第１９期　情報化推進懇話会
第３回例会：平成１７年６月２３日（木）

『ソフトウェア最前線
情報サービス産業界に革新をもたらす真実』

講　師

富士通総研　経済研究所

主任研究員

前川　徹　氏

財団法人 社会経済生産性本部
情報化推進国民会議

『ソフトウェア最前線

情報サービス産業界に革新をもたらす真実』

― プロフィール ―

富士通総研　経済研究所
主任研究員
前川　徹　氏

◆ 略　　歴 ◆

1978 年名古屋工業大学情報工学科卒業。78 年通産省入省。機械情報産業局情報政策企画室長、JETRO New York センター産業用電子機器部長、情報処理振興事業協会、(IPA) セキュリティーセンター所長、早稲田大学　大学院　国際情報通信研究科客員教授（専任）を経て、2003 年 9 月に（株）富士通総研に入社。経済研究所　主任研究員。早稲田大学　国際情報通信研究センター　客員教授、国際大学　グローバル コミュニケーション センター　客員教授を兼任。

◆ 著　　書 ◆

『サイバースペースとアメリカ情報産業』　（株）スパイク　1997 年 6 月
『ネットビジネス最前線』　（株）スパイク　1998 年 5 月
『EC ビジネス最前線』　（株）アスペクト　1999 年 11 月
『ネットバブルの向こう側　ＥＣビジネスの未来戦略 』　（株）アスペクト　2001 年 7 月
『IT 革命を読み解く（共著)』　技術評論社　2001 年 11 月
『次代の IT 戦略（共著)』　日本経済評論社　2002 年 10 月
『サイバージャーナリズム論（共著)』　東京電機大学出版局　2003 年 10 月
『ソフトウェア最前線―日本の情報サービス産業界に革新をもたらす 7 つの真実』（株）アスペクト　2004 年 9 月

『ソフトウェア最前線 情報ｻｰﾋﾞｽ産業に革新をもたらす真実』

１．背景あるいは問題意識

我々の社会、生活は、現在その多くをソフトウェアに依存しています。例えば車の運転でアクセルを踏んだときには、その踏み込まれた量をセンサーが検知し、エンジンの回転量や温度、スピード、外気温などの条件を見て、噴出量と着火のタイミングを決めているのです。ですから、プログラムに不具合があると大変なことが起きてしまいます。先日もプリウスで不具合が見つかったようですが、過去には、ダイムラー・クライスラーでは最先端の電子制御ブレーキに問題があり 68 万台がリコールされるという事件が起きていますし、ＢＭＷは高速走行中にエンストを起こす、ジャガーの場合は突然ギアがバックに入るというソフトウェアの不具合が発見されたこともあります。つまり、自動車は運転者が直接操作しているのではなく、実はコンピュータ制御されているのであり、そのコンピュータはソフトウェアで動いているのです。また、これは電力、ガス、銀行のシステムや洗濯機などの家電も同じことです。

ここで厄介なのは、ソフトウェアのトラブルはリカバリーが難しいということです。ハードウェアの場合には、予備のシステムをホットスタンバイさせておいてトラブルがあったら切り替えることが可能であり、ネットワークも非常用の回線を１本引いておけば、通常使っている回線が切れても大丈夫です。しかし、ソフトウェアの場合はそうはいきません。昔、所沢の航空機の運行管理をしているコンピュータが故障したときには、ホットサーバにすぐ切り替えたにもかかわらず、同じソフトウェアが動いていたため同じところでコンピュータが停止し、復旧するまでに１日近くの時間を要したことがありました。

そういう事態を防ぐためには、別系統で同じプログラムを二つ作っておかなければなりません。それはコスト高だというなら、前に確実に動いていた古いバーションを一つ用意しておくという方法もあります。また、組み込みソフトの場合ですと、不具合があったものは全部回収しなければなりません。いずれ家電などは全部がインターネットにつながって、バグがあったものは自動的にアップデートするようになると思いますが、今はほかに方法がないのです。

過去に起こった実際のトラブルの例を見てみますと、金融機関が非常に多いのですが、航空管制システム、ＪＲ東日本の運行管理システムもトラブルを起こしたことがありますし、東京証券取引所のシステムも 2003 年 10 月にソフトウェアの不具合でトラブルになっています。また、組み込みシステムの場合、いちばん多いのはやはり携帯電話ですが、ほかにはデジカメ、ハードディスク内蔵型のＤＶＤレコーダーのソフトに異常が見つかったという例もあります。

２．日本のソフトウェアに国際競争力はない！

これだけ現在の社会ではソフトウェアが重要になっているのですが、日本のソフトウェアの国際競争力はどうなっているのでしょう。業界団体の計算によると、2000 年の数字で輸入が約 9000 億円､輸出が約 90 億円で､圧倒的に輸入が多くなっています。この数字から考えると、日本のソフトウェア製品はその機能や品質がものすごく悪いか、値段がものすごく高いかで国際競争力がないということになります。

こういう話をすると、ＯＳ、ワープロ、表計算、ＤＲＰのパッケージ、データーベースはみんな欧米製だから当たり前だと反論される人がいます。1995 年と 2000 年の輸入を調べてみると、確かにベーシック（ＯＳ）、アプリ（ＯＳ以外のパッケージ部分）も伸びていますが、いちばん伸びているのはカスタムメイドの部分なのです。つまり、受託生産型のソフトウェアの輸入がこの５年間に８倍になっているのですが、これは海外にソフトウェアを委託して開発するケースが増えていることを意味しています。やはり日本のソフトウェアには国際競争力がないのです。

ところが、マイケル・クスマノ教授が 2002～2003 年にかけての日本、米国、インド、欧州、イスラエルのソフトウェア開発プロジェクトを調べたところ、１人のプログラマーが１か月に書いたコード行数は日本 469、米国 270、インド 209、欧州他 436 でした｡また､リリース後 12 か月に発見された 1000 行当たりのバグの数は､日本 0.020､アメリカ 0.400、インド 0.263、欧州他 0.225 と、日本はアメリカの 20 分の１となっているのです。つまり、この数字を鵜呑みにすると、日本のソフトウェア開発は生産性も高いし、品質もよいということになってしまいます。

実は、この数字にはまやかしがあります。コード行数の測定の分母は１か月であって、何時間にではありません。もしかすると、アメリカのプログラマーが１週間に 40 時間しか働いていないのに、日本のプログラマーは残業や土日出勤でその３倍ぐらい働いているのかもしれないのです。また、生産性をコード行数で測っていいのかという問題もあります。つまり、同じ機能を持つプログラムを、上手な人なら早く短く組めるのに、下手な人は時間をかけて冗長で効率の悪いプログラムを組むということもありえるのです。また、バグの数は、利用者がどれだけいるかによって発見される数が変わってきます。

３．ウォーターフォール・モデルの仕組みとその問題点

私が日本のソフトウェア産業について研究を始めて、すぐに気付いた問題が、ウォーターフォール・モデルの問題でした。つまり、お客さんからどういう業務をシステム化するかという要求を聞いて、要求仕様書を作る。それに基づき、プログラムはこういう機能を持つべきだという全体の概念設計を行う。それからモジュールを切り出

していって、それぞれのモジュールにどういう機能を備えるべきかという詳細設計（内部設計）を行う。その設計書に基づいてプログラムを組み、単体でテストする。幾つかのプログラムを結合してテストする。最後に全体のシステムのテストをする。そして、運用・補修に入っていく。これが、ウォータフォール・モデルです。素人目には大変分かりやすいモデルだと思います。

ところが、ソフトウェア開発分野でバイブルになっているフレデリック・Ｐ・ブルックスの『人月の神話』には、「ウォーターフォール・モデルは、1975 年にたいていの人が抱いていたソフトウェアプロジェクトについての考え方だった。だから、不幸にも DOD-STD-2167 というすべての軍事ソフトウェアに関する国防総省の仕様書に記述され祭り上げられてしまった。そのために思慮深い専門家のほとんどが不適切さに気づき捨て去ってしまってからも、なお生き延びた。幸いなことに、国防総省もそれ以後ようやく気づき始めたようだ」と書かれています。実際、国防総省は 1990 年代半ばにウォーターフォールで開発してはいけないという通達を出しているのですが、日本にはそのニュースはほとんど伝わらなかったようです。

では、何が問題なのでしょうか。ウォーターフォール・モデルの問題点をＶ字型の絵にしてみました。要求定義→機能設計→モジュール設計→コーディング→単体テスト→結合テスト→機能テスト→受入テストといくこのモデルでは、コーディングのミスは単体テストをするときに発見されます。同じように、結合テストではモジュール設計のミス、機能テストでは機能設計のミス、受入テストでは要求定義のミスが明らかになるわけですが、かりに要求定義段階で何か誤りがあった場合、それが発見されるのは最終の受入テストのときになってしまうのです。つまり、その意味で、このモデルは大変危険なモデルなのです。

４．ウォーターフォール・モデルの改善

この危険性は昔から多くの人たちが指摘しているところで、1981 年に出た Boehm の本では、要求定義の誤りがもたらすコスト増について、要求定義のときに気づいたときを１とすると、設計段階で３～６、コーディング段階で 10、開発テスト段階では 15～40、受入テスト段階では 30～70、稼働段階では 40～1000 と見積もっています。つまり、ウォーターフォールでソフト開発をすすめ、最後のユーザー受入テストの段階でミスが発見されると、要求定義の段階へ戻ることになってしまうということです。

しかし、考え方を変えて、最初から手戻りがあるというスタイルにしたらどうかというのが私の提案です。つまり、ウォーターフォールを２～３回繰り返すことを最初から考えてやるということです。この３月に動き出した佐賀市の電子自治体の新システムでは、韓国のサムスンＳＤＳが、住民基本台帳から始まる納税業務などの基幹業務のシステムを、普通のベンダーなら３年近くかかるところを１年ちょっとで開発し

てしまいました。サムスンは最初にプロトタイプとして印鑑登録のシステムを作り、オープンシステムでの稼動を確認した後、第１回目の開発を行いました。開発を開始したのが一昨年の 12 月で、完成したのが去年の７月なのです。しかし、その１回めの開発でできたものは、完成度はあまり高くなかったそうです。例えば、韓国の自治体では住民が転入届を出せば転出は自動的に行われるようになっているため、転出処理がなかったなどの問題があったのです。それでサムスンの主要部隊を佐賀市に連れてきて２回めの開発を始め、この３月に動き出したのです。

つまり、早く作ってユーザーに見せて違っているところを直していく、何回もスパイラルをやるという方法も一つあるのです。我々は、よく分からないことについて、ＰＤＣＡ（Plan Do Check Act）でやるという癖があります。ソフトウェア開発も、最初から完ぺきなものを作ろうなどということはそもそも間違いなのです。何回も繰り返してどんどん良くしていくことによって、本当にいいものができるのだと思います。

ですから、最初から完全な仕様書なんて作れるはずがない、仕様書はそもそも仮説であり、それを検証するのがプログラムだと考えた方がよいのだと思います。必要なら、仕様書はソフトウェアができ上がってから作ればいいのです。クスマノも次のように書いています。「従来のソフトウェア工学では、終盤の設計変更は悪いことであり、仕様が誤っていたとか不完全であったことを示唆した。だが、ここでの理念は、製品作成の過程で、ユーザーからのフィードバックにこたえることだ」。彼が唱えているのは、synchronize and stabilize の手法で、マイクロソフトがやっている Daily-built のような世界なのです。旅行でも山登りでも、計画を立ててそのとおり進まなければならないということはありえません。予想外のことも起きます。あるいは、登ったことのない山なら全然予想は立たないはずなのです。とりあえずプランは作るが、臨機応変に対処していく方法のほうがいいのではないでしょうか。

５．ウォーターフォール・モデルに替わる開発モデル

ウォーターフォール・モデルに替わる開発モデルとして、今までいろいろな方法が提案されてきました。プロトタイピング・モデルもその一つですが、これですべてをプロトタイプできるわけではありません。見た目だけとか、インタフェースだけということになりますので、これもそんなにいい方法ではないと思います。また、インクリメンタル（スパイラル）・モデルでは、要求定義を決めてしまって、設計・コーディング・テストの部分だけをスパイラルでやるという、あまりよくないものもあります。

そこで、実態を調べてみようと去年の夏にアンケートを取りました。調査対象は、ソフトウェア開発プロジェクトに従事している会社員で、調査時点までの１年間にス

ケジュールの遅延、予算超過を経験したことのあるかたですが、そのとき使っていた開発モデルで多かったのはやはりウォーターフォールで、約６割を占めていました。あとはプロトタイピングが 23%、インクリメンタル／イテラティブが 17%です。ただ、この定義は、要求仕様を全部固めたあとでのスパイラルあるいはイテラティブで、要求仕様まで立ち上らないタイプであり、私の推奨している方法ではありません。

また、規模別に見ると、1000 人月の規模では８割ぐらいがウォーターフォール・モデルを使っており、規模が大きくなればなるほどこれが多くなっています。また、10 人月のように規模が小さいところでも、その半分ぐらいはウォーターフォールでやっています。

また、そのトラブルの原因を聞くと、４割が要求仕様、概要設計が 24%と、この二つで大体３分の２を占めています。また、どれぐらいの予算超過になったかを聞くと、要求仕様に原因があると 100%以上（予算の２倍）という大幅な超過が多くなっています。さらに、開発手法ごとに見ても、プロトタイピングやインクリメンタルでもトラブルは小さくなっていません。

では、どうすればソフトウェア開発がうまく進められるのでしょうか。第１には、完ぺきな仕様書を求めず、とにかく早く動くソフトを作ってそれをエンドユーザーに見せるということです。そして第２に、何をしたいのかをユーザーに書いてもらい、動くもので確認を求めていくということです。そして第３には、動くソフトウェアで進捗確認を行い、遅延も予算超過もないプロジェクト管理を行うということです。つまり、ソフトウェア開発におけるＰＤＣＡサイクルを実施するということになるでしょう。

ただ、これを中途半端にやると、大変なことになります。某大手ＩＴベンダーの人にこの方法を勧めたところ、「うちもこれをやりましたが、お客さんがいつまで了承しないので、すごい赤字になってしまいました」という返事が返ってきました。この場合は定額契約ではだめです。何か月という停止状況を契約に入れたり、コスト＋インセンティブ型などの契約を使うべきです。

また、ウォーターフォールが続いている背景としては、下請け構造が災いしているのかなという気がします。例えば設計までやって、コーディングからパートナー企業や下請け企業にお願いをするとか、外部設計だけして内部設計以降を下請け会社に任せるとか、極端なケースは要求定義だけやってあとは全部任せるとか、こういう構造がある限り、なかなか、ウォーターフォール・モデルから脱却できないのかもしれません。

６．個人の能力をより重視すること

二つめの私の提言は、個人の能力をより重視するということです。それは、優れた

人が優れたソフトウェアを作るからです。これは、エドワード・ヨードンの言葉です。ウェブ上で、「プログラマーの世界は、10%の優秀な人と、40%の普通の人と、50%の足を引っ張る人からなっています。プログラムは10%の優秀な人が作ります」と書いているプログラマーもいます。

プログラマーの能力に関する研究はアメリカで 1960 年代から行われており、1968 年に発表された論文では、「プロミラミング（デバック作業を含む）における個人の能力差は、最大で 28 対１」となっています。また、トム・デマルコとチム・リスターのレポートによると、「個人の生産性の差は５対１」、2000 年に出た論文では、100 人をできるグループとできないグループに分け、その中央値を比べると、「個人の能力比は２～７（最大で 7.3 対１）」ということになっています。また、トム・デマルコは作業時間で見た個人の能力差をグラフにしていますが、そこでは中央値と優秀な人との差が 2.5 倍になっているのです。日本のプログラマーの能力も同じように大きな差があると考えてよいと思います。

では、日本のプログラマーは、その能力を正当に評価されているのでしょうか。プログラマーの給料を見てみると、給料はきれいに年齢に比例しています。しかし、年を取るとプログラマーの生産性が上がるということは、一般的に見てありえません。確かに経験を積めば業務や業界の常識に詳しくなったりするので、間違いを起こすケースは少なくなるかもしれませんが、20 代に比べ 50 代の人が３倍の能力を持つということはありえません。また、標準偏差は年収の２割ぐらいになっていますが、能力差が２～７ぐらいあるということを考えると、これもおかしい話です。さらに（社）情報サービス産業協会のデータを見ると、年齢が一つ上がると年収が 15 万円ほど上がっていることが分かります。最近、能力主義といわれながら、いちばん能力が測りやすそうな分野で年功序列になっているということです。

また、若い人たちの間では残業手当がけっこう払われており、20 代、30 代は全報酬に占める時間外手当の割合が高くなっていますが、これには少し違和感を感じます。なぜなら、プログラムを作る能力のない人ほど時間がかかるからです。アメリカのソフトウェア業界では、基本的に残業手当など払わず、年棒制、あるいはプロジェクトベースで支払っています。日本のような状況では、できる人より残業をしなければできない人のほうが高い給料をもらうことになってしまいます。

今、ソフトウェア業界は、納期が迫ってくると徹夜、土日出勤は当たり前という状況です。恒常的な残業がある職場は、若い人たちにとって職業として魅力はありません。ですから、優秀な人が来なくなり、ソフトウェア産業全体の生産性が落ちるという悪循環に陥ってしまう危険性があります。これを好循環に変えていかなければいけないのです。

７．ユーザー側の問題と責任

日本の国際競争力の低下には、もう一つ顧客側にも問題があると私は思っています。今、2000 の市町村、47 の都道府県でＩＴ投資に使われているお金は、推計で 7000 億円といわれていますが、各市町村が同じようなシステムをそれぞれ開発しているのです。つまり、重複投資が極めて多いのです。都道府県の業務はモデルを一つ作って、それをみんなが使えば、恐らく予算は 10 分の１以下で済んでしまいます。実際、平成 15 年版の「情報通信白書」を見ると、米国と比較して、日本ではオーダーメイドで構築している部分が３倍強になっています。

日本でパッケージソフトが普及しなかった理由としては、日本企業は細部にこだわり、独自性を好むこと、エンドユーザーのリテラシーが低く、パッケージソフトが使えないこと、日本の情報システムがメーカーごとに細分化されていたために、パッケージソフト市場が少なかったこと、リスクの大きなパッケージソフト開発よりも、受託開発型のビジネスを好む企業が多かったこと等が挙げられますが、今はかなり事情が変わっています。

ただ、パッケージソフトを使うに当たっては、落とし穴に気をつけなければなりません。私は大学で学生に「キリンを冷蔵庫に入れるにはどうしたらいいか」という質問をしたことがあります。ある学生は「キリンを切り刻む」と答えましたが、正解は「冷蔵庫を開ける。キリンを入れる。冷蔵庫を閉める」です。これは、パッケージを利用するときに、切り刻まないで（大幅にカスタマイズしないで）できるだけそのまま利用した方がよいということを意味しているように思います。あるエネルギー関連企業が、米国製のＥＲＰのパッケージをカスタマイズしたところ、次のバーションアップができなくなって困ってしまったという話があります。これはまさに導入に当たって、キリンを切り刻んでしまったわけです。

二つめは、ソフトウェア開発の現場を回ると、「お客さんがひどくて」という話をよく聞かされます。「プロジェクト開始時点で仕様書が全然できていないうえに、プロジェクトをやっている最中に仕様書を変更された」、「仕様書に何が書いてあるかよく分からないので工数の見積もりさえできない」、「大事な制約条件が仕様書に書いていなくて、でき上がってから、これは業界の常識でしょうと言われてしまいました」、あるいは、「予算はこれだけしかないので、これでやってほしいと押し切られてしまう」というような話です。

ソフトウェア産業も「店が客を育て、客が店を育てる」世界だと思います。日本にも、技術力があっていいソフトウェアを作っている会社があります。また、同じ機能を持っているソフトウェアでも、できのいいものと悪いものがあります。ユーザーができのいいものを見分け、それなりのお金を払うことができれば、技術力のあるいい会社が生き残っていくことになるはずです。

かつて「ソフトウェア工場」というコンセプトがありました。しかし、物の生産とソフトウェアの生産はまったく異なります。工場での物の生産は、一回作ったものの再生産をしているのです。受託生産のソフトウェアというのは、基本的に一品生産なのです。ピート・マクブリーンも、ソフトウェア工場という発想について、「肉体労働と知的労働を混乱させていないか」と書いています。物と違って、ソフトウェアというのは目に見えませんから、作業状況の把握もできないし、管理も非常に難しいのです。この「ソフトウェア工場」というのは、「アナロジーの罠」なのです。

その辺に気づいた人が最近増えてきて、ウォーターフォールの「ソフトウェア工場」からもう少し軽量型のものへ変えよう、包括的なドキュメントではなくて動くドキュメントを重視しよう、プロセス重視よりも結果重視にしよう、バグのないソフトよりは顧客が満足するソフトを作ろうというパラダイムシフトが起きてきているようです。

また、従来の考え方ではお客さんは発注者でしたが、これからはお客さんはチームの一員であり、一緒になって開発をやっていくという姿勢でないと、本当にいいものは作れないのではないでしょうか。それから、綿密なプランを立てて、とにかく計画どおりに進めるというウェーターフォールの考え方から、もっとアジャイルに変化に対応していく。プロセスとツールに力を入れるのではなく、個人とチームワークを重視する。それから、物事は全部ドキュメントにできるという形式知重視から、暗黙知重視に切り替える。そして、事前合理性を追求するのではなく、事後合理性を追求し、結果として最善のものを作っていく。こういう方向になるべきだと思います。

実は、これはフォードの生産方式とトヨタの生産方式の違いなのです。トヨタもきちんと設計はするのですが、動き出してから改善を積み重ねていくのです。つまり、インターネット総研の藤原所長の言によると、「ＯＳＩとインターネットの違い」です。ＯＳＩとは異機種のコンピュータをつなごうというものでした。それを 1978 年から 20 年間も世界の英知を集めて延々とやったのですが、今、異機種のコンピュータはインターネットでつながっています。なぜ、インターネットが成功したかと言えば、ラフコンセンサスとランニング・コード（きちんと動くソフトウェア）を重視したからです。

８．汎用ソフトウェアにおける問題

最後に一つ付け加えます。天才を発掘して育てて、その成果をビジネスにすることにより、日本のソフトウェア産業をもっと大きくできるのではないかということです。過去、ＵＮＩＸを作った Ken Tompson ＆ Denis Richie など、世の中を大きく変えた本当に創造的なソフトウェアは、それぞれ１人から数人の天才といわれるような人が作ったものです。こういう人たちを探さなければいけません。1984 年に公開された「ア

マデウス」は凡人と天才の映画と紹介されていますが、サリエリは凡人ではないと思います。しかし、才能はあったものの、天才ではなかったのです。受託開発型のソフトウェア開発ではサリエリのような人がいっぱいいるといいのですが、パッケージはサリエリでは多分作れません。モーツアルトのような飛び抜けた才能がある人を発掘する必要があるのです。京極純一先生は『文明の作法』で、「日本の平等主義の背景には、個性と多様性の信仰ではなく、万人同型同質の信仰がある」と書いていらっしゃいますが、それぞれ能力を持った人を見つけだして、それを最大限育てていく仕組みが必要だろうと思います。

私も少し関与したのですが、1999 年の夏、経済産業省の新政策として、天才を発掘しようというプロジェクトの構想が生まれ、2000 年にＩＰＡの「未踏ソフトウェア創造事業」がスタートしました。、テーマや分野でプロジェクトを選ぶのではなく、人材に着目して、卓越したアイデア、発想力、独創力があり、末恐ろしい才能を持つ若い人たちを育てようというプロジェクトです。しかも、審査員を多くするのではなく、プロジェクト・マネージャーが責任を持って選ぶという形にしようということになりました。韓愈の『雑説（四)』に、「世に伯楽有りて、然る後に千里の馬有り。千里の馬は常に有れども、伯楽は常に有らず」という一説がありますが、天才（名馬）を見抜くような人（伯楽）がいるから、名馬が見つかるのだということです。日本にはこれだけの人口がいますから、きっとソフトウェアの天才はいます。

このプロジェクトは 2000 年にスタートし、５年めを迎えた今、順調にいろいろな成果を生みだしています。SoftEther を世に出した登君などもその１人です。あとはこの天才が生みだした成果をどうやってビジネスにするかだと思いますが、これもＩＰＡで着々と取り組みをやっているところです。

以上