ソフトウェアエンジニアリングの発展シナリオ
玉井哲雄

たまに SeaMail に投稿するのは，他の目的で書いた原稿の再利用ばかりで申し訳ない．柄にもなく表題のような未来予測的な文章を書いたのは，文部科学省科学技術政策研究所というところから，「注目科学技術領域の発展的シナリオ調査」に協力し，ソフトウェアエンジニアリング領域の今後 10 年から 30 年の発展シナリオを自由に書いてほしい，という依頼があったからである．1 年先のことも分からないの，10 年先などとんでもないし，まして 30 年先の予想など冗談でしかないと思ったが,それだけに勝手なことが書けるかとつい引き受けた．それが 2004 年 9 月のことだから，いまやかなり古い話ではあるが，なにしろ 30 年先だから，1-2 年の経過はあまり関係ないともいえる．きわめて短期間で書いたので，正直なところ，内容は思いつくまま書き散らした，という感がある．
この報告書は 2005 年 5 月に刊行されていて，その概要は
http://www.nistep.go.jp/achiev/ftx/jpn/rep096j/idx096j.html
で見ることができる．この執筆に際し，文章を他で利用してよいか尋ねたところ，構わないとのことだったのでここに投稿する次第である．

(1) 現状分析

[ソフトウェアの需要]

ソフトウェアの需要は増大し続けている．経済産業省の特定サービス実態調査では，情報サービス産業の内のソフトウェア開発/プログラム作成の売上高は 2003 年度で 6 兆 7 千億円である．しかも，これはソフトウェア専業者の統計で，コンピュータ・メーカやコンピュータ利用企業で生産・消費されるソフトウェアは含まれていない．たとえば組込みソフトウェアはほとんど含まれていないが，その年間開発規模は別の統計から，約 2 兆円といわれている．
また，現在のソフトウェア市場の顕著な動向は，開発サイクルの短期化である．新しい機能が次々と求められ，それらの開発には数週間から数ヶ月という短い期間が割り当てられる．

[社会生活に及ぼす影響力の増大]

現代の生活の周囲にはソフトウェアが満ち溢れている．携帯電話だけでなく家庭内にあるあらゆる電気製品には膨大なソフトウェアが組み込まれている.またクルマには数 10 個のマイコンが積まれ，そこに載せられているソフトウェアは合計すると数百万行の規模に達する．Web サービスに代表されるインターネットの職場や家庭内での利用の増大も，大量のソフトウェアの開発に依存している．
このことは，ソフトウェアの信頼性や生産性の欠陥が一般社会生活に及ぼす影響力がきわ

めて大きなものであることを意味している.

[ソフトウェアの生産性]

ソフトウェア技術は進歩したが，生産性の伸びはハードウェアと比べれば低い．この 30 年で，ハードウェアの伸びは 50,000 倍であったのに対し，ソフトウェアの伸びは 3~10 倍程度であったといわれる.ソフトウェア開発に多くの手法とツールが使われるようになり，プロジェクト管理技術も発達したが，一方でソフトウェア開発が労働集約的であるという実態は，相変わらず続いている．とくに携帯電話市場など，激しい競争条件下にある分野では，開発量の増大はソフトウェア技術者の長時間労働で対処されている.

[日本のソフトウェア産業]

日本のソフトウェア産業は，上からと下からの技術の空洞化という問題点を抱えている．基本ソフトウェアはマイクロソフトに代表される米国企業の製品に押さえられ，その追随に終始せざるをえない状況がある．一方で，定常的に生産されるソフトウェアは，中国，インドなどへの外注（アウトソーシング）への依存度が高まっている.

日本のソフトウェアは輸入が輸出を上回る入超が常態化しているが，その超過量は圧倒的である．従来，日本語の壁で国内市場をある程度守ってきた経緯があるが，それは裏返して言えば,日本のソフトウェアの輸出を妨げ,輸出意欲を発達させない要因となっている.

[ソフトウェアエンジニアリグ技術]

ソフトウェアエンジニアリングという概念は 60 年代末に提唱され，70 年代は構造化プログラミング，構造化分析・設計を主導概念として発展した．80 年代はプロジェクト管理，品質管理，構成管理などの管理技術にシフトする一方，プロトタイピング，レヴューなどの実践的技術が普及した．90 年代はオブジェクト指向技術が脚光を浴びるとともに，並行してソフトウェアプロセスのモデル化や評価が注目された.2000 年代はプロセスからアーキテクチャへの関心の回帰が見られるとともに，コンポーネント技術などオブジェクト指向を発展させた新たな技術が展開してきている.

[現状のソフトウェアの性質と課題]

現在のソフトウェアの特徴は，遍在性（どこにでもある），発展性（常に変化する），多様性（多様な版とユーザが存在する）にある．ソフトウェアエンジニアリングは伝統的に大規模複雑化への対処ということを謳い文句にしてきたが，現在はこのような遍在性，発展性，多様性にいかに対応しつつ，信頼性を高め生産性を上げるかが課題となっている.

(2) 今後 10～30 年程度の発展シナリオ

[ハードウェア技術の予測]

ソフトウェアが載せられるハードウェアの情況をまず見ると，現状はギガ(10 の 9 乗)の世界である．すなわち，CPU はギガヘルツ，メモリはギガバイト，通信速度はギガビット/秒(bps)というオーダーの性能を持つ．これが，2015 年にはテラ（10 の 12 乗），2030 年にはペタ(10 の 15 乗)の世界になるだろう.

[ソフトウェアの需要]

過去のコンピュータの歴史では，ハードウェアの能力・容量が飛躍的に増大するたびに，それに見合う需要があるのか，との疑問が常に出されてきた．しかし，いつの場合もそれを上回る需要が生み出される結果となった．

したがって，今後 10 年から 30 年の間についても，需要が拡大し続けることはほぼ自明である．

[ソフトウェアエンジニアリングの技術]

(a)大量なコンポーネントのもたらす複雑性への対処

すでに相当量のソフトウェア・コンポーネントが作られ，一部流通しているが，2015 年までには膨大な量のコンポーネントが投入され使用されることになろう．そのような大量のコンポーネントから構築されるシステムは,複雑性のボトルネックに直面することになる．コンポーネントは，さまざまな技術者，組織により，長い期間にわたり，高い頻度で追加され変更されていく．そのようなコンポーネント群から，適切なものを検索し選択する技術，それをカスタム化し組み立てる技術が必須となる．

その検索，選択，変更，組立てをすべて人手で行うことは不可能となるため，なんらかの知能化，自動化が工夫されなければならない．具体的には

1. コンポーネントに知識を付加し，エージェント化
2. ノウハウとしての設計パターン，要求パターンなどの部品化と知能化
3. コンポーネントの擦りあわせの自動化
4. 人とエージェントの共同

が行われていくだろう.そして 2015 年前後にはかなりのレベルで実現していると予想される．

(b)ソフトウェア開発の自動化

ソフトウェア開発の自動化は古くから追求されてきたテーマである．上に述べたコンポーネント技術の知能化，自動化をベースに，2015 年には，モデルからのソフトウェアの自動生成が実用的なレベルで実現しているだろう．ただし，生成のもととなるモデルは，システムの構造や振舞いについて，ある程度の詳細を記述したものである必要がある．

さらに，2020 年ごろには，ソフトウェアに対する要求，ポリシーからソフトウェアが自動生成されることになろう．さらに 2025 年ごろには，問題領域で普通に使われる言葉からの自動生成が可能となろう．

(c)自律性をもち環境に適応して自己組織化を行うソフトウェアの実現

ソフトウェアは常に環境に適応することを求められる．環境にはビジネス環境やシステム環境があり，それらが変化すればソフトウェアも変わらざるをえない．また，ソフトウェアが環境間を移動すれば，それに適応してソフトウェア自身が変化しなければならない．さらに，新しく開発されるソフトウェアも，独立に存在するものでなく，すでにある環境に投入される．

これらの適応変化は，これまでは基本的に人間の手でなされてきた．しかし，ソフトウェア自身が自律的に適応し，また自己組織化する技術が今後 30 年の間には進展するだろう．その発展には 2 つの段階が予想される．

1. 動かしながら変える（動的な変更） 2010-2015 年ごろ
   一部は自動化されても,変化を起こす主体はあくまでも人間であるような適応技術は，今後 5 年から 10 年の内に充分に実用化されよう．しかし，これは稼動しているシステムを停止せずに行うという意味で，見かけほど簡単ではない．従来，コンパイル時やシステム生成時に行われてきた，システムの変更，追加，部分削除に関する技術が，すべて実行時に移行していく．この範囲では，とくに事故，故障に際してのソフトウェアの自己診断とそれにもとづく自己修復の機能の実現が達成されよう．
2. 動きながら変わる（動的な変化） 2030 ごろ
   ソフトウェア自身が,自己と外界を知りそれに応じて自己を変化させる能力を備える．また，利用者のフィードバックが直接ソフトウェアを変えていくようになる．この段階で初めて，ソフトウェアの自己再生産，自己組織化ができるようになったと見なせよう．

(d)検証技術

ソフトウェアの信頼性，安全性，セキュリティを保証するための検証技術は，数段のレベルの向上が要求される．とくに上記の自己適応技術と関連し，自律的に変化したソフトウェアが正しく動作することも自動的に検証できなければならない．適応技術の発展段階に応じた検証技術の発達が予想される．

[ソフトウェアエンジニアリングの理論]

上に述べた技術を実現するためには，その基盤の理論的強化が必要である．まず，すでに理論としてはかなりの成果が挙げられているが，その実用化は今後に期待されるものがいくつかある．たとえば，関数型プログラミング，論理型プログラミングのような明快な計算モデルに基づくプログラミング技術は,5 年後か 10 年後に何かのきっかけで急速に普及することがありうる．また，ソフトウェアの仕様技術の一つとして，形式仕様記述言語を用いることは，分野を特定しながら徐々に普及していくだろう．さらに，定理証明技術による自動検証は，形式仕様技術と関連しながら，実用化されていこう．

技術の項で挙げた大量なコンポーネントからなるシステムの複雑性に対処していくには，エンジニアリグとして個々のコンポーネントと全体のアーキテクチャを設計し管理するだけでは不十分であることが予想される．そこで，人工物であるソフトウェア･システムを，あたかも自然物のように観測，分析する手法が求められよう．そこでは，統計力学に相当するようなマクロな分析手法と，その分析結果をソフトウェアのアーキテクチャやコンポーネントのミクロレベルにフィードバックする方法論が求められる．

そのような「科学的」アプローチとして，より大きな理論的枠組みを想定するとすれば，以下のようなものが考えられよう．

1. 連続時間の導入（2015 年ごろ）
   計算は離散的な量を対象とするため，本質的に連続量である時間の扱いがこれまで困難であった．もちろん，応答型システム，実時間システムなどは時間を意識したシステムであるが，時間を理論的に扱うにはやはり離散化して対応するのが常であったと言える．しかし，ニュートン力学は連続時間を扱い，微分積分の概念を導入することで美しい理論をうち立てることができた．ニュートンの方程式に相当する計算理論の構築が求められるゆえんである．
2. 連続空間の導入（ 2030 年ごろ）
   地球規模の互いにリンクされたおびただしい量の計算資源は，空間連続体として捉えることが可能である．これを理論的に扱うにはマックスウェルの電磁方程式に相当する時空間の計算理論が求められよう．

[他分野に学ぶ]

今後のソフトウェアは，計算機科学のみならず他の学問分野から学び，技術として発達させていく必要があろう．

1. 生物学
   すでに挙げた環境適応，自己組織化という概念からして生物学から導入されたものである．さらにたとえば，免疫系（とくにウィルス攻撃への対応などのセキュリティ対策），反射系と中枢神経系と脳との役割分担(とくにロボットソフトウェア)，などが有用であろう.また,すでに述べた大量なコンポーネントがネットワーク上に生息し，生成，変化，分化，消滅を続けている状況の分析には，生態学的アプローチも有効であろう．
2. 文献学，考古学
   ソフトウェアも一種の文書であり，それを歴史的なパースペクティブで文献学的に研究することは，新たなソフトウェアエンジニアリグ技術を発見するためにも有効であろう．
3. 社会学，文化人類学
   ソフトウェアの開発組織はきわめて人間的な組織である．その分析，改善，研究には社会学や文化人類学の手法が有効であろう．

(3) 作成した発展シナリオを踏まえた日本のとるべきアクション

[教育，人材育成]

アクションの第一は教育，人材育成である．とくに次のような世代の特徴を活用することが有効であろう．

1. 生れたときからデジタル環境に育った世代のセンスの活用
   これまでのユーザインタフェースの研究と実践は，機械を使い慣れないユーザにいかに使いやすいインタフェースを提供するか，ということであった．しかし，これから

就業年齢に達していく世代は，生れたときからデジタルな機器が身の回りにふんだんにある環境に育っている．そのような世代が使用し，また開発するインタフェースはこれまでとは異なるセンスのものになるだろう．それを積極的に育てていくべきである．

また，不法アクセスなどの問題を起こす「ハッカー」は，当然若い世代が多いが，彼らの多くは高い技術と関心をもっている．その力を創造的な方向へ向かわせることが望ましい．それには，情報倫理教育も「しちゃだめ」でなく創造の芽を育てるようなものとしなければならない．

2. 熟年世代の経験と時間の活用

   団塊世代は情報科学/工学教育を受けた最初の世代であり，またその後多くが，業務での本格的なソフトウェア開発経験を豊富に持つことになった．今後，その世代以降が次々と定年に達するが，その経験と技術を生かす方策が有用であろう．その方向は，楽しみの開発と楽しみながらの開発ということになるのではなかろうか．

3. ソフトウェア技術者の地位向上

   現状では，計算機科学分野を志望する学生の数が米国でもヨーロッパでも減少傾向にあり，問題となっている．日本ではむしろかなり以前からその傾向が見られた．将来のソフトウェアの需要から考えると，この事態は深刻である．これに対処するには，ソフトウェア技術者の成功物語，ヒーローの誕生が必要である．より卑近な策としては，ソフトウェア技術者の作業環境を改善し，処遇を高めることが望ましい．

4. デザイナの役割増大

   WWW ページに代表されるように，コンテンツ創造とソフトウェア開発の一体化しつつある．そこで広い意味でのデザイナの役割が増す．そのためにソフトウェア技術に高い素養があるデザイナの育成が肝要である．

5. 他分野の専門家であると同時にソフトウェア技術者であるような人材の育成

   生物，物理，化学，アート，経営学などの分野でエキスパートでありながら，片手間でなくきちんとしたソフトウェアエンジニアリング教育を受けた人材がますます必要となる．

[研究開発投資]

研究開発投資にはさまざまなテーマがありうるが，以下いくつかに絞って記述する．

1. 知的財産処理の仕組み

   知的財産権というととかく権利保護が全面に出がちだが，オープンソースのような自由な開発形態を阻害しない工夫も必要である．一方で，知的財産権で保護されているコンポーネントやソフトウェアを正当な対価を払って簡単に使える制度，システムを構築すれば，権利者，利用者ともに便益がある．

2. 基盤研究の重視

   UNIX，関係データベース，オブジェクト指向技術などに見られるように，ソフトウェ

アの研究成果が広く普及するには10年から20年以上の年月を必要とすることが多い．そこでやはり基盤的な研究を助成することは欠かせない．研究評価は重要であるが，その評価も長い期間にわたり追跡できる体制が必要である．

3. 国際交流

IT 技術はもっとも国際的な分野である．途上国でもパソコンを 1 台買えば，最先端の技術にアクセスが可能となる．そして，今日開発された技術は，明日には全世界に拡がる．したがって，欧米とだけでなく，アジア，南米など多くの地域との人的交流が，今後ともますます重要となろう．そのためには，他の文化の人々と闊達にコミュニケーションができる人材が求められる．このことは，人材育成のあり方にも密接に関連してくることである．また，交流を促進する資金，制度が求められる．

[産学によるソフトウェアの大規模な実験の場の構築]

ソフトウェアは元来，実験が難しい．それは，ソフトウェア開発に人的な要素が大きく，実験をコントロールするのが難しいためである．企業は競争的な開発に追われ，実験をする余裕がない．一方，大学にはアイディアをスケールアップして試す場とリソースがない．たとえばセキュリティ関連の技術は実験が有効であるが，計画，管理が難しい分野である．そこで産学が共同して大規模な実験を行えるような場を作り，そこに資金を投入して他では見られないような実験を実施すれば，新たなソフトウェアエンジニアリング手法のきちんとした評価がなされ，大きなインパクトを与えることができるだろう．