はじめに

ソフトウェアの変化

これまで、ソフトウェアそのものは大きく変化してきた。その変化は、これからも継続するだろう。先ずここでは、そのソフトウェアの変化から考えてみたい。

我々は既に、ユビキタス・コンピュータの時代を迎えている。ユビキタス・コンピュータとは、我々の身の回りのそこら中にコンピュータが存在する状態をいう。

「半導体の集積度は 1 年半で 2 倍になる」というムーアの法則が、1960 年代からこれまで 50 年を超えて成立し続けてきた。この結果、最初は空調の効いたコンピュータ・ルームの奥に鎮座していたコンピュータがオフィスの中に現れ、さらに我々の机の上に載った。今はノートブック PC や iPad として我々の鞄の中に入っていたり、スマートフォンなどの形になって我々のポケットやハンドバッグの中に入っていたりする。コンピュータの小型化と高性能化が、どんどん進んできた訳である。

この半導体の集積度の向上がさらに続けば、そのうちコンピュータは我々の服やめがね、腕時計など身につけるものの中に存在したり、今我々がコンピュータとは無縁と考えている家庭用などの電気製品の中に存在したりするようになる。あるいはスーパ・マーケットやコンビニエンス・ストアで売られる各商品に、「IC タグ」と呼ばれる一種のコンピュータが付いたりするようになる。そして半導体の専門家は、「ムーアの法則は今後、さらに 10 年は成立し続けるだろう」と予言する。ユビキタス・コンピュータの時代は既に始まっているが、これからのこの本格的な到来は必至である。

使い古された言葉で恐縮だが、「コンピュータ、ソフトがなければただの箱」という川柳がある。ユビキタス・コンピュータの時代にコンピュータがまだ箱の形をしているかどうかは疑問だが、この言葉をコンピュータのハードウェアとソフトウェアの関係を簡潔に述べたものと考えたい。ユビキタス・コンピュータの時代になっても、これは変わらない。つまりユビキタス・コンピュータの時代とは、ユビキタス・ソフトウェアの時代でもある。

ソフトウェアの世界でも、既に変化が始まっている。例えば、これまで手作りのものが主流だったある種のソフトウェアに、パッケージが多く使われるようになってきた。今でもまだ人が徹夜と休日出勤を繰り返して作っているある種のソフトウェアは、そのうち人が作るのはそのソフトウェアの“要件定義書”だけになり、プログラムそのものはその要件定義書を基にコンピュータが生成するようになるだろう。

さらに、当初は人がその内容を読むことを前提にしていたインターネットが今ではセンサーとサーバを繋ぐネットワーク（Internet of Things（IoT））になり、そこで集められ、蓄積された大量のデータ（ビッグデータ）を分析して我々の生活の利便性を向上させることが普通に行われるようになっている。ここでもソフトウェアが活躍している。

このような変化は、広がりと奥行きを伴って今後一層進むだろう。しかしユビキタス・ソフトウェアの時代になって、最初から全てのソフトウェアがパッケージや自動生成の対象になるとは考えにくい。パッケージ化の進展や自動生成の対象が広がっても、対象になるコンピュータの種類と数の増加によって、人が作らなければならないソフトウェアの量は今以上に増大するだろう。

そしてその時にソフトウェアの品質がこれまでのような状態であれば、この世の中はたいへんなことになる。ユビキタス・コンピュータの時代は、ユビキタス・トラブルの時代にもなりかねない。私は今それを、懸念している。

ソフトウェア工学の勧め

今のソフトウェアの品質に問題があることは、残念ながらソフトウェアに関わりを持っている人たちの「常識」になっている。例えば、日経コンピュータ誌の記事を待つまでもなく「動かないコンピュータ」が頻発している。2002 年（平成 14 年）4 月のみずほ銀行、2005 年（平成 17 年）10 月と 12 月の東京証券取引所、2011 年（平成 23 年）3 月の再度のみずほ銀行の例を出すまでもなく、金融機関などでのシステム障害はまだ頻発している。最近は病院の情報システム化が進んだこともあって、病院でのシステム障害をマスコミが報じることも多くなってきた。我々がソフトウェアについて心配しなければならない対象が、我々の財産に関わる部分から我々の生命に関わる分野まで広がってきたことになる。しかもこれは、何も日本だけの話ではない。アメリカにもこの問題が多く存在している。このことについては、また本文の中で述べる。

しかし日本にもアメリカにも、“良い”ソフトウェアも数多く存在している。例えば、イラク戦争でまた我々が目にすることになったアメリカ軍の兵器の素晴らしさは、それについてのソフトウェアの素晴らしさの証明でもある。日本の場合このような形でソフトウェアの素晴らしさを目にすることはないけれど、しかし頻発するシステム障害とは無縁の金融機関がやはり存在している。つまりアメリカや日本のようなソフトウェアの先進国では、ソフトウェアは良いものと良くないものに二極化している。この原因は、ソフトウェア作りの二極化にある。あるところでは素晴らしいソフトウェアを作ることができ、別のところでは残念ながらそれができない。この違いを生み出しているものは、ソフトウェアを開発する組織でのソフトウェア工学への取り組みの違いによるものである。

永年のソフトウェア工学の分野での研究によって、良いソフトウェアの作り方を我々は既に知っている。その内容は改めて本文で述べるが、端的にいえば良いソフトウェアを作るための方法は、ソフトウェア工学のこれまでの成果をソフトウェア開発の現場にしっかりと適用することである。言葉でいえば簡単なことだが、これがたいへんに難しい。理由は、いくつかある。

その理由の 1 つ目は、我々の本質的な弱さから来る。例えば、早寝早起きは健康に良いと知りながら、我々はよく夜更かしをする。たばこは身体に悪いと知りながら、喫煙を止められない人が多い。学生は勉強するのが本分であるのに、学期のはじめから勉強を積み上げることができる学生も少ない。ソフトウェア工学の成果を開発現場に適用することが大切と分かっていてもなかなかこれが適用できないことには、このようなことと同じ我々の本質的な弱さから来るものがある。

2 つ目は、ソフトウェア工学の成果が、特に日本で、実際にソフトウェアを開発している人たちの間で広がっていないことがある。日本で今、本格的にソフトウェア工学を教えることができる大学の数はたいへんに少ない。さらにそのような大学でも、ソフトウェア工学を自分の専門分野として取り組む学生の数はさらに少ない。授業にも人気がない。多くの学生は映像や音響のコンピュータ処理などに大いに興味を示し、ネットワークにも関心が高い。そのこと自体は、悪いことではない。しかし大学でそのようなテーマを専攻した学生も、その多くは大学卒業後にソフトウェア開発に従事している。つまり大学で情報関係を専攻してきた学生でも、ソフトウェア工学のバックグランドを充分に持っている学生は非常に少ないというのが、日本の現実である。大学で情報関係を専攻してこなかった学生については、いうまでもない。

組込み系の世界では、元々ハードウェアの技術者だった人がソフトウェアも手がけるようになり、経験だけをベースに今ではソフトウェア開発を業としている人がいる。組み込み系については、すぐ後で論じる。

既にソフトウェアの開発現場で働いている技術者たちは、忙しすぎてこの分野の勉強に割く時間がない。さらに、ソフトウェア工学を知らなくても今のところ毎日の仕事には困らない。あるいは、情報処理技術者試験の受験のために他の領域と合わせてやむなくこの領域も少しは勉強するけれど、勉強したことを実際の毎日の仕事に生かそうとするつもりはない。この勉強は試験を受験するための勉強であって、仕事のやり方を変え、その成果を仕事に生かすための勉強ではない。日本の現役のソフトウェア技術者の現状は、こういったところだろうか。

繰り返しになるが、今のソフトウェアの問題を解決するための方法を我々は既によく知っている。それは、ソフトウェア工学のこれまでの成果をソフトウェアの開発現場にしっかりと導入し、定着させることにある。これをここでは、「ソフトウェア工学の勧め」と呼んでおく。

組込みソフトウェアについて

2015 年（平成 27 年）10 月に、ドイツの自動車メーカであるフォルクスワーゲンがディーゼル・エンジンを搭載した自動車のソフトウェアに不正に手を加え、厳しいアメリカの排気ガス規制をクリアするために試験中だけ CO_2 などの濃度を下げ、それ以外の時は汚い排気ガスを放出するようにしていたという問題が明らかになった。この事件の特徴は、ソフトウェアが全面的にこの不正に関わっていた点にある。

今や自動車は「『走る』という単一の機能を持ったコンピュータ」と呼ぶにふさわしいほど、たくさんのソフトウェアが組み込まれている。AI 時代を迎えて、これから 2 回目の東京オリンピックが開かれる 2020 年に向けて、自動車メーカ各社は自動運転車を実現するべく一層そのソフトウェアに磨きをかけようとしている。

自動車は元々メカニカルな技術の塊で、コンピュータやソフトウェアとは縁のないものだった。それがある時期からミニコンやマイコンが自動車に組み込まれてその制御で必要な機能が実現されるようになり、今や前述のように「走るコンピュータ」と呼ぶにふさわしい状態になっている。これは、自動車に限る話ではない。工業製品の多くのものに共通するテーマである。つまりソフトウェアは、ほとんど全ての産業で不可欠な要素になる。別の言い方をすれば、ほとんど全ての産業がソフトウェア産業になる。

日本はその工業製品の品質では、世界に冠たる地位を築いている。これはメカニカルの領域の先人の技術者の努力のたまものであるが、今やその一部をソフトウェアが引き継いでいる。この地位を将来にわたって一層確実なものにするために、この組込みソフトウェアの品質が重要な位置を占めることになる。

私自身は、組込みのソフトウェアに関わった経験はない。従ってこの原稿での記述は、ビジネス系の話で一貫している。しかしビジネス系も組込み系も、開発から運用に至る一部のプロセスが異なるだけで、その本質は変わらないだろう。この原稿がビジネス系だけでなく、組込み系の領域でも有益であることを期待している。

この原稿の目的

この原稿の目的は、ソフトウェア工学のこれまでの成果を広く多くの人に知って頂くことにある。この「多くの人」には、次のような人たちを想定している。

- 今実際の開発現場で、ソフトウェアの開発に従事しているソフトウェア技術者たち
- そのソフトウェア技術者を管理し、指導する立場の人たち
- ソフトウェア会社の経営に関与している人たち
- 将来ソフトウェア技術者を目指している学生諸君

- その学生たちを指導する立場にある教員の皆さん
- その他、ソフトウェア工学に興味と関心を持っている人たち

多くの人たちにソフトウェア工学の成果を広く知って頂くためには、この原稿の入手が容易で、安価でなければならない。そのためこの原稿は、インターネット上で容易にアクセスして頂ける形にして、置いておきたい。さらにこの原稿をインターネット上に置くことで、いつでも気楽にその内容を改訂／変更できるということが私自身にも可能になる。立ち上がり段階では、完成した範囲と順序で、気楽にアップロードできるという側面もある。

私のスタンス

ただこの原稿だけで、ソフトウェアの開発現場の問題解決に必要かつ充分なだけの知識と技術を伝えきることはできない。この原稿は、単なる紹介のためのものでしかない。いうならば、「広く、浅く」がこの原稿の持ち味である。したがって、この原稿で広くソフトウェア工学の成果を捉え、関連しているソフトウェア開発組織の問題解決のためにどの知識／技術、あるいはどの標準が役立つかを把握して、別の方法でそれに関連する知識と技術をより深めて、その上で開発現場の問題解決に役立てて頂きたい。これが私の望みである。そのためこの原稿には、インターネット上のページやいろんな文献など関連するものへの情報を極力充実させたい。

ソフトウェア工学全般にわたってこれまで、特に欧米の学者やコンサルタント、技術者などを中心に、すばらしい本が何冊も書かれてきた。今でもこれまで書いたものを定期的に改訂して、常に最新の情報を提供してくれる人もいる。これらの本は主に英語で書かれているが、そのうちの何冊かは日本語に翻訳されて、我々日本人も容易に読むことができるようになっている。

それにも関わらず、なぜ私がこの原稿を書こうと考えたのか。理由は、2 つある。

1 つ目は、ほとんどの著者が最新の成果を紹介することに注力し、既存の、どちらかといえば古くなった、それでもソフトウェアの開発現場で今まだ盛んに使われている技術について、ほとんど何も書いていないことについての私自身の疑問から来ている。例えば、これらの本には今 Java についてしっかりと書かれているけれど、COBOL についてはほとんど、あるいは全く書かれていない。しかしソフトウェア資産に占める COBOL の割合はまだ圧倒的に高く、大学で Java や C++を勉強してきた人たちも、ソフトウェア開発の現場に入って既存のシステムの保守を担当しなければならなくなったとき、改めて COBOL を勉強しなければならない。この本ではプログラム言語についてほとんどふれないけれど、ソフトウェア工学の技術全般にわたってそのようなことが多く見られる。それを、私なりに改めたいと思う。

2 つ目は、ソフトウェア工学の標準について書かれた本が少ないことである。ISO と IEC に、最近は IEEE も参画して共同で決めている国際標準が、ソフトウェア工学の領域をすでに広くカバーしている。これらの標準は最新のソフトウェア工学の成果から見ると、10 年以上遅れているといわれている。しかし一般の企業やソフトウェア会社から見ると、これらの標準も結構良いことをいっており、導入することで何らかの成果を上げることができるものが多い。この本では、そのような標準を多く紹介してゆきたい。

この原稿の構成

既に記したように、この原稿はインターネット上に置いておいていつでも気楽に改訂したいという私の思いから、章立てとその章をまとめる部についても、原稿の改訂が可能なように気楽に、小さめに定めることにした。ここでは章ごとの議論では話が細かくなるので、部ごとの

議論をしておきたい。

まず第 1 部では全体のイントロダクションとして、ソフトウェア工学の定義とカバーする範囲／領域、ソフトウェア危機の典型的な症状、そのソフトウェア危機を引き起こす原因などについて述べる。

本文でも記述するが、私はソフトウェア工学の基本は品質への配慮にあると考えている。したがって第 2 部では、品質保証を含めてその品質の問題を取り上げる。

第 3 部では構成管理、第 4 部ではソフトウェアとソフトウェア作りの計測（メトリクス）を取り上げる。これらはどのような考え方や方法でソフトウェアを開発するにしても、いずれもしっかりと対応しなければならない事項であると私は捉えている。

第 5 部はソフトウェアを開発する手順、つまりプロセスの問題を、第 6 部は開発方法論について述べる。

さらに第 7 部では、レビューを取り上げる。レビューはソフトウェアの品質向上を図る上で、不可欠な作業である。

そして第 8 部から第 13 部までの 6 つの部で、ビジネス系のソフトウェアを開発する上での各作業について、順次述べる。つまり第 8 部は企画、第 9 部は要件定義、第 10 部は設計、第 11 部はプログラミング、第 12 部はテスト、そして第 13 部は保守について取り上げる。

その後、第 14 部ではソフトウェアの再利用の問題を、第 15 章では数学的な記述方法でソフトウェアの仕様を提示するフォーマル・メソッドと、それを活用して高信頼性ソフトウェアを開発する方であるクリーンルーム開発を取り上げる。また第 16 部では、ソフトウェアの購入に関わる問題について述べる。

さらに第 17 部では、ソフトウェアプロセス改善について記す。ソフトウェアプロセス改善は、ソフトウェア工学の成果全体の集大成とでもいうべきものであると私は捉えており、技術についての全体の話の後にこの議論をすることにしたい。

この後の第 18 部は人と組織に関わる問題を、第 19 部でソフトウェア技術者の資格について考えてみたい。

そして第 20 部で、プロジェクト管理の問題を述べる。プロジェクト管理がソフトウェア工学の一部に位置づけされるのかということについて議論があるだろう。しかし仮にその範囲に含まれないとしても、ソフトウェアの開発プロジェクトへの固有の問題があると私は考えている。ここではそれを中心に考えてみたい。その後の第 21 部では、ツールについて述べる。

この後の第 22 章はソフトウェア工学に関わる標準の問題を、第 23 部は教育の問題を取り上げる。そして第 24 部で、ソフトウェア工学の将来について考えることにしたい。

謝辞

私がソフトウェア工学に興味を持つようになったきっかけは、1972 年（昭和 47 年）のアメリカ出張にさかのぼることができる。当時私が勤めていた日興證券（社名も当時）が突然私に 3 ヶ月間のアメリカ出張を命じて、私はニューヨークで 2 ヶ月間とロサンジェルスで 1 ヶ月間、IBM の学校に通った。

それまでの私は、単なるプログラマの 1 人でしかなかった。もちろんどうすればプログラミングの生産性を上げることができるかとか、どうすればもっとバグを少なくすることができるかなどについて考えて、自分一人で工夫は重ねていた。しかしこの出張中に IBM の学校で、もっと体系的にこの分野にアプローチする方法があることを認識し、それに興味を持った。その意味で、当時の日興證券業務部（IT 部門）の部長だった櫻井達彦氏と直属の私の上司だった櫻

井節夫氏の 2 人の櫻井氏に、まず感謝の言葉を贈りたい。

その後日興證券の第 2 次オンラインシステムの開発（1974 年（昭和 49 年）5 月～1978 年（昭和 53 年）3 月）で、この分野の私自身の技術と知識を社内で活用する局面があった。この間共にシステム開発に当たった故浜口幸夫氏、田和達夫氏、藤木登伸氏、近藤静夫氏などに、感謝したい。それぞれの人の技術領域や得意とする分野は異なったが、ソフトウェア技術者としての私にとって、充実した、ある意味で楽しい期間だった。

しかしその後の日興證券とそのシステム子会社の日興システムセンター（社名はいずれも当時）に勤めていた間は、この分野での私自身の大きな展開は無かった。ソフトウェア技術者であることをそのうちに止めなければならなくなって、この間システム子会社を含む社内でいくつかの立場を経験したけれど、基本的には「証券会社の経営と営業に、コンピュータとネットワークをどう活用すればよいのか」に答えを出すことが私の主たる仕事だったように思う。たまに海外出張があるとニューヨークあたりの大きな本屋に立ち寄って、ソフトウェア工学などの英語の本を数冊手に入れて戻って来て、往復の通勤電車の中で苦労しながら読んだ程度だった。この間ソフトウェア工学は、いわば私の趣味の対象だった。

ただこの間に、感謝しなければならない人が 4 人いる。そのうちの 2 人は当時の日本 IBM の日興証券を担当する営業だった野口直氏と、SE だった篠原正樹氏である。野口氏は定期的に出版される IBM System Journal などを、発行される都度私の机まで届け続けてくれた。篠原氏とは、会社は異なるもののもう 50 年近い付き合いとなり、少なくとも私の技術力の維持・向上に力を貸してくれた。

もう 2 人は、やはり当時富士通の日興証券担当の SE だった斑目廣哉氏（後の富士通（株）執行役員専務）と蛭田勝四郎氏（後のファイナンシャル・ネットワーク・テクノロジーズ（株）社長）である。この両名と私を含む数人のソフトウェア技術者は、日興證券の債券情報システムの開発に際して、トム・デマルコなどが「融合したチーム」[1]と呼ぶ素晴らしいチームを構成することができた。

1995 年に日興システムセンターを辞めて阪南大学の教員になったとき、私は仕事と趣味を逆転させた。つまり大学の教員としての専門領域をソフトウェア工学に定め、証券会社の経営と営業にコンピュータとネットワークを活用するというテーマを、「こんなこともできますよ」というレベルに落とした。

この大学の教員になることが決まったとき、私はそれまでソフトウェアというものを基礎からきっちりと勉強したことがないという事実に改めて気がついた。それで正規の手続きを経て、奈良先端科学技術大学院大学の情報科学研究科博士前期課程（修士課程）に入学した。ここでソフトウェア構成学講座（講座名は当時）に所属したことで、ソフトウェア工学の全容を認識することができた。このことについて、当時のこの講座の教授だった鳥居宏次氏（後の奈良先端科学技術大学院大学学長）と助教授だった松本健一氏（現在同大学情報科学研究科教授）の両氏に感謝の言葉を贈りたい。

この後、当時ミャンマーのヤンゴン・コンピュータ大学（University of Computer Studies, Yangon、UCSY）の学長だったパイティン博士（Dr. Pyke Tin）の要請を受けて、2 回にわた

[1] 「融合したチーム」とは、トム・デマルコなどが書いた「ピープルウェア」[DEM87]の中で紹介されているチームのことで、何かの弾みでこのようなチームができ、そのチームは品質面や生産性の面などで、素晴らしい製品を作ることができるとされている。私自身のソフトウェア技術者としての経歴の中で、この時ともう 1 度そのような経験を持った。

って UCSY の大学院博士課程の学生にソフトウェア工学の集中講義を行う機会を持った。この時の経験が、この本の構成を決めるに当たって役に立っている。その意味で、パイティン博士にも感謝の言葉を贈りたい。

また大学を定年退職した後、日本情報システム・ユーザー協会（JUAS）の一隅をお借りして JUAS のいくつかの研究活動に参画させていただいた。ここで、当時の JUAS の専務理事（その後副会長。今は一般社団法人アドバンスト・ビジネス創造協会副会長）である細川泰秀氏にたいへんお世話になった。さらに最近細川氏は、この原稿をつぶさにレビューして下さった。この細川泰秀氏に感謝して、この謝辞の部分を締めくくりたい。

キィワード

ユビキタス・コンピュータ、ムーアの法則、Internet of Things、ビッグデータ、組込みソフトウェア、AI 時代、自動運転車

略語

IoT：Internet of Things

参考文献

[DEM87] トム・デマルコ／ティモシー・リスター著、松原友夫／山浦恒央訳、「ピープルウェア 第 3 版 ヤル気こそプロジェクト成功の鍵」、日経 BP 社、2013 年.

前記の書籍はこの本の第 3 版であるが、この第 3 版には初版の内容がそっくりそのままの形で収録されている。

この本の原書は、以下のものである。

Tom DeMarco, Timothy Lister, “Peopleware 3nd Edition Productive Projects and Teams,” Dorset House, 2013.

2017 年（平成 29 年）1 月 2 日

玉置　彰宏

（2004 年（平成 16 年）3 月 26 日　新規作成）
（2006 年（平成 18 年）2 月 16 日　一部修正）
（2007 年（平成 19 年）5 月 19 日　一部修正）
（2007 年（平成 19 年）10 月 9 日　一部修正）
（2008 年（平成 20 年）8 月 11 日　一部修正）
（2010 年（平成 22 年）7 月 15 日　一部修正）
（2011 年（平成 23 年）10 月 15 日　一部修正）
（2013 年（平成 25 年）11 月 5 日　一部修正）
（2016 年（平成 28 年）1 月 11 日　一部修正）
（2017 年（平成 29 年）1 月 2 日　一部修正）