あるソフトウェア工学者の失敗

日本の IT は何故弱いか

林晋
京都大学文学研究科

はじめに

本書において、私に課せられたテーマは、日本の情報産業、特にソフトウェア産業が何故弱いか、そのことを分析・説明することである。その本稿が、思い出話風であることに読者は戸惑うかもしれない。本稿を「思い出話」にする理由は、私が日本のソフトウェア産業、そして、ソフトウェア産業に限らず、日本の多くの企業・産業が陥ってしまった落とし穴の性格が、いわゆる「技術的」「産業的」なものではなく、社会的、文化的なものである故に、工学、技術、経済学、などの理論的・学問的な言葉では語れないと思うからだ。

自転車の乗り方を物理理論で語っても、自分が会得した自転車の乗り方を伝えることはできない。私が、本稿を、思い出話として書くのは、これと同じ理由である。本稿で、私が伝えたいことを、私は私自身の失敗の連続の中で理解した。無意識の内に自分に嵌めていた何重もの「タガ（箍）」に、失敗するたびに気づき、それを一つ一つ外していくことにより、私は、それを理解した。私がこの経験から得たものは、理論的言葉によっては伝えることができないだろう。そう思って、思い出話として、この稿を書いている。

あるソフトウェア工学者の失敗の半生

現在の私は人文学者だが、もともとは数学者だった。学位研究の分野は、構成的数学というもので、チューリング計算可能関数と関係が深い。しかし、学位を取得したものの職がなく、ソフトウェア工学の形式的技法、特に形式的検証というものに分野を変えた。

その後、さらに UML, アジャイルなどのソフトウェア工学、情報人材育成のための政策研究、などと情報関係の実際的分野での仕事を経て、最後に現在の様な、文系の情報学、社会学と歴史学、さらには、それらと哲学の間のような研究分野に移ったのである。

一見、好き勝手に分野を変えているように見えるかもしれない。しかし、この私の研究分野の変転の理由こそ本稿で伝えたいことなのである。振り返れば、それは IT の社会の中における役割の変化を反映するものだった。この様な分析は、後回しにして、まずは、私がどの様に失敗し、その結果、どの様にタガを外し続けたかを語ろう。

数学から形式的検証へ

私は、数学、詳しく言うと数理論理学で学位を取得したが、職がなく、危うく「オーバードクター」第一号になるところだった。幸い博士論文の主査の御好意で職を得たものの、それは長く座り続けるべき職ではなかったので、職を求めて IT に転向した。

私の学位研究のテーマだった構成的数学を使って「検証された関数型プログラムを生成」する技術が提唱されていたので、その仕事をすることにした。これは後に、創始者の佐藤雅彦氏により「構成的プログラミング」と名付けられたものだが、要するに形式的検証(formal verification) の一種である。

つまり、私のソフトウェア工学者としての第一歩は数学紛いのものだった。当然というべきか、そのころの私はソフトウェアというものを、数学的に理解し、「ソフトウェアとは、数学の関数の一種である」という立場に立っていた。

これは、一旦は、忘れられたものの、MapReduce, Hadoop などの普及によって「復活」した関数型言語の思想である。ただ、この当時の私が信じていた思想は、現在の実用的なものとは異なり、「学的純粋性」が強かった。私は当時の関数型プログラミングの信念「副作用の存在故に、ソフトウェアの開発において、ソフトウェアの動向を論理的に推論できず、そのためにバグが入る」を本当に信じていた。

この信念のもと、私は構成的プログラミングを現実のものにするプロジェクトを開始した。この当時の構成的プログラミングは、理論的可能性を実験でちょっと示してみた、という段階にとどまっていたので、「将来、それがソフトウェアの標準となる」と関数型プログラミングの推進者たちが主張していた「副作用がない純粋な関数型プログラミング」の範囲で実用的なプログラムを生産できるツールを作ることにした。

仕様のバグ

このツールは、形式的証明からプログラムを「抽出」する故に、Program eXtractor、PX と名付けた。理論的には、数学の証明の形式言語版である形式的証明からプログラムを作ると、その正しさが数学の証明の正しさにより保証され、抽出されたプロラムにはバグが存在しえない。バグの無いプログラムの生産法を提供する、それが PX プロジェクトの目的だった。数年を経て、PX は完成し、最初の大きな抽出実験を行った。

1. まず、証明記述用の形式言語も開発しつつ、1,2 週間程度でターゲットの形式的証明を書いた。
2. それから PX に完全に自動的に関数型プログラムを生成させた。
3. 抽出されたバグがないはずのプログラムの正しさを「一応」確認するため、簡単なデータを抽出されたプログラムに入力して走らせた。

その正しさに「やはり、形式的検証は正しい」と納得する…。その筈だったのだが、抽出されたプログラムは、間違えた結果を返してきた。PX が自動抽出したプログラムは、最初の例から躓いたのである。訳が分からなかった。最初に考えたのは抽出プログラムか形式的証明の文法チェックプログラムのバグだった。しかし、幾ら見直してもバグが見つからない。悶々とした何日かを過ごした後、気が付いたのが、プログラムの仕様 specification の記述にバグがある可能性だった。仕様とは、ソフトウェアが満たすべき条件を文書にしたもので、この場合は形式言語で書かれていた。仕様を検討してみると多数のバグが見つかった。

ソフトウェア工学には V&V, Verification & Validation という言葉がある。Verification は「ソフトウェアが示された仕様に対して正しいこと」を、validation は、「仕様、そして、それを実現したソフトが、仕様の背景にある、そのツールに期待される機能（要求）に対して正しいこと」を、それぞれチェックすることであり、日本語ではどちらも「検証」という。

仕様は数学の方程式の様なものである。数学の応用問題の場合、「(問題の状況を表現する）方程式を正しく立てることが出来たか」と、「方程式を正しく解けたか」という二つが問われる。細かいことを言うと、少し違うのだが、ここでは、前者が validation であり、後者が verification だと思えば良い。

現実の問題やシステムを方程式や数式で記述することは、工学など様々な分野で行われている。そういう分野では、例えばロボット技術者がアームの動きを数式で記述しロボットを制御できるように、数学は現実を記述できる。ロボットの動きを表す数式が間違っていて、validation の意味で正しくないことは日常的に起きるが、それを技術者が現実的に許容できる位の労力と時間で修正できるのである。

形式的検証の推進者は、同じことが形式的検証でも起きると信じていた。実際、私の場合は、形式的検証という「未来の技術」を、この数学の方程式、特に微分方程式とのアナロジーを使って説明していた。しかし、私が PX の最初の実用的例で経験したのは、この見解が楽観的すぎるという事実だったのである。

形式的検証にも、モデル・チェッキングという成功例があり、CPU のユニットの様に基本的構造は簡単だが、可能な状態の数が多すぎて人間には把握しがたいハードウェアやソフトウェアの検証に企業などで使われている。しかし、私が目指していたのは、モデル・チェッキングの方法では、方程式にあたるモデルを記述できない様な種類のソフトウェアの検証だった。ブラウザや OS も、この種類に属する。こういう現在、最も多く使われるソフトウェアの場合には、私の楽観的見通しは正しくなかったのである。そして、その見通しの甘さが、社会と IT の本質的関係を見落としたことから来たことを、私は以後のキャリアの中で段々と理解していくことになった。

私だけではなかった

今から見れば、私の失敗は歴史的必然だった。実は、私の経験と同じ様なことが同時並行的に起きていたのである。私が、形式的仕様のバグが頻発し容易に修正できないことに驚いたのは、1986 年頃だったと思うが、同じ頃に、英国における軍事用 MPU Viper の verification プロジェクトで、ほぼ同じ結果が得られていた。また、私の実験の直後に１年ほど滞在したスコットランドのエジンバラ大学コンピュータ・サイエンス学科でも、私は同じようなものを見た。この研究機関は形式的検証の世界的拠点で、数学の理論を形式的証明のライブラリとして構築するというプロジェクトが進められていた。

すでに何度も出て来た形式的証明というのは、ソフトウェアのソースやＸＭＬのデータと似たもので、プログラムやＸＭＬデータの文法的正しさがコンパイラやバリデータで検

証できるように、証明検証系というツールを使って形式的証明としての正しさをチェックできた。これを使って、丁度、ソフトウェア開発のように数学の形式的証明のライブラリを開発していくのである。それは実に根気のいる仕事である。

ある日、みんなが廊下に出てガヤガヤやっている。何事かと聞いてみたらクレアという院生が数カ月かけて作った位相幾何学のライブラリにバグが見つかり、それまでの努力が無駄になったという。私の実験と同じで、問題は、証明検査系のバグか、数学上の概念をプログラム言語に似た形式言語に写し取るときに起きたバグのどちらかにある。そして、このケースでも、バグは後者だった。

私の形式的証明の製作は実質 1 週間もかかっていなかったし、すぐに修正できた。しかし、クレアの場合は数か月である上に、修正も容易ではない。数学としては高度とも言えない基礎概念の形式言語による表現の誤りが数か月も発見されなかったのである。私は、今でも、廊下に呆然と立つクレアの当惑の微笑を思い出す。

指摘されていた問題

実は、仕様のバグの問題は、形式的検証研究の反対派が、その遥か前から指摘していた。形式的検証の研究者も、それは知っていたが、反対派も我々を納得させるだけの現実的な規模の実例を持っていなかった。だから、我々は、「やれば出来る」「大した問題ではない」などと信じて前に進んだ。しかし、皮肉なことに、推進派である我々が、つまり、私、Viper のグループ、院生のクレアが、反対派が提供できなかった「学問上の理屈ではなく、問題を実感できる規模と内容をもつ『仕様のバグの問題の重さを示す実例』」を作ってしまったのである。

これは重要な発見であり、それを契機に形式的検証のグループを飛び出し、反対勢力に飛び移ることもできた。しかし、私はそうはしなかった。仕様のバグの問題を経験した当時、私は、まだ京大の助手であったが、欧米で PX システムが知られるようになり、それなりの評価を得るようになった。その後、龍谷大学理工学部の助教授に転出したころからは、頻繁に欧米の国際会議やワークショップの招待講演者や委員を務めるようになった。そういう評価を捨てるなど考えもしなかった。というより、このころの私は、まだ、これが大きな問題だと気が付かなかったのである。私は、日々の仕事に没頭し、それなりの評価に満足し、この問題を真剣に考えようとしなかった。

自己欺瞞

学問的な事実が実験・調査などにより示されても、それを完全に正しいと言い切ることは難しい。ある主張の不可能性を示すことは特に難しい。すべての研究者が暗黙の了解としているために見落とされている条件があり、それが見つかって、出来るわけがないと思われたことが、簡単に実現されてしまうことは良くある。

これを「悪用」すると、「誤差です」「ちょっと間違えていましたが修正可能です」「データが十分でないだけです」とか言って、間違えた結論を擁護し続けることができる。だから、最後は、社会がそれを認めるか認めないか、そういう社会的レベルでの決着になるこ

とさえある。これはＳＴＡＰ細胞を巡って、我々が目撃しつつあることだ。

この様な言い訳の連鎖は、自分自身を「欺く」ために格好の方法である。自身の、そして、クレアの例を見ながら、私が選択したのは、その自己欺瞞の道だった。つまり、私は、新しい世界に飛び出すことはせず、「それまでの自分のやり方を何とか生かす」という道を選んだのである。

最初のタガを外す

私の自己欺瞞のタガを外させたのは、形式的検証の教科書、「プログラム検証論」、1995、共立出版、の執筆だった。研究大学と程遠い大学に勤務していた私は、教科書を書く際、日々接する学生たちに何を提供できるのかを考えざるを得なかった。そこで、この教科書を少しでも IT 産業の現場に直結するものにしようと、形式的検証の現実への適用例を収集し、それらの分析と解説を行ったのである。

それが眼を開かせた。良い応用例はあるが、労力がかかりすぎて適用して意味がある範囲があまりに狭い。また、仕様や要求の不安定さは大きな問題だった。カナダの軽水炉の緊急炉停止装置制御プログラムで、冷却水の水位の定義がプログラムのあちこちでまちまちに解釈されていたことが形式的検証を適用する努力の中で発見されたという事例があったが、深く考えてみれば同じことが形式的仕様においても起きる可能性がある。そして、この本を脱稿した後のことだったと思うが、Viper プロジェクトの仕様のバグの論文も知った。

それまで「たいしたことない」と思っていた仕様のバグの問題が、現実的に大変大きな問題であることに漸く気が付き、自分の PX での実験の意味も初めて理解できた。私はクレアの様に呆然と立ち尽くすしかなかった。

この教科書の苦しい執筆期間が終わるころ、私は、構成的プログラミングを捨てる決意をしていた。しかし、構成的プログラミングの分野で研究を続けている若い人たち、特に私と同じ大学の二人の若い助手をどうするかという問題がのしかかった。自分独りなら「間違いでした」と言って恥をかくだけで済むが、それは、この二人の仕事の評価にも影響する。そこで何とか構成的プログラミングに意味をつけようと、「証明アニメーション」というものを考えた。

教科書執筆時の調査で、仕様をインタラクティブに実行して validation を行う「仕様アニメーション」という技術を開発しているグループを知った。仕様のバグとは、仕様が valid でないということなので、これは私の PX での実験に深く関係していた。

構成的プログラミングは、「正しいと形式的証明から抽出されたプログラムにはバグが無い」という原理だが、対偶をとれば「抽出されたプログラムにバグがあれば、もとの形式的証明にバグがある」こととなる。これは形式的証明を実例でデバグすることである。

実は PX を使った実験で仕様のバグに気づいたとき、この方法を使ってバグを発見・修正し、その後も、これを使ったお蔭でライバルたちより高速に形式的証明を構築できていた。仕様のアニメーションとの類似性から、これを証明アニメーションと名付けたのである。

この方法は、最終的には形式的検証を行うので形式的検証なのだが、「デバグ自体が悪だ」という、私の周辺の理論的な形式的検証推進者の精神に反していたので、周りには隠していた。転向時には、その「形式的証明のデバグ」が主役になったわけである。

数学の証明検証系は、数学教育、数学研究への応用も期待されていたので、そちらでは役立つのではないかと考えて、そちらに向けて方向転換したいと二人の助手に提案した。

残念ながら二人は検証技術としての構成的プログラミングを捨てることに反発した。彼らのことを思って提案したが、それを蹴られたのならば、後は自分だけでやって良いはずだ。そう考えた私は、以後、彼らを自分の部下のように思うことを止め（制度上は部下ではなかった)、自分独りで自由に進むことにした。丁度、その頃、神戸大学の工学部に転職する機会が訪れた。研究以外でも能力を超えて働き体調を崩し限界を感じていた私は、独りで転職することに決めた。

UML とアジャイルへ

構成的プログラミングは捨てたが、仕様のアニメーションや、それを元に自分で考え出した証明アニメーションの研究は続けていた。しかし、神戸大の学生たちには、これさえも不評であった。彼らや彼女らは、すぐに役立つ技術を求めていた。

そんなとき、ある学生が「半形式的（semi formal)」と呼ばれる、UML (Unified Modeling Language)のことを教えてくれた。調べてみると変化しつつあった、私のソフトウェア観にぴったり合う。UML のモデルというのは、ある意味で視覚化された半ば実行可能な仕様なのである。UML モデルでは、視覚化などで技術者とソフトウェアの関係への配慮がなされていた。つまり、モデルは数学的存在ではなくて、カー・デザイナーが作る自動車のクレイモデルのような位置に置かれていた。ある意味で、仕様のバグの問題にアタックする道が開かれていたのである。

また、UML はソフトウェア技術者の基礎知識のひとつとして急速に普及しだしていた。これは良いものだと思った私は、数学出身の院生などは証明アニメーションの研究に残し、それ以外の研究室の全学生をあげて UML のモデラ（モデル作成用ツール）SMART システムを開発するプロジェクトを始めた。SMART では、モデルが実行可能であり、それを利用して当時流行していたテスト駆動開発を使ってモデルを開発できた。

形式的検証や UML などを使いプロジェクト初期に全体の労力の大半を投資して、仕様やモデルを極力最終版に近づけ、後でそれを極力修正しないで済むようにする方法をアップフロント開発という。テスト駆動開発というのは、このアップフロントの対極に位置するアジャイル開発の技法の一つだ。

アジャイル（迅速）開発では、極端に言うと何も考えずに開発を始め、不具合を見つけては、それを修正してシステムを改善していくことにより開発を進める。テスト駆動開発では、その不具合を見つけるためにテストを使う。システムのユニット（小さなサブシステム）をテスト用の状態に設定し、ユニットを動作させ、結果が期待される結果に一致するかを自動的にチェックする「テスト・ケース」というプログラムを作り、これを動作さ

せてはエラーメッセージを見ながらシステムを開発していく。

ロボット・アームの制御プログラムで言えば、そのプログラムで実際の制御を行う前に、徹底的にプログラムを検査してバグを追い出すやり方がアップフロント、不完全なプログラムにロボットを制御させ、起きた不具合を見てプログラムを改善していくのがアジャイルにあたる。ロボットでアジャイル法をやったら、ロボットが壊れるが、ソフトウェアの場合は、エラーメッセージがでるだけだから、こういう方法が使えるのである。

アジャイル法は小さいサイクルで改善を繰り返すため開発が常に前進しているという感覚を得られエンジニアにやる気を起こさせる。この様に、アジャイルは日本発の生産方式 KAIZEN を思い出させる。実は、アジャイル法の提唱者の多くはトヨタ生産法や日本型経営に影響を受けているのである

SMART の特徴は、水と油と思われていたアップフロントとアジャイルを融合した点であったが、そのアイデアの元は、実は証明アニメーションであった。SMART は、常時実行のトレースを保存する仕組みを持ち、バグが起きた際には、その状況を自動的に図示し、またトレースとテスト・ケースを組み合わせて自動的にモデルの修正の仕方をエンジニアに提案する意味論的クイック・フィックスという機能を持っていた。これも実はトヨタ生産法などをヒントにして考えた。

この機能のため簡単なモデルの場合は、テスト・ケースを少し書くだけで、モデルが殆ど自動的に構築できた。このプロジェクトだけは、参加した優秀な学生・院生たちのお蔭で、私のソフトウェア工学人生のなかでは、唯一、成功しつつあったプロジェクトだったが、このプロジェクトの半ば、私は神戸大工学部を辞めて、京大文学研究科に転職したために、これが継続できなくなったのは、実に残念なことだった。

人文社会学ターン

この時期の私の変化は狭い意味でのソフトウェア工学内部に留まらなかった。ある日、大学の書店で平積みなっている本を何の気なしに取り上げた。それはアメリカでベストセラーとなった社会学者ジョージ・リッツアーの「マクドナルド化する社会」の和訳だった。少し読んでみて驚いた。ソフトウェア工学や IT と関係がない社会学の本のはずなのに綴られていたのは、私が直面していた問題と非常によく似ていたのである。当時、私は、仕様・モデルのバグの問題は、仕様アニメーションや SMART の様な技術的方法だけでは解決できないのではないかという疑問を直観的にもちつつあり、この本が、その直観を理論化するためのヒントになりそうに思えた。

早速、この本を読み、さらにその背景であったウェーバー社会学を学んだ。丁度、この少し前から、私の生活は、昼は工学者、夜は歴史家という風に変化しつつあった。龍谷大学時代に引き受けた岩波文庫の仕事に本格的に着手したところ、最初の専門だった数理論理学の歴史の通説が矛盾や誤りだらけであることに気付き、中学生のころからの歴史好き

の虫が動き始めてしまったのである。この話は、以前あるところで書いたので省略するが、[1] 実は、その歴史研究とウェーバー社会学は深い関係があったため、工学者ながら、難解なウェーバー社会学を簡単に理解することができた。

その結果、verification は、ウェーバー社会学の形式的合理性の概念に、validation は、実質合理性の概念に対応することに気が付いた。これは仕様・モデルのバグは、文化・価値のような社会的ファクターにまで関係する問題であることを意味していた。仕様のバグの問題を突き詰めると、最後は工学的手法では対処できない問題が出てくるはずだという私の直観はウェーバー社会学により理論化されたのである。この発見以後、私は、ソフトウェア工学を社会学と関連付けて思考するようになった。

科学技術政策研究所・動向センター

私が社会学ターンを行いつつある、ある日、1980 年代の通産省による第 5 世代コンピュータ・プロジェクトで知り合った黒川利明さんからメールが届いた。それによると、現在は CSK のフェローだが、文部科学省の研究所である科学技術政策研究所　NISTEP（現在は科学技術・学術政策研究所）の研究員も務めていて、その関係で、私が最近何をしているか知りたいという内容だったと思う。そこで、SMART プロジェクトを説明するメールを書いたのではないかと思うのだが、よく思い出せない。兎に角、NISTEP で何か話せということになった。しかし、SMART の話をしても聴衆は理解できないだろうし、面白くないだろうから、科学技術政策立案をしている人たちとって面白いだろう話をすることにした。

仕様のバグの問題と並んで、その頃の私を悩ませたのは、日本のソフトウェア産業の問題だった。不況にも関わらず学生たちは有名企業のソフトウェア関係の職を射止めてきた。ところが、卒業生たちの話を聞くと、どうも変だ。その当時の学生の一番人気の企業に、厳しい競争を経て学科推薦で入社した院生が、後輩のプロモーションのために返って来たとき、ひどく寂しそうな顔をみせた。上司が非ソフトウェア部門の出身で、私に教わったソフトウェア工学の原理を理解してくれない、上申書のような物を書いたが駄目だった、というのである。辞めようかと思いつめる教え子を前にして、私は返す言葉がなかった。そして、他にも、同じ様な事例を卒業生たちから聞かされた。ある中堅企業のソフトウェア子会社に就職した院生などは、休職して学位を取得する予定だったのに、休職直前に働き過ぎでパニック症候群になってしまった。

日本のソフトウェア部門が何かおかしい。自動車産業などでは、日本は世界に伍することができるが、ソフトウェアは全く振るわない。そして、その中で働く教え子たちは苦しんでいる。どうしてだろうか。どうしたら、この問題を克服して、日本に他の産業部分と比較しても恥ずかしくないようなソフトウェア産業にできるのか、そういう問題を私は考え続けていた。

---

[1]「情報学者の人文学研究」. 以文、京大文学部・文学研究科同窓会誌、2005 年号、http://www.shayashi.jp/HI_DEP/ibun.pdf.

この問題は、大変に難しかったが、先に述べた V&V とウェーバーの合理性理論の関係から解決の糸口が見えつつあった。そこで、SMART で融合したアジャイルとアップフロントの関係を、スパイラル開発の提唱で有名なバリー・ベームが、軽快な猿（アジャイル）と重厚な象（アップフロント）の関係として説明する面白い寓話を作っていたので、この話を使い、V&V に対応する「形式合理性と実質合理性」のペアを「象の合理性と猿の合理性」と説明し、さらには、日本型の生産や経営は猿の合理性であることからトヨタの様な非ソフトウェア産業に学んで、日本のソフトウェア産業を振興させるという案を考え、これを話した。

これは大変に好評で、NISTEP の動向センター（正式には科学技術動向センター）の客員研究員となって、この講演をレポートに纏めよということになった。その結果、月に 1, 2 度、動向センターが出版している月刊誌の編集会議に出る様になったのだが、そこでの議論は、人工衛星から、砂塵（黄砂）の増加傾向の話までの最新の話題だった。この会議で聞いた話が、半年や数年後に世間の話題となっていたのである。雑学大好きの私は、楽しくて、普通は大学の先生は、あまり出ないらしいこの会議に頻繁に参加した。特に社会学ターンをして以来、リッツァーに倣って技術や社会現象の社会学的背景を探ることが性癖になっていたが、その格好の材料が、この編集会議ではゴロゴロ転がっていた。

また、この雑誌「科学技術動向」は、国の科学技術政策の現場で働く人たちに資するために発刊されていたので、日本のソフトウェア産業の問題を改善する助けになれるかもしれないと思ったのも、この仕事にのめりこむ大きな理由であった。

結局、NISTEP 動向センターには 6 年近く在籍し、その間に三つのレポートを書いた。そして、この三つのレポートを書くための調査が、私に最後のタガを外させていった。

三つのレポート

実は、動向センター最後の 2 年ほどは活動を完全に停止していた。この活動の停止の理由こそ、実は、本論で最も伝えたいことなのである。その話をするために、この科学技術政策研究所・科学技術動向センター発刊「科学技術動向」の三つのレポートがどんな内容だったか説明しよう：

第 1 レポート：林晋、黒川利明共著、二つの合理性と日本のソフトウェア工学

第 2 レポート：林晋著、情報通信技術と「思想」 ―科学技術の能力としての「思想」―、

第 3 レポート：林晋、黒川利明共著、日本の危機としてのＩＴ人材問題

それぞれ、2004 年 9 月号、 2006 年 10 月号、2008 年 7 月号の掲載である。

第 1 レポートの内容は、すでに述べた講演の内容と同様に、Verification/Validation=形式合理性/実質合理性の見方をもとに、アジャヤイル開発とトヨタ生産方式(TPS)の類似性を指摘し、TPS が生まれた日本では、その知恵や人材を IT に流れ込ませることにより、プロセス改善、ひいては IT 産業の振興を一気に行うことが可能ではないか、という提言をすることだった。つまり、これはソフトウェアという人工物を、車という人工物と同じ様なものと考え、別の業界の生産方法をソフトウェア業界でも真似ることにより、その劇的改善

の可能性を指摘したものだった。アジャイルを学ぶにも、日本のソフトウェア業界には、米国のようにそれを教える人材がいない。しかし、機械工学の世界には、日本にも、そういう人材があるはずだ、その人たちに学ぼうという提言であり、後の二つのレポートで掘り下げることになる IT 人材の問題への意識の端緒が見られる。

第 2 レポートでは、そのような人材問題などへの言及はあるものの、飽くまでソフトウェア・プロセスの改善に主眼があった第 1 レポートと異なり、プロセスを担う人の問題に視点が移った。トヨタでは社員が「洗脳」されいるが故に TPS が機能するように、プロセス（生産工程）を担う人たち、つまり、IT 技術者のマインドを変化させないと良いプロセスは実現できない。そのためには、プロセスを理解するだけでなく、その背景にあるトヨタウェイのような「思想」まで理解して身に着けないといけない。そういうことを、IT におけるオープンの思想から、明治時代のお抱え医学者ベルツの自然科学についての演説なども交えて議論した。

その最後の方で私は、IT の能力、特にソフトの能力が、殆どの産業の競争力の源になる時代が到来するのではないか、その時に、「ソフトウェアに弱い日本」は、「科学技術、産業に弱い日本」を意味するのではないか、という危惧を書いた。これは、動向センターで聞いた NISTEP のデルファイ調査などの話を聞き、また、動向センターの客員研究員になったお蔭で様々な産業での動向を目にするようになった結果、持ち始めていた危惧だった。

動向センターの同僚の話では、確か 2010 年代半ばには、もう情報分野というのは、個別の重要分野ではなくなり、すべての分野のエンジニア、研究者たちが、自分で IT を駆使する様になると多くの技術者・研究者が思っているらしかった。そして、その時代は到来しつつある。あるいは、世界的には既に到来している。IT の手法で EV を手掛けるテスラ・モーターズや３D プリンタは、その象徴である。

そして、この問題意識が最後のレポートに繋がることになった。第１，２レポートのような提言を実現するには、まず人材が第一である。そういう人材を確保しないと、IT 分門だけでなく、日本の産業全体が沈みかねない。それは目の前の危機である。では、どの様な人材を、どうやって、この国で育成するのか、その問題に踏み込んだのが、このレポートだった。

第 3 レポートと深い挫折

この第 3 レポートが、動向センター時代に、もっとも力をいれたものだ。私は、共同研究者の黒川さんと共に、日本やアメリカの先進的な IT 教育の事例を探し出し、それらの幾つかを取材した。また、黒川さんの人脈を頼り、企業の方々にもコンタクトをとって議論を重ねた。

それらの活動の中で、公立はこだて未来大学の学部の教育システムを取材したときのことである。各研究室が、こんなツールを作ってほしいという「注文」をだし、それを学生がグループを作って制作し、出来たものは実際に研究や教育で使われるという教育システムを、この大学は実施していた。その教育コースを中心になって動かしていた鈴木恵ニ助

教授(当時)に話を伺ったときのことである。

この取組は非常に成功していて、こういう時は、非 IT 系の教員などが水をかけて来るものなのだが、そういうことが無い。非 IT 系の研究室も自分たちのためになるツールが出来るものだから、むしろ後押してくれるという話を伺って、これは良い人材育成モデルになると考えつつも、ある懸念が心に浮かんだ。自分の SMART プロジェクトなどでも、学生たちの学習の活性化とスキル・アップが見られたが、学生たちが社会に出ると、上司に上申書を書いた私の研究室の卒業生の様に、そこで水をかけられてしまうことを思い出したのである。教えたことと社会の現状との乖離が、むしろ、学生のストレスに繋がる。

それを思い出し、そういう質問をしたところ、それまで楽しそうに説明をしてくださっていた、鈴木氏が悲しそうな顔になり、ある事例を話してくださった。

この教育システムで育った学生が、ある鉄鋼グループの子会社の S ソリューションズに内定した。この会社は中堅システム・インテグレータの中では評価が高い会社であったし、どうやら、鈴木氏の教育システムを背後からバックアップしていたようである。だから、学生や鈴木氏は、この内定に大変喜んだらしいのだが、学生の父親が、独り言のように「S と言っても鉄を作ってるわけじゃないのか…」と言って、それを聞いた学生が大変に落ち込んだというのである。

この時、突然、私の心の中にあるイメージが浮かんだ。それは太平洋戦争末期、沖縄沖に停泊する米海軍艦隊に勇猛に突っ込みながらも、猛烈な弾幕に次々と撃ち落される特攻機の映像だった。その操縦席には若者たちが座り、そして、その背景には、それを送り出した上官たちがいる。自分たちは、その上官と同じことをしているのではないか。

幾ら人材を育成しても、それは社会の中の人材なのだから、社会がそれを受け入れないのでは、その人たちを傷つけてしまうだけだ。日本の IT の問題を、人材を生み出す人材育成の問題だと誤解し、それを改善すればすべてが解決すると安易に思っていたのである。しかし、これは人材の受け手側の問題でもあり、実は、そちらの問題の方が大きいのだ。どのようにプロセス改善や人材育成で頑張っても、受け手側が水をかけて消してしまう。これでは湿ったマキに火をつけようとする行為と変わらない。徒労ではないか…。

実は、その瞬間は「撃ち落とされる特攻機」の映像を思い浮かべて、少し背筋が寒くなっただけだったが、宿に帰り、京都に返り、何故、そういう映像が心に浮かんだのか分析しなおして行くと、そう思えたのである。そして、この徒労感は、はこだ未来大の取材の 2 週間ほど後のスタンフォード大学デザイン・スクールの取材で、さらに絶望感に変わった。現在の日本社会で IT を強化することは不可能だ、そう思うようになったのである。

スタンフォード大学デザイン・スクール

ソフトウェアを、ハイデガー哲学などを使って、社会的存在として理解するアプローチがあり、これを始めた人が、タンフォード大学教授ティム・ヴィノグラード氏である。氏は若いころは AI 推進派で、その可能性を実証する史上最初のシステムを作ったのだが、その後、丁度、日本の第 5 世代コンピュータ・プロジェクトの前期が終わるころ、AI や自然

言語認識の不可能性を解く著書『コンピュータと認知を理解する － 人工知能の限界と新しい設計理念』を出版し、この陣営を去り、CHI とか HCI と呼ばれる、人間とコンピュータのインタラクションを重視する、ある意味では、当時の AI の反対勢力に鞍替えしてしまった人である。

IT 人材育成の調査を進める一方で、私は、その第 5 世代コンピュータ・プロジェクトのリーダであった渕一博さんの生涯と、第 5 世代コンピュータの経緯を歴史学者として追っていた。これは渕さんが急逝され、その追悼集会での基調講演を依頼されたためだが、AI 分野には疎かったため、この調査の中で、私は始めてヴィノグラード氏の「デザイン思考法」と呼ばれるソフトウェアの社会的側面を考える思考法を知った。それはアジャイルによく似た思考法で、ヴィノグラード氏の転向は、形式的検証からアジャイルに鞍替えした私の転向を、15 年ほど先取りして進んでいるように見えた。

そこで、黒川さんの人脈を頼って米グーグル本社を取材する際、デザイン・スクールとヴィノグラードさんの取材もさせていただくことになった。ヴィノグラード氏の助言に従い、まず、デザイン・スクールの実際の授業を拝見したのだが、デザイン・スクールは想像以上にアジャイルに近いマインドで運営されていた。ビデオ作品作成が本職の准教授の方に、何を教えているのかと聞くと「失敗の仕方を教えている」ということだった。まずはやってみろ、失敗は上手く受け身して、その失敗から情報を得て、次に進め。まさにアジャイルの精神そのものだった。私は嬉しくなって、この様な教育システムを日本に導入すれば、日本でも IT 人材が育つのではないかと思った。

このデザイン・スクールという大学院レベルの教育機関は、ドイツのソフトウェア会社、SAP の創業者の一人ハッソ・プラットナー氏の寄付により運営されており、ドイツの IT 人材育成に問題を感じたプラットナー氏が、パロアルト流の思考法を育成するシステムを作り、さらには、それをドイツに移入することを目的に設立したようだった。実際、ドイツ、ポツダム大には、ドイツでもグーグル社のような会社を生む人材の育成を目指して、ドイツ版のデザイン・スクールが設立されていた。

日本でも、政府とか経団連のような業界とかが、よく似たシステムを作ることは可能だろう。ヴィノグラード氏を日本政府か財界が招請して、日本版デザイン・スクールを作る、そういう提案を第 3 レポートに書こう、私はデザイン・スクールの、セッションと呼ぶ方がよい、授業風景を見ながら、そう思った。

その後、ヴィノグラード氏にインタビューさせていただいた。この人は、グーグルの創業者のラリー・ペイジの副指導教員で、グーグルの成功物語を書いた書籍などでも、よく登場する。私は、そのグーグルの企業運営にデザイン思考法的なものを強く感じていた。だから、最も聞きたかったのは、ペイジの思考法はヴィノグラード氏の思考法に影響を受けたのではないかということだった。

この質問は、氏にとって意外だったらしく、驚いたような顔をされて 2, 3 秒考えた後で、「それは違う。ラリーも私もパロアルトの精神に影響されたのだ」と答えた。これは感激

的な答であると同時に、後で私を絶望に追いやる言葉でもあった。しかし、そのことには、その時には気づかず、何か心に引っ掛かるものを感じながら、別な質問をした。それはスタンフォード大学機械工学の熱力学の専門家、スティーブ・クラインを知っていますか、という質問だった。

この人は、アメリカが日本の産業に劣性だったとき、その原因を探り対処を考える委員会の一員だった人で、その中からチェーン・リンク・モデルというものを提唱し、それがアメリカの製造業の復活に資したと経営学の世界などでは言われていた。このクラインの思考法は、デザイン思考にそっくりだった。しかも、その提唱時期が、ヴィノグラード氏のデザイン思考の提唱と、ほぼ同時期なのである。同じスタンフォード大学の工学部の教授であることから、きっと、この二人には人的繋がりがあるに違いないと信じ、クラインをご存じか聞いてみた。

驚くことに、ヴィノグラード氏は、どうもクラインを知らないようだった。そこで、クラインの提言したことをまとめたWEBサイトを見せた。それを読んだ途端ヴィノグラード氏は、大変嬉しそうに「私の考え方と同じだ！」と声をあげたのである。

ほぼ同じ思考法が、面識のない二人のスタンフォード大学に、ほぼ同時期に生まれる。これは、「ラリーも私もパロアルトの精神から影響を受けたのだ」というヴィノグラード氏の言葉の正しさの何よりの証拠のように私には思えた。

絶望的結論と第3レポート

私が、この取材で経験したことの意味を本当に理解するには、少し時間がかかったが、はこだて未来大学での取材と、このスタンフォードでの取材から、次のような結論を引き出した。「パロアルトで多くの優れたIT人材が育っているのは、Xerox PARCや、スタンフォード大学のような優秀な人材が集まる拠点が存在するという箱もの的な側面もあるが、何より重要なのは、広く根付いている失敗を恐れない文化だ。しかし、そういう文化的なものは、明治維新後の開化運動のように大規模かつ徹底的に行わねば成功しない。たとえ特区のような特定地域で実現したとしても、そこで育った人材は、その外に出た途端、水をかけて火を消そう、それにより現在の自分の世界を守ろうとする日本社会に撃ち落とされてしまう」だった。

これは実に絶望的な結論だった。また、この結論は、日本社会のために少しは役立つことを願って行っていた動向センターでの活動は役に立たないということも意味していた。人材育成の前に、まずは、日本社会、その成員の考え方を根本的に変えてなくては、どうしようもない、そうでない限り、日本ではITが育たない、というのが私の結論なのである。科学技術政策というようなものでは、どうしようもないのである。

この結論に至った時には、本当に絶望した。あまりに暗い結論だったので、動向センターの同僚には話さなかった。パロアルトに同行した黒川さんにさえ暫くの間は隠していた。私は、本心を隠し、取材先でみた素晴らしい取り組みの幾つかを紹介し、また、デザイン・スクールが目指すような人材を「新T型人間」と名付け、当たり障りのない範囲で、第3

レポートを書き終え、その後は動向センターでの仕事をバッタリ止めた。

ソフトウェア：数学的存在から社会的・文化的存在へ

以上で、私の失敗続きのソフトウェア工学人生の話を終える。まとめとして、これが何だったのか分析してみよう。それにより、何故、日本のソフトウェアが弱いのか、では、どうすべきか、ということも自ずと解るはずだ。

私の変化は実はＩＴの歴史的変化を少しずつ遅れながら追いかけている。初期のコンピュータの目的は弾道計算などの数学の問題を解くことだった。形式的検証時代の私がソフトウェアを数学の関数として理解していたのは、その時代なら、あながち間違ではなかった。この時代の主要アプリは、数値計算、制御、旧タイプの DB などであり、その特徴は、ユーザがエンジニアなどの専門家であることと、ユーザとのインタラクションが少なく、それに比して正確性・精度などが求められたことである。それは、この時代のソフトウェアの応用対象が、自然科学の様な比較的安定した法則を持つものが中心であったからである。だから、その仕様も容易には変わらないものだった。だから、アップフロント開発というものが意味を持ち得たのである。この時代の、ソフトは機械装置の様に作られていた。

しかし、コンピュータの応用ターゲットは、数値計算からオフィスツールなど様々なものに広がっていった。とはいうものの、その拡張の初期には、PC がかなり普及して、仕事以外にコンピュータが使われることも増えたとはいえ、それらのユーザインタフェースは技術者やオフィス・ワーカが使用する前提で作られたウィンドウシステムを、少し一般向けに変えたものだったという意味で、基本的には専門家が仕事に使う道具であることに変わりはなかった。

UML やアジャイルは、そういう時代の最後を飾るものともいえる。システムが大規模化して大勢の技術者でシステムを作ることが当たり前となり、そのために共通言語が必要となる。それが UML だった。また、ビジネス向けのシステムは機械装置の様に作ることは不可能だ。機械装置や自然科学の法則と異なり、事業所というものは、その運営の仕方は生き物のように変化して行く。

また、大きな事業所の全体像を隅から隅まで把握することなどもともと不可能だ。当然、それをサポートするシステムに要求される要件を最初から全部書き出すなどということはできない。それはシステムを使ってみて初めてわかるのである。だから、昔は悪とされた仕様の変更にもアジャイルに対応できることが求められる様になったのである。その主たる応用ターゲットが変化するに従い、システム開発の在り方は、大きく変化していったのである。私が神戸大時代に開発していた SMART などは、まさに、この変化を体現したようなシステムであったといえる。しかし、それはある時代の終わりの象徴であった。

私の NISTEP でのレポートは、第 3 レポート執筆時には、すでに「社会とソフトウェア」という本質的問題に気づきながらも、それを隠したために、第 3 レポートも含めて、すべて IT という「専門技術」についてのレポートとなっている。第 1 レポートがソフトウェアの生産技術の話、第 2 が生産を担う技術者のマインドの話、そして、第 3 が、その技術者

をどうやって育てるかの話という風に段々と、技術そのものからは離れて行ってはいる。しかし、第 3 レポートを企画した最初の段階には、IT 人材をあたかも「ソフトウェアを生産するための装置」の様に考えていたことは否めない。その「装置」を作る為の良い「生産装置」をパロアルトから移入すれば、日本の IT の問題は解消する、その様に考えたのである。

しかし、それは間違えていた。ヴィノグラード氏とのインタビューから私が学んだことは、グーグル検索の様な優れたソフトウェアやサービスを生み出すのは、社会や文化なのである。それが持つ「精神」なのである。そして、IT は、装置の時代から、社会の時代に実際に突入したのである。

今や、IT は我々の生活の隅々にまで浸透し、仕事や消費の形だけでなく、人間のつながり方まで変えてしまった。スマホを持った子供たちや若者は、我々には理解できない形態でコミュニケーションしている。日本最初の UNIX システムのユーザだった私は、日本の最初期の電子メールやネットのユーザでもあった。しかし、私にも、今の若者や子供たちの IT を使ったコミュニケーションの方法は理解できない。IT は文化・社会を変えてしまったのである。

そういう時に、新しいものを理解できない旧世代は、新しいものに抵抗しようとする。「なんだ鉄を作っているのではないのか」という一言は無意識の内の自分が理解できないものへの抵抗と見るべきだろう。日本社会という安定・安穏に寄りかかりたいという傾向を強く持つ社会が、現代の様に満ち足りた状態に置かれていれば、新しいものを拒否しようとするのは当然である。

これに対して、パロアルトが象徴するアメリカのある部分は、如何に豊かであろうとも、常にイノベーション・革新を求めて、先に先にと進もうとする。よしんば、世界中の富の大半を手中に収めていても、もし富をさらに拡大できるのならば前に進む。前に進むこと、努力すること、変わることが善だ、今可能なのに、それをしないことは怠慢であり罪悪だ、そういう彼らの倫理観・道徳観が、そうさせるのであり、金銭欲がそうさせるのではない。日本の IT の問題は、実は技術の問題ではなく、文化・社会の問題だったのである。

社会技術としての IT

WWW やスマホ、タブレットなどが登場して、IT の文化的・社会的性格は、さらに増加している。WWW 初期の主役が、WEB ページ、ブラウザ、メーラ、e-コマースの様に個と大規模組織を繋ぐものであったのが、ブログ、SNS、Skype、Twitter、LINE のように個と個が繋がるためのツールへと変化していったことは、その象徴である。IT は公的レベルから、さらには生活の私的レベルにまで浸透した。この傾向は、さらに進むだろう。

IT は生産、流通、交通、通信、販売、医療、福祉、治安、軍事、さらには人間関係など人間社会のあらゆる側面やレベルに浸透している。IT は社会そのものになりつつある。IT は単なる個別技術でなく、社会そのものを体現する、社会技術なのである。その IT のあり方を変えるということは、社会を変えるという事に他ならない。

もし、日本の IT の弱さ、IT 人材の不足の問題を変えたいというのならば、それは日本社会そのものを変えるしかない。そして、それに貢献することは「科学技術政策」の研究所では不可能なことであろう。私はその様に考えて NISTEP 通いをやめた。

私は大学教師である。教師は次代を担う若者たちに何かしらのメッセージを送ることができる職業である。若者たちの、そして、社会のマインドに風を送り続け、いつの日か、この社会の心の薪が乾き、それに火が付くことを夢見て、何十年かかろうと、自分の寿命が尽きようと、だれかが私に代わって湿った薪に風を送り続けてくれることを願いつつそれを続ける。もし、私が日本の IT、いや、日本社会のために何か貢献できるとしたら、それしかできることはない。私は、そう考えて、今日も微風を送り続けている。