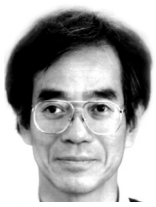
特集1

二つの合理性と日本のソフトウェア工学

客員研究官　林　　晋 *
客員研究官　黒川　利明 **

1. 始めに

日本人は論理的でないという、また、合理的でないという。それが日本のソフトウェア産業の構造的弱さと関係しているのではないか？　いくら重点的に予算を注ぎ込んでも日本のソフトウェア分野が浮上しないのは、日本社会が合理的・論理的思考法に弱く、また、それを軽視するためではないか？　日本のソフトウェア産業・ソフトウェア工学の「弱さ」を前にして、我々は、そう考えていた。

アメリカ社会やヨーロッパ社会にみられるような合理的思考法が身に付かない限り、日本という社会は、ソフトウェアの分野に限定すれば、アメリカやヨーロッパと同じレベルで競うことさえできないのではないか？　社会全体に、合理的思考法・論理的思考法が行き渡らない限り、この国のソフトウェア分野は永久に浮上しないのではないか。そして、この国が合理的・論理的になることなど、百年河清を待つようなことなのではないか。だから、日本のソフトウェア産業は容易なことでは浮上しないのではないか。これを打破するには、合理的思考・論理的思考を、学校教育に取り入れるしかなく、それは、日本をさらに西欧化・アメリカ化することであろう。

最近まで、我々は、そう考えていた。しかし、その「本家」アメリカで、この考え方を覆すような事態が起きている。失われた10年の間に、世界の最先端を行くアメリカのソフトウェア・コンサルタントたちが、トヨタ生産法のような「日本式」を、ソフトウェア工学の手法に取り入れ始めたのである。

「合理性」は、決して、西洋社会、特にアメリカ社会の専売特許でもなければ、唯一正しい合理性の形態、絶対の合理性、とでも言うようなものがあるのでもない。多くの文化があるように、多くの合理性が存在する。アメリカが発見した「日本式」は、その合理性の一つだったのである。そして、今、それが急速に変化していくビジネスに対応するために、ソフトウェア工学の技術として不可欠なものになりつつある。

このことを深く理解して利用すれば、日本のソフトウェア工学、ソフトウェア産業のレベルが飛躍的に高まる可能性がある。現在は、日本のソフトウェア工学を世界の水準に高める千載一遇のチャンスである可能性が極めて高いのである。このチャンスを逃してはならない。

2. 日本のソフトウェア技術力

ソフトウェアという言葉は、マンガ、アニメなどのポップカルチャーから、コンピュータ・ソフトウェアまで様々な意味に使われる。日本は、マンガ、アニメなどの競争力は群を抜く。しかし、ゲームソフトなどポップ・アート的要素の強いものを除外した、主に業務用のソフトウェアと、その生産技術に限定すれば、日本のソフトウェア技術力は極めて弱い。本論文では、「ソフトウェア」と「ソフトウェア技術力」を、この日本が弱い分野に限定する。日本の産業・技術政策として、ソフトウェアを考えるとき、もっとも問題をはらんでいるのは、この分野であり、また、産業規模も大きいからである。

この意味での、ソフトウェア産業、および、それに必要とされるソフト

* はやし すすむ　●　神戸大学工学部 情報知能工学科 教授　●　http://www.kobe-u.ac.jp/
** くろかわ としあき　●　株式会社 CSK e ソリューション技術本部技術調査部基盤技術グループ CSK フェロー　●　http://www.csk.co.jp/index.html

ウェア工学は、さらに2つに分類することができる。本稿の分析を政策のレベルに応用する場合、この分類を意識することは重要である。それぞれに異なる人材を必要とするからである。しかし、この分類を技術の観点から説明するのは案外に難しい。そこで、これを産業形態の観点から説明しよう。

2-1 ソフトウェア技術力の2分類

MITのCusumanoは、ソフトウェア企業のモデルをMicrosoft、Adobeのような「ソフトウェア製品企業」（products company）とIBM、NTT Dataのような「ソフトウェアサービス企業」（service company）に分類している[4)]。大雑把に言えば、前者は不特定多数の顧客が買うことを前提としたソフトウェアを作り、そのコピーを大量に販売することで成り立つ企業である。後者は、比較的少数の顧客に対して、それぞれの顧客の要望に応じて、ソフトウェアやシステムを構築することを仕事とする企業である。これはあくまでモデルであり、実際のソフトウェア企業の多くは、中間的だったり、双方の業態を兼ね備えているものである。しかし、ここでは、ソフトウェア構築という仕事を分類するために中間的ケースは考えない。

Cusumanoは企業経営の観点から、この分類を考えているが、技術力について考察する本稿で、この分類を持ち出したのは、これらの2つ企業モデルが必要とするソフトウェア開発技術が異なっており、それにより、ソフトウェア構築技術を分類できるからである。ソフトウェア製品企業のために働く技術者には、Excel、Java、Windows、Linux、Oracle、GNU…などの、基本ソフトから、ビジネス・アプリ、あるいは、ゲーム・ソフトなどの個別の「製品」としてのソフトウェアを開発する能力が求められる。経産省の「未踏ソフトウェア事業」の目標は、この意味でのソフトウェア技術力を持つ個人の発掘と考えることができる。この種の企業におけるソフトウェア技術者は、ソフトウェアを自動車や家電製品を開発するように作る。

一方で後者のソフトウェアサービス企業で働く技術者には、ウォーターフォール・モデル、スパイラル・モデル、アジャイル方法論、要求仕様工学などのソフトウェア開発方法論を身につけ、その方法論を元に、発注元の要望を要求として顕在化させ、低コストで、短期間の内に、高品質なカスタム・ソフトウェアを、各発注元のためにカスタム・メードし、また、それを管理・運営する能力が求められる。この種の企業におけるソフトウェア技術者は、ソフトウェアをゼネコンが建築物を建設するように作るのである。産業規模としては、この種類のソフトウェア生産の大きさが、前者の「製品」としてのソフトウェアの生産・販売を遙かにしのいでいると言われる。

この後者の種類の技術を考えるとき、もう1つの技術力を考慮する必要がある。アメリカのソフトウェア産業界では、ソフトウェア技術者たち自身が、独自のソフトウェア開発方法論を開発し、それが各企業の競争力の源泉となり、さらには、その技術を知識として販売するということが起きている。現在の世界のソフトウェア工学のスターたちの多くは、大学の研究者ではなく、このような立場のソフトウェア・コンサルタントたちであり、それが産業に直結しているのである。これは生産技術を「販売」する生産工学のコンサルタントたち、たとえば、トヨタから独立した「トヨタ生産法の伝道師たち」にあたる。この種類の技術は、直接、製品を作り出すのではなく、生産技術を作り出すものといえるので、産業規模としては小さいように見えるが、長期的には、これがソフトウェア技術の競争力の差を生み出す。

本稿では、この3種類の技術を、次のように（p型）と（s型）に分類する。

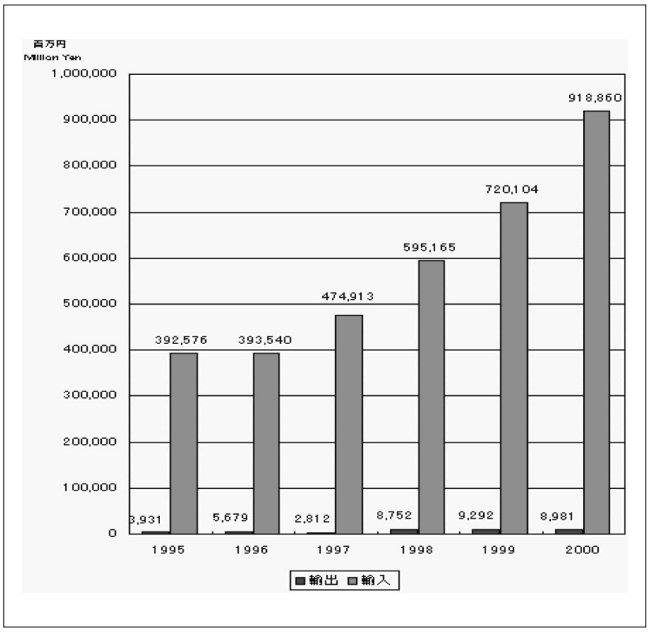
- （p型）「ソフトウェア製品企業」（products company）が必要としている技術力で、3種類の最初のもの。
- （s型）「ソフトウェアサービス企業」（service company）が必要としている技術力で、3種類の2番目と3番目を合わせたもの。

日本のソフトウェア技術力を検討するときには、（p型）と（s型）のどちらについて語っているかを明瞭に意識する必要がある。特に、ソフトウェア産業振興政策を考えるとき、その政策がどちらに重点を置くものか意識することが重要である。ソフトウェア産業が必要とする「生産機械」は、せいぜいコンピュータと通信インフラ程度であり、設備投資にほとんど費用がかからない。ソフトウェア産業は極めて労働集約的である。つまり、ソフトウェア産業最大・最重要の生産装置は「人」、つまり、技術者であり、振興策としては人材育成以外に重要な方策は考えられないのである。しかし、（p型）と（s型）の技術力では、それに従事する技術者のタイプが大きく異なる。たとえれば、それは芸術家と企業人のように違う。実は、この2つの能力は相反することさえあるのである。

当然、人材育成の方策は異なる。Cusumanoは文献[4)]で、好況、不況の波に影響されないためにはハイブリッド型の企業形態がよいと勧めているが、集団である企業は（p型）と（s型）双方の能力を持

ち得ても、一人の個人が、（p 型）と（s 型）の双方の能力を十分に兼ね備えることは容易なことではない。そのような人材を育てるのは実に難しいのである。国の政策を考える場合には、未踏ソフトウェア事業が、大きく（p 型）に傾いているように、どちらかの能力に焦点を当てて人を育てるのか、あるいは、両方の能力を 1 つの個人の中に育まねば、結局は必要な人材は育たないと判断し、そういう人材を育てる教育に投資するか、そういう点まで考慮する必要がある。

図表 1　ソフトウェア輸出入統計調査



※ 2000 年度実績
http://it.jeita.or.jp/statistics/software/2000/4.html　より

2 - 2
日本の技術力の分析

では、日本のソフトウェア技術能力を、この我々の分類を元に検討するとどうなるだろうか。残念ながら、両分野とも極めて弱いというのが結論である。（p 型）においては、Ruby など見るべきものが僅かながら出てきつつあるが、電子情報技術産業協会の統計[10]に見られるように、ソフトウェアの輸出入は、およそ、100 対 1 という超輸入超過である（図表 1）。ただし、ゲーム・ソフトを除いての統計であることに注意せねばならない。

（s 型）の開発法は、方法論の問題であるだけに、このような統計を使っての判断が難しい。Cusumano が一時、日本のソフトウェア産業の形態を、ソフトウェア工場と命名して賞賛したように、この分野では、ある程度の競争力があるという意見もある。しかし、産業としての現状をみれば、（s 型）の 2 つの内のさらに後者、つまり、「それを教えることによりコンサルタントが商売をできるような技術力」においては、まず、ソフトウェア・コンサルタントという業種自体が、日本では確立されておらず、また、大学の研究者も、企業の研究者も、産業に直結する技術は生み出せていない。この分野では、見るべきものはゼロというのがソフトウェア工学者の偽らざる感想であろう。

ソフトウェア部門では、アメリカが圧倒的に強く、アメリカ以外は、すべて弱いともいえる。日本だけが特に弱いわけではない。しかしながら、たとえば、ヨーロッパに比較しても、日本のソフトウェア技術力は劣っている。特に（s 型）の開発法においては、形式的技法①のような基礎理論からユースケース②のような現実的技術まで、ソフトウェア工学の多くの概念がヨーロッパ産であるのに比べ、これに匹敵するものが日本には全くない。日本は IT に弱いといわれるが、ハードウェアに限れば、米国に次ぐ地位を確保していることに留意しなくてはいけない。IT 業界においては、ハード、ソフトを問わず寡占の傾向が強いことを考慮すれば、日本の IT ハード部門は健闘しているといえる。さらに、ゲーム機においては、ハード、ソフトとも、いまだ日本が「覇権」を持っている。これらを考慮すれば、ゲームを除くソフトウェア部門における日本の弱さは極めて特徴的だ。我々は、この事態は単なる歴史的偶然ではなく、何らかの根本的原因に起因するものと考える。

用語説明

①形式的技法
形式言語、形式論理学などを利用するソフトウェア工学の方法の総称。プログラムが仕様に対して正しいことを論理学的・数学的な証明を使って保証するプログラム検証論は、形式的技法の代表的分野である。

②ユースケース
スウェーデンの Ivar Jacobson が創始したシステム要件記述の形式（で記述された 1 つ 1 つのシステム要件の記述例）。

3. 弱さの原因は何か？

では、日本のソフトウェア技術力の弱さの原因は何だろうか。これについては、多くの異なる意見がある。おそらく原因は1つではない。しかし、我々は、その主な原因は、日本社会の「合理的思考法の弱さ」のためではないかと考えている。

3-1 ソフトウェアと合理性・論理性

日本社会は、合理的・論理的思考法に弱いといわれる。これは日本人論などにおいて繰り返し主張され、議論や非難の対象にもなる説である。我々は、この説に部分的ながら賛同し、そして、日本のソフトウェア産業の弱さは、この日本社会の弱点に起因すると考える。

この説とは異なり、語学力、特に英語力の弱さを原因にあげる人もいる。「知」を記述し記録し加工し伝達するための最良のメディアは、言語である。UML[③]のようなグラフィカルな方法も「グラフィカル言語」という。言語とは、「知」を記述し記録し加工し伝達するための手段の総称なのである。当然、語学力の弱さは、非合理性･非論理性の原因となり、我々の説が正しければ、ソフトウェア産業の弱さの原因となりえるのである。この「言語説」も、我々の説と矛盾しないことを注意して、話を元に戻そう。

日本のソフトウェア産業とソフトウェア工学の「弱さ」は、日本人の非合理性・非論理性に起因する。では、なぜ、非合理性・非論理性が、ソフトウェア産業の弱さに結びつくのだろうか？　それはソフトウェアの本性が合理性・論理性そのものだからである。本性が合理性・論理性であるものを、非合理的・非論理的に作成しようとすることが困難なのは当たり前のことである。

では、ソフトウェアはどういう意味で合理的・論理的なのだろうか。それにはソフトウェアの本性を検討する必要がある。我々はソフトウェアの本性は、次の2つであると考える。

- ソフトウェアはサイバー空間を支配する人工的ルールである。
- ソフトウェアは目的をもって構築される。

3-2 ソフトウェアと論理性 ―検証―

ソフトウェアを一言で表せば「サイバー空間を支配する人工的ルール」ということができる。著名なソフトウェア・コンサルタント、A. Cockburn は、ソフトウェアを哲学者ヴィトゲンシュタインの「言語ゲーム」という概念で説明している[3)]。ゲームソフトやシミュレーション・システムを考えれば分かるように、コンピュータの内部では現実の物理法則に反する仮想的な宇宙さえ構築可能である。コンピュータというサイバー空間においては、プログラマは物理法則をさえ創造できる「神」と化すのである。すべては人工であり、あらゆる現実的規則や法則からの自由が保障される。実際にはハードウェアの能力という大きな制約があるものの、理論上はソフトウェアの世界は論理にしか限定されない「自由な空間」なのである（我々は「論理」を広い意味で使う。たとえばアルゴリズムの効率は「論理」の中に含まれている）。

ソフトウェアは抽象数学などと同じく、概念の世界の装置であり、物理則などの「この世界の法則」に拘束されることが少ない。それを拘束するものは、概念が従うべき法則、たとえば論理法則などの少数のものに限られる。これがソフトウェアを機械装置などの他の人工物と分かつ大きな特性であり、この白いカンバスのような世界に、（広い意味での）「論理」という絵の具で自由に絵を描くこと、それがソフトウェア開発ということなのである。

人工的ルールによる真理性（形式合理性、目的合理性）を統治するのは、広い意味での論理である。それは技術的には、数理論理学でいう形式系や項書換系[④]などの機械的推論により体現され、それ故に、形式的検証 formal verification がソフトウェア工学・計算機科学の基礎とされるのである[6)]。

3-3 ソフトウェアと合理性 ―要求工学―

この verification と対になる言葉として、使われる validation と

用語説明

③ UML
ソフトウェアの設計書に擬されるモデリング言語のデファクト・スタンダードになる可能性をもつ半形式言語。創始者の3名は“Three Amigo”と呼ばれ、I. Jacobson はその1人。

④形式系、項書換系
形式系とは三段論法のような論理を機械的規則としてまとめたもの。項書換系は形式系の一部と見なされることもあるが、論理よりは主に数学でいう式変形や計算を表現する。

いう言葉がある。日本語にしてしまうと、これは確認となり、検証とあまり区別がつかないが、この二つは大きく異なる。verificationとは完成された仕様にソフトウェアが合致するか否かを確かめることである。このときソフトウェアに要求される機能を記述する仕様は絶対的な公理と考え、これを変更することは考えない。verificationとは論証幾何学で公理から定理を証明するような行為であり、サイバー空間から一歩も外に出ずに行える（図表2）。

一方、validationの方は、完成したソフトウェアを動作させることにより、それが仕様以前の要求、つまり、ソフトウェアが構築された本来の目的に合致するかを調べることを言う。この場合には、仕様が目的に合うかどうかもチェックされる（図表2）。つまり、verificationの場合と異なり、仕様は公理の位置から降ろされ、物理現象を記述する際の微分方程式のような位置に置かれるのである。ある現象を記述したはずの微分方程式の数値解を求め、それが現実と大きく矛盾していれば、数値解析に問題がなければ、微分方程式の方が間違いなのであり、それが交換される。つまり、仕様とは、目的の定式化であり、その定式化された目的と、それを解決するための「解」としてのプログラムを本来の目的に照らし合わせて、同時に検討すること、それがvalidationであり、verificationとvalidationは、ソフトウェア開発の車の両輪として認識されている。

現実のvalidationでは、目的の定式化である仕様と、プログラムの両者が同時に検討される。しかし、これはプログラムが完成したときにのみ行える。しかし、実際の開発プロジェクトで、最も恐ろしいことは納期直前に見つかる仕様のバグである。その多くはサイバー空間の中だけで解決されるような自己矛盾に関するものではなく、完成したソフトウェアが提供すべき機能と、本来の目的、つまり、要求との乖離によるものである。このため仕様と目的の摺り合わせ、あるいは、ぼんやりした目的を明瞭な目的に洗練し、さらには、それを最大限忠実に表現する仕様を作成することが、現実のソフトウェア開発において急速に重要度を増している。この問題を解決すべく考えられたのが、要求工学（requirement engineering、要求管理工学、要求定義工学などともいう）である（図表2）。この要求工学こそが、ソフトウェアの本性のもう一方、「ソフトウェアは目的をもって作成される」の目的を開発する部分である。

図表2　Verification, validation, requirement engineering

verification	すでに何らかの方法で明瞭に記述されたシステムの必要条件を仕様というが、その仕様に対して、システムが正しいかどうかを調べること。
validation	システムが仕様作成以前の要件に合うか、あるいは仕様が、本来の要件にあるかを調べること。Verificationに比べて言葉の解釈にブレがある。本稿では広い意味にとっている。
requirement engineering	開発すべきソフトウェアが必要とする要件を洗い出すための技術。特にカスタム・ソフトウェアの開発現場では不可欠の技術。

4. 日本のソフトウェア産業は絶望的か？

ソフトウェアの本性の「人工的ルール」の部分は、すでに述べたように論理性そのものである。一方で、もう1つの本性、つまり、目的の定式化としての要求定義、それに必要な要求分析の部分は、いわゆる伝統的論理学は別として、近代的論理学が完全に放逐してしまった部分であり、これは論理ではない。この部分は、社会学者Max Weberの（未完の）「合理性の理論」の用語で説明することがふさわしい。Weberの用語に従えば、第1の特性「人工的ルール」は形式合理性・目的合理性であり、要求工学的に第2の特性を解決することは、価値合理性・実質合理性に基づいて、形式合理性・目的合理性の出発点を作成・検討することにあたる。つまり、ソフトウェアの本性は「合理性と論理性」からなり、それを実現するには、当然のことながら、合理的・論理的思考法が必要となるのである。

藤本隆宏は、日本の自動車産業の強さを分析した文献[5]において、工業製品のアーキテクチャを「モジュラー型v.s.インテグラル型」と「オープン型v.s.クローズド型」の2つのパラメータ軸により4分類し、日本は自動車、ゲームソフトなどの「囲い込んでの摺り合わせ」を必要とするクローズド・インテグラル型アーキテクチャを持つ製品に強く、パソコン、パッケージソフトなどのオープン・モジュラー型の製品には弱いとした。

藤本の説は設計・製造という行為を「情報の転写」としてとらえる「設計情報転写説」という理論に基づいており、たとえば自動車のボディのプレスを鋼板への形状という情報の転写と考える。そして、日本が得意とするのは製品が鋼板のような「書き込みにくい媒体」の場合だとする。この部分の藤本の議論は不明瞭な所が多いが、要するに「書き込みにくい媒体」に優れた転写を行うには「作り込み」が必要であり、そのため

に細部へのこだわりや芸術的な技術が必要な「囲い込んでの摺り合わせ」能力が製品の品質に直結するという主張である。

藤本の「情報の転写」という言葉を使えば、要求工学は、暗黙知という情報を形式知という情報に転写することであり、仕様に基づくプログラムの開発とは「形式化された目的」である要求を実行可能なプログラムの形式に転写することにあたる。また、サイバー空間は、論理以外の制約を受けないという点において、究極の「書き込み易い媒体」である。つまり、合理的・論理的にプログラミング言語などの形式言語で「ソフトウェア設計情報」を記述すれば、実は、すでにそれが製品なのである。自動車の場合のように、さらに、それを鋼板に転写するというような工程は必要ない。つまり、「設計イコール生産」に近いといえる（実は、そうではなく、「設計」が多段階となるので、これはあくまで比喩である）。

藤本は、日本のソフトウェア産業の国際競争力のなさを、「日本の技術は情報転写が容易な分野では弱い」という説で説明している。ソフトウェアという製品は、記憶媒体に書き込むだけで転写できるのである。「ソフトウェアは合理と論理だけの存在である」という我々の分析と、「ソフトウェア分野は書き込み易い媒体を対象とする」という藤本の分析は、ほぼ同じことを主張していると言えるだろう。我々は、組み立て方に注目し、藤本は組み立てたものの書き込み方に注目したのである。

もし、日本人は本来、その文化の根底から非合理的・非論理的であるという説が正しく、かつ、我々のソフトウェアは合理性と論理性の結晶であるという説が正しいとしたら、日本のソフト産業の弱さの理由は文化的なものとなり、日本がその文化を変更しない限り、日本のソフトウェア産業は永久に浮上しないという結論になる。また、我々とは全く独立に、媒体への転写という異なった視点を使い、生産工学の立場からなされた藤本の日本のソフトウェア分野の弱さの原因分析との合致は、この結論をサポートするかのように見える。さらには、ソフトに限らず、これが「失われた10年」の原因ではないかと結論したくなるところである。

5. アジャイル法という逆説

しかし、現実は、そのように単純ではない。藤本は、現在、リーン生産法と呼ばれ、世界に普及している日本発の生産法、つまり、トヨタ生産法がアメリカで「発見」されたのは「失われた10年」の間であったと指摘している[5)]。しかし、この時期に、アメリカが発見した「日本的」なものは、生産工学だけではなかった。

日本的生産法の強みの発見より僅かに遅れて、ソフトウェア工学の最先端国アメリカで流行し始めた、アジャイル法と呼ばれる一群の新開発法があった。それらは、もともとは生産工学における日本的手法の発見とは関係なく、カスタム・ソフトウェアの開発現場における現実に対処するために生まれてきた。

従来のソフトウェア工学の常識では、ソフトウェアを開発するには、まず、綿密な計画を練り、それ以後は、計画を変えることは悪とされた。また、仕事の内容は、モジュール化し、分散し、それぞれのモジュールのインターフェースは、詳細な「契約」により記述されるべきだとされていた。この前提は、あらゆるソフトウェア工学の基礎であり、これを逸脱するものは「悪」とされていた。そして、ソフトウェア工学とは、悪が横行する現場を如何に教化するかということでもあった。

しかし、アジャイル法は、その根底的原理を、現場での生産性の高さと品質の高さという現実により見事にヒックリ返してみせたのである。これは重量的計画的生産方式で凝り固まっていたデトロイトにトヨタ生産方式が与えたショックに似ている。しかし、ソフトウェア工学では、ショックは外からでなくアメリカの内部から起きた。

これらの技術は、従来のソフトウェア工学の常識を完全に覆すものであるため、その内の、もっとも有名なものは、eXtreme Programming「過激プログラミング」とさえ自称する。現在、XPという名前で知られ、この1、2年で、アメリカのみならず、日本でも大流行し始めている技術である。アジャイル法に分類される方法論には、XPの他に、Scrum、Crystal、Adaptive software development methodなどがあり、最近のものとしては、Lean Software Developmentが知られている。現在、世界のソフトウェア産業は、従来の技術の思想を根底から覆すかのような、この技術思想といかに付き合うべきか苦闘している。

これらの方法は、10数名程度までのプロジェクトならば、凄まじく迅速で柔軟であると言われる。この方法の創始者たちは、その技術を、Iacocca研究所が提案した生産工学の手法を真似て、アジャイル（agile）と呼んだ。その名前は、医療産業、自動車産業などにおける、エンドユーザーの要望とその変化に柔軟で迅速に対応するため

の方法論からきているのである。

そして、このアジャイルが、日本発の生産方式、経営学と堅く結びついていることが、この数年、アメリカで主張され始めたのである。最も典型的なのは、Mary Poppendieck が主唱する、Lean Software Development であろう。この方法論のモノグラフ[8]には、Ohno（大野耐一）、software kanban などのトヨタ生産法に関連する人名や用語が使われており、導入部からしてトヨタの話から始まる。また、Scrum という方法論は日本発の経営学で著名な野中郁次郎の著作による名前である。さらに、アジャイルのスターとでもいうべき、Kent Beck は、XP の国際会議 XP2003 のパネルで、XP の中心的技術である Test Driven Development におけるテストケースをリーン生産法の「無駄」の概念を使って論じている[1]。

一時、マスコミで喧伝された「IT 産業＝次の時代の新産業」、「機械工業＝滅び行く旧産業」、そして、「新産業という時代の波に乗った国＝アメリカ」、「旧産業にしがみつき時代の波に乗り遅れた国＝日本」という単純化された、「勝ち組 v.s. 負け組」の理論からすると、負けたはずの日本の旧来型の自動車産業などに起因する知恵が、勝ち組アメリカの、その勝利の象徴であるはずの IT 産業の中心的部分において、最新の知恵として称揚されていることになる。まさに逆説的である。

このアジャイル法が、どれだけ「日本的」であるか、少しではあるが、具体的に説明しておこう。XP では、完結した仕様記述は悪であるとされる。仕様と、その実装という 2 重構造のメンテナンスが開発の効率を阻害すると考えるからである。詳細な要求を書き、綿密な Plan を立ててから始める開発法を、up-front（先払い）開発法という。舳先、つまり、開発初期にコストの多くを投入すべきだという、この考え方では、時間軸を横軸としてコストのグラフを書くと、舳先、つまり、開発初期のグラフがピンと上を向く。これを称して up-front というのである。

アジャイル法では「システムは非線形であり創発的あり、また、顧客の気持ちや環境は変わるもの」と割り切って勇気をもって変化と対峙するという方法をとる。これは、藤本がトヨタの思考法の持つ合理性として指摘した「事後合理性」そのものである（up-front は、藤本の「事前合理性」に対応する）。

XP はツールに重きを置かない。実は、XP には仕様記述が巧妙に組み込まれており、しかし、それが極力「軽く」あるように仕組まれている。そのために、特殊なツールに依存することは避け、ツールはコンパイラだけにとどめるように示唆される。また、仕様記述にあたる CRC というテクニックが使われるが、これは、単純なフォーマットを印刷した紙のカードである。これはデトロイトの重いコンピュータによる生産在庫管理と、カンバン（これも紙）によるトヨタの方法の対比を思い起こさせる。

アジャイルでは、個人プレーよりチームプレーが重視される。XP では、プログラミングさえ、必ず 2 人 1 組でやることになっている。仕事場が個室でなく大部屋であることは当然である。その方が他のメンバーの議論が耳に入るのでよい。メンバー全員がプロジェクトの全体を理解すべきなのは当然である。

アジャイルでは決定は先延ばしにするべきだとされる。もちろん、問題を先送りせよというのではない。現時点では決定が困難・不可能な事項を無闇に決定してはいけないという考え方である。

アジャイルでは顧客との連携を重要視する。“The Customer is God”、「お客様は神様です」というフレーズは、アジャイルではちょくちょく耳にするフレーズである。あげくの果ては顧客をプロジェクトのメンバーとして常駐させ、変更や先延ばしにしていた決定が必要となったら、その場で顧客に決定と指示を求めるというオンサイト・カスタマーという技法さえ使われている。

アジャイル法と「日本的」思考法の類似性は、これだけではなく、

図表 3　アジャイル・マニフェスト

http://agilemanifesto.org/ より

その同型性は気持ちが悪いほどである。そして、この「日本的方法」との同型性が、アジャイル法の創始者達に意識され始めたのは、アジャイル法の評価が定着した後であるということは、これが「日本趣味」の結果でないことの何よりの証拠である。

6. 象性と猿性、そして、その融合

このアジャイル法は、優秀なプログラマ、特に、「UNIX文化」という名で知られる特定の思考法を行うプログラマたちには、現実的方法として理解されやすい反面、従来のup-frontの思想で教育を受けた技術者には、おおきなとまどいになっている。スパイラル・モデルの提唱で著名なB. Boehmは、この状況を解決すべく、2004年の初頭、up-frontの持つdisciplineとagilityの関係を明らかにすることを目的として“Balancing Agility and Discipline”[2)]を発表した。この著書は、象と猿の逸話で始まっており、これが見事にagilityと合理性の関係を示している。その逸話を要約してみよう。

ジャングルの辺の村に1匹の象が住んでいた。象はジャングルから村人の食料としてバナナを採っては運び村人に感謝されていたが、ある日、1匹の猿がやってきて見たこともない珍しい食物を村に運び始めた。バナナに飽きていた村人は、猿の仕事を大層喜び、象はいつしか忘れられてしまった。しかし、村の人口が増えるに従い、小さな猿で村人の需要をまかなうことはできなくなった。村人の批難を浴びて困った猿は、忘れ去られて寂しく暮らしていた象を訪ね、身軽な自分が珍しい果物と見つけるので、象の力で、それを大量に運んでくれないかと頼んだ。こうして珍しい果物を大量に提供できるようになった象と猿は幸せに一緒に暮らした。

Fordism⑤、Taylorism⑥、up-front開発のような、事前の入念な計画を主とするシステム合理性に基づく方法が象である。一方で、アジャイル開発、トヨタ生産方式、などは身軽で迅速な猿なのである。Boehmは、象と猿の逸話のように、agilityも、up-front的disciplineも、ともに重要であると結論したのである。これはアジャイル法の現実的成功への妥協、非合理性への合理性の妥協なのだろうか。実は、そうではない。

合理性に多くの型（タイプ）があることは、社会学の前提と言ってもよい。その合理性の多様性を指摘したWeberの著作の中に、up-front的なSystematiker（システム構築者）の合理性と、不断の近似改善により漸進的に現実に適合しようとする合理性が対比されている箇所があることを、社会学者の矢野善郎[11)]が指摘している。前者は、藤本が事前合理的と呼ぶ、Fordism、Taylorismの合理性であり、後者は、藤本が事後合理的と呼ぶ、アジャイルやトヨタ生産法の合理性なのである。つまり、象と猿の逸話は、合理と非合理の妥協ではなく、象の持つ合理性「象性」と、猿の持つ合理性「猿性」の融合の物語なのである。

生産工学の専門家によると、Lean⑦、Agile⑧、TOC⑨などの新しい生産方式は、実はFordism、Taylorismが基本単位のミクロ部分を担い、いろいろな目的に合わせて、それを様々に組み合わせたものであるという。ソフトウェア工学でも同じことであり、アジャイル方法論を子細に分析すると、実はup-frontの基礎理論と同じ仕組みがしっかりと組み込まれていることがわかる。たとえば、XPの主要部分であるTDDにはup-front開発の基礎とされるホーア論理学に基づくプログラム開発法が巧妙に組み込まれているといえる[7)]。

現代の社会は非常に複雑であり、また、猛スピードで変化する。これに合理的に対処するにはup-frontだけでは不可能であり、社会学や哲学でいうリフレクショ

用語説明

⑤ Fordism
Henry Fordによる大量生産方式。フォード生産方式。

⑥ Taylorism
科学的管理法ともいう。生産方式、作業方法などを科学的に分析し、生産性を高める方法で、日本のTQC運動の祖先。米国の技術者Taylorが提唱した。科学におけるrationalismが終わったころに始まった技術におけるrationalismの魁という、思想史的意義も大きい。

⑦ Lean
リーン生産法。トヨタ生産法を米国MITで体系化したもの。無駄の除去に特徴がある。

⑧ Agile
Leanと関連があるが、こちらはもともと米国産の生産法。無駄の除去よりは柔軟性と迅速性、つまり、変化への対応に力点がある。

⑨ TOC（Theory Of Constraint）
E. Goldrattが提唱した生産管理方式でLeanと類似しているが、システム全体のパーフォーマンスに力点を置く。

ン、藤本の言う事後合理性が必要なのである。あまりに複雑な「やってみないとわからない」という事が増えている。しかも、解を与えた途端に、その解の故に最初の要求が変わるというのは、ソフトウェア開発者が「ソフトウェア開発の難しさを理解しないユーザー(発注者)」に対して常に持つ不満である。しかし、これを「ユーザーが悪い」とすることはできない。そういう無理難題とも思える要求にさえ対処する必要があり、それに対処できないものは、競争に敗れるのである。

実は現実的 up-front 開発の代表である、UML ベースのモデリングの研究を通して、分かりつつあることは、要求仕様の獲得としてのモデリングにおいて、最も効率的に要求を集める手法の1つはアジャイル法を使うことなのである。これは象性と猿性の融合そのものである[7)]。

以上の解説が、日本のソフトウェア産業を考えるときにもたらす教訓は明らかだろう。日本企業が発揮した猿性は合理性なのである。それは Taylorism などに代表される象的合理性とは別種の合理性なのだ。そして、Taylorism、Fordism が世界を席巻したころと異なり、現代は象的合理性と猿的合理性の両方が同時に求められているのである。そのために象性の国アメリカは、日本の猿性に学んだのである。日本は確かに事前合理的な Systematiker の合理性、つまり、象性に欠けるのかもしれない。しかし、現代社会が2つの合理性を要求し、しかも、そのひとつは日本発の普遍的な合理性であるとしたら、我々は、すでに半分を持っている。アメリカが猿性という「半分」を学んだように、我々は象性という「半分」を学べばよいのである。

7. 結論

日本のソフトウェア産業の規模は大きい。銀行のオンラインシステムなどのカスタム・ソフトウェアの売り上げは巨大だ。その市場は「言語障壁」、「文化障壁」にまもられたドメスティックなものである。しかし、最近話題になっている新生銀行のシステムの例にみるように、インド人技術者などの進出はめざましく、さらに中国人技術者が大量に日本に入るようになれば、このドメスティックな産業が海外資本に席巻される可能性は否定できない。

また、もし、そうならなければ、それは常に日本の情報システム、特に、これからの社会の競争力を決定する重要な要因となる情報システムの性能において、日本が米国、ヨーロッパ、アジア諸国の後塵を拝することになり、結果として日本社会の競争力を著しく低下させることになる可能性もある。そして、その兆候は各所にある。

この状況の原因は単純ではなく、その根は、現代日本社会が持つ特有の思考法、特に、合理性・論理性への無知と誤解からくるものである可能性が高い。その原因は明治以来の社会システム、特に教育システムに起因する可能性が高い。我々は、この観点から、研究を進めているが、本論説では、その視点からソフトウェア産業の内、特に、技術的な側面であるソフトウェア工学についての分析を行った。

7-1 象性の補填と猿性の促進

我々の結論は、「ソフトウェア開発に必要とされる2つの合理性の内、象の合理性を補填する必要がある」というものである。その際、すでに日本社会がもっている猿性を顕在化し、それを保持しつつ、改善し、象性と融合する必要がある。

従来のソフトウェア工学では象性のみが重視され、猿性が軽視される傾向があった。しかし、真の解は、それらの融合であることが、ソフトウェア工学の最前線で明らかにされつつある。また、この思想は、ソフトウェア工学を超えて、「生産」、「設計」に関連する実に多くの分野で有用であることも分かってきている。ソフトウェア工学におけるアジャイルが、生産工学や経営学という、常識的には設計や生産の形態が全く異なる分野のアイデアから大きな影響を受けていることは、その1つの証拠であろう。

外来の手法を「公理」のように拝聴するか、それらを「非現実的」と断じて、一方的に無視する両極端の立場が横行することが多い我が国の状況からすると、「融合」が解である場合、この両極端しかとらない思考法が、その競争力の根源的弱点になりかねない。たとえば、日本のソフトウェア技術者には、猿性を非合理的とみなして、一方的に攻撃し、自らの社会の利点をつぶす、あるいは、卑下する傾向がなかったであろうか？

猿性、そして、猿性と象性の融合こそが、ソフトウェア工学の要であるという思想が認識し始められた今こそ、日本のソフトウェア工学を世界水準に高める絶好の機会なのである。トヨタの秘密はリーン生産法の理論形成の後も、いまだに完全には解明されてない。たとえリーン生産法を採用しても、トヨタと同じ生産効率に達している自動車会社は、世界のどこにもない。そし

て、トヨタ自身でさえ、その方法の全貌はつかみきれていないという[5]。我々は、そのトヨタと同じ文化に属す。光は足下にある。これを生かさないことこそ最大の非合理だ。それによって世界に追いつくだけでなく、追い越せる可能性さえ否定できないのであるから。

7－2 政策割り出しのための研究

象性と猿性の融合が提唱されている現在は、日本のソフトウェア産業が、世界のトップに躍り出る絶好の機会である。この機会を生かすには、まず、少なくとも次の3つのタスクを実行し、必要な政策を割り出し実行する必要がある。

① Boehmの象性、猿性の観点から、ソフトウェア工学とトヨタ生産法などの生産工学の手法の徹底的研究を行い、それに基づいて日本のソフトウェア産業と、自動車産業などの比較研究を行い、ソフトウェア産業の構造的問題の解明を行う。
②日本が強いといわれるゲームやモーバイルにおける日本の技術力が、我々の理論の反例になっていないかを検討する。
③本来、全く異なった生産の形態の例とされていた自動車生産とソフトウェア生産の背景に潜む同型性を解明し、これらを他工学分野や経営学などにも及ぼす。つまり、これらの工学における「生産」、「設計」という問題の上部構造を解明し、その理論を構築する。

この3つについて、少し詳しく説明しよう。ソフトウェア産業への即効性を考えれば、最初の2つが重要である。どちらも、自動車とゲーム、モーバイルという、我が国が強い分野の研究であり、特に自動車の場合、多くの研究が存在するので、一挙に研究が進む可能性が高い。Cusumanoや藤本の分析も、この分野は避けて通っているといえる。生産工学との比較研究は、従来なかった観点であり、また、比較対象となるカスタム・ソフトウェアの生産現場は、比較的、政府の施策からはずれていた分野でなかったかと推測する。その意味で、新たな、施策が効果を上げる可能性は少なくない。この研究には、藤本の情報転写説に基づく研究が重要な導きの糸となるだろう。

ゲームと、モーバイルの2分野に日本が強い競争力を持つことは、我々の理論への強力な反論の論拠たりえる。この事実に妥当な説明を与えられない間は、我々の理論は、いまだ砂上の楼閣にすぎない。しかし、我々が主な議論の対象としたカスタム・ソフトウェア分野と、これら2つのIT分野が大きく異なることは、ユーザーがシステムとやりとりする論理的情報量が、この2分野では、通常のPC等と比べて極端に少ないことから推測することができる。この2分野の競争力の強さを説明できるようになったとき、我々の結論は、極めて強固な基礎を得るだけでなく、我々の理論に新たな展開があるものと推測している。

携帯電話の場合、初期には日本製製品のインターフェースは、明らかに場当たり的であり、ソフトウェア設計の原則をふまえたNokiaなどの海外製品に比べて劣っていたが、これが急速に改善されている。また、一方で、ゲーム・ソフトウェアの生産はアメリカにシフトしつつあるとも聞く。これらの変化の原因を追及することが、この研究の糸口となるだろう。

最後の「上部構造の解明」であるが、将来へのインパクトを考えれば、これが最重要である。スウェーデン・Telelogic社では、要求開発プロセスを、カスタマー、サプライヤーの重層構造という、サプライ・チェーン的な発想で捉えている[12]。また、逆に、藤本[5]は生産を設計情報の転写と考える。これらは、本来、全く関係ないと思われていた分野が深く関連しており、理論的にも、それを解明できる可能性を示唆している。設計と生産という2分法を廃棄する必要がある可能性も捨てきれない。

7－3 考えられる政策の方向

この「猿性と象性の融合プロジェクト」の最終段階は、日本のソフトウェア教育・情報教育、というより、あらゆる教育の方法の根本からの改革に結びつく可能性が高い。その改革は、学校の改革ではなく、社会の改革が、学校に及ぶ形である必要がある。

日本のソフトウェア能力の根源的問題点は、日本社会が持つ「黒か白かの極端思考法による脆弱な思考力」による可能性が高い。また、情報システムをすぐにコンピュータやソフトウェアを発注したり購入したりすることとしか理解できないという「情報」という概念への無理解も大きな問題である。

しかし、このような思考方法の抜本的改革を、学校システムの政府主導の変革のみに求めるやり方は、失敗する可能性が高い。学校システム、教育システムの変革は、社会の側から湧き起こるべきであり、この形態であってこそ、このような改革は可能だろう。そのためには、政策割り出しの研究の終了をまっての、社会における象性能力の開発と普及などというup-front的な計画はとるべきでなく、研究途上において、生産現場、そして、教育現場において、研究によって得られた成果を次々と適用し、その問題点をさぐるという方法が必要である。つまり、情報

技術者をカスタマー、情報技術者を生産する教育機関をサプライヤーとみなし、これ自体に、トヨタ生産方式のような生産工学的発想を適用するのである。教育機関と、その「生産物」たる人材を受け入れる企業・社会をサプライ・チェーンの発想で見直すのである。

社会は大学等に積極的に必要な人材像を提供し、また、大学は、その人材像を大学に提供できるような人材を育て社会に提供する必要がある。どちらかが先ではなく、両者が同時に改善されなくてはならない。そして、上記の①～③の研究は、その中で同時に行われる必要がある。教育システムの抜本的改革は、これらのタスクが行われる中で自然発生的に起こるべきことである。

この改革は国が主導して行うべきことではなく、市民の側から湧き起こるものでなくてはならない。しかし、それを国が助けることは可能であろうし、また、その種を撒くことは可能である。その一つの方法としては、現在決定的に不足している「良いカスタマー」を育てる教育がある。これはCIOを育てる教育と言ってもよい。しかし、おそらく産業界でも、「良いカスタマー」やCIOは、どのような人物であるかという明確なイメージはないだろう。まして、その教育体系など現在の日本の大学にはない。この両者を、双方からの要求に基づいて、同時に育成する。おそらく、これが現在なすべき最も重要な政策であろうと考える。

参考文献

1) A panel at XP 2003, Test Driven Development（TDD), Steven Fraser, Kent Beck, Bill Caputo, Tim Mackinnon, James Newkirk, Charlie Poole http://www.xp2003.org/panels/fraser.html
2) Barry Boehm and Richard Turner: Balancing Agility and Discipline: A Guide for the Perplexed, 2004、翻訳：「アジャイルと規律～ソフトウェア開発を成功させる2つの鍵のバランス～」、ウルシステムズ株式会社、2004
3) Alistair Cockburn：Agile Software Development. 翻訳:「アジャイルソフトウェア開発」、ピアソン・エデュケーション、2002
4) Michael Cusumano: Strategy for Software Companies: What to Think About, An invited talk at XP 2003：http://www.xp2003.org/keyspeeches/cusumano.html., cf. Michael Cusumano: The Business of Software : What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad, Free Press, 2004
5) 藤本隆宏：能力構築競争―日本の自動車産業はなぜ強いか―、中公新書1700
6) 林晋：プログラム検証論、共立出版、1995
7) Susumu Hayashi, Pan YiBing, Masami Sato, et al.: Test Driven Development of UML Models with SMART Modeling System, to appear in the proceedings of UML 2004.
8) Mary Poppendieck, Tom Poppendieck：Lean Software Development：An Agile Toolkit for Software Development Managers
9) James P. Womack, et al: The Machine That Changed the World : The Story of Lean Production
10) 電子情報技術産業協会：ソフトウェア輸出入統計調査2000年実績、平成14年7月31日：http://it.jeita.or.jp/statistics/software/2000/
11) 矢野 善郎：マックス・ヴェーバーの方法論的合理主義、創文社、2004
12) Elizabeth Hull, Ken Jackson, Jeremy Dick, Requirements Engineering, Springer-Verlag, 2002.