望です。Windows やUNIX 等のオペレーティングシステムも、企業の基幹業務レベルのものから民生用の組込み製品レベルのものまで、必要に応じてオペレーティングシステムそのものがコンポーネント化しカスタマイズ可能なものになって行くでしょうし、こういったシリーズに適合した生産支援環境の開発も必要です。また、近年のオープン化の契機となっているインターネットの普及に見られるように、デジタルコンテンツそのものの拡大もソフトウェアと同期しており、ソフトウェアと情報との融合化も今後の大きな課題の一つと言えます。

以上のように、ソフトウェアエンジニアリングの技術というのは、ハードウェアやオペレーティングシステムレベルと、顧客へのソリューション提供という今後ますますギャップが広がってゆく２つの世界を繋ぐところにあり、原理的には抽象的な機械を構築し積み上げて行く技術（デザインパターン的な側面）と、顧客という人間世界の欲求・願望・要求や組織のミッションという社会的コード・規範（アナリシスパターン的な側面）を同時に視野に入

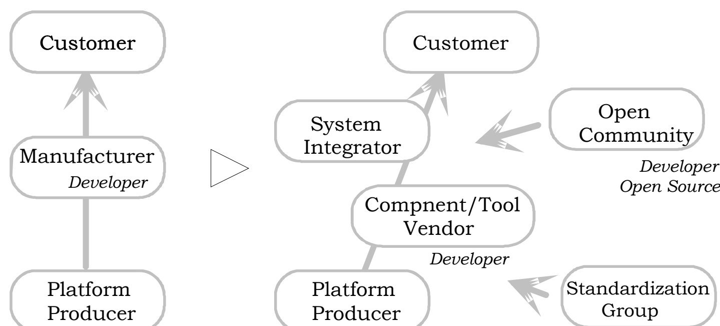

図8 ソフトウェア開発における産業構造の変化

れなくてはならないでしょう。そして、パターン技術の範囲は、旧来のメインフレームやワークステーションで培われてきた技術をダウンサイジングしたり、カスタマイズして分野をまたがって適用して行く広義の開発知識の再利用技術と、新たなる人間の要求に答えるべくより抽象度の高い機械を提供して行く方向性とがあります。システムの価値や、開発コストといった考え方自体が、旧来の物理的な工業生産物のパラダイムに基づいた所有権や交換経済の枠組みではとらえることができないということであり、新しい社会的なコード（規範）を組み入れたものになって行くことを示唆しています。私は、この領域を『記号論的ソフトウェアエンジニアリング(Semiotic Software Engineering)』と呼んでいます。

ソフトウェアエンジニアリングは、ソフトウェア開発にかかわる知識を、供給者（生産者）・利用者（消費者）といった人々が関わる情報流通・蓄積を行なって行くための基本的な技術として位置づけることができます。従来のソフトウェアエンジニアリングの成果というのは、技法や開発環境（ツール）といったどちらかと言うと開発ノウハウをガイドブックやアプリケーションプログラムという形で静的に閉じ込めたものでした。しかし、近年のソフトウェア開発プロジェクトでは、より市場性が求められ、ダイナミックに変動する社会の中で、適確な開発ノウハウとしてのソリューションをいかに得るかということが課題となっています。従って設計ノウハウを持っている人間と、それを使う人間とを旨く結びつけて行くことや、システムの提供者と利用者との共通了解を得る仕掛けを構築することがソフトウェアビジネス振興の要となります。いわゆるコンポーネントの開発に関しても、占有型の商用ソフト開発のみならず、オープンソース型の重要性も増してきています。こういったことは、近年のネットワーキングの発達によるところが大きいですが、図８に示すように、知的財産を共有化して行く新しい社会システムが台頭してきています。こういった産業構造の変化は、従来のメーカという位置づけでシステム開発を行なって来た組織が衰退し、ソフトウェアそのものの開発は優れたデザイナによる専門家集団（コンポーネントベンダやオープンソース供給）が行ない、顧客へのソリューション提供はシステムインテグレータやコンサルタントが行なうようになってきていることを意味しています。

〔この記事は、（社）日本電子工業振興協会「先端ソフトウェア技術に関する調査報告書」（2000 年 3 月）の執筆内容に基づき、加筆・修正したものです。〕

（おおつき　しげる）