第 42 章　ソフトウェア技術者の特質

なぜ「ソフトウェア技術者の特質」を考えるのか

一般に、ある職業に従事している人たちは共通した特質を持っていることが多い。例えば、小学校の先生には良くも悪くも小学校の先生らしさがあり、証券のセールスには独特の押しの強さがあった。警察官にも保険の外務員にも、上級公務員にもコンピュータのセールスにも、その職業に共通した特質がある。これは、ある特質を持っている人たちがその特質を生かせるような職業を選ぶという側面があるためかもしれない。あるいは逆に、特定の職業に長く携わっていると、人々はその職業につきものの特質を無意識のうちに身につけるのかもしれない。

ソフトウェア技術者にもまた、独特の特質がある。ここでは、そのソフトウェア技術者の特質について考える。なぜ、ソフトウェア技術者の特質を考える必要があるのか。これだけコンピュータが普及し、社会のあらゆる側面で広く使われるようになると、「良い」ソフトウェアを作ることが非常に重要になる。つまりソフトウェアの善し悪しは、それを使う人たち全員に大きな影響を及ぼす。情報システムを直接使用する人が少なかったときには、影響は限定されたものだった。ここに来て影響範囲は、社会全体に広がっている。

それでは、その「良い」ソフトウェアを作るために必要なことは何か。これは多分、「良い」自動車を作るために必要なこと、「良い」野球のチームを作るために必要なことなどと、基本的には変わらない。つまり、

- 優秀な人を集め、
- 堅実な教育・訓練の仕組みを作り、
- 仕事をするための良い環境を用意し、
- 良いマネージメントがチームを統率する

こと、などにつきる。

しかしここから先は、「良い」ソフトウェアを作ることと、「良い」自動車を作ること、「良い」野球のチームを作ることとは異なる。自動車を作るために必要な特質、野球チームのメンバーが持っている特質と、ソフトウェア技術者の特質が同じではないからである。「良い」ソフトウェアを作るためにはソフトウェア技術者が持っている特質を踏まえて、前に述べた事柄を実施することが重要である。

ソフトウェア技術者の特性

それではソフトウェア技術者は一般に、どんな特質を持っているのだろうか。まずこれを一覧にして示すと、以下のようになる。

- ソフトウェア技術者は楽観的で、たいへんな自信家である。
- ソフトウェア技術者の能力差は、たいへんに大きい。
- ソフトウェア技術者は、自分が作ったプログラムを自分の「分身」のように考えている。
- ソフトウェア技術者は論理的で努力家だが、非社交的。
- ソフトウェア技術者は保守的。
- ソフトウェア技術者は狭い範囲の問題解決に興味を示し、マクロに物事を捉えることが苦手。
- ソフトウェア技術者にとって、仲間から「認められる」ことが何よりの喜び。

- ソフトウェア技術者は「明確な目標」 を実現するために努力し、多くの場合成功する。
- ソフトウェア技術者は、「社会的な権威」を認めない。

以下で、これらの項目を一つずつ、順に見てゆきたい。

ソフトウェア技術者は楽観的で、たいへんな自信家

ソフトウェア技術者はきわめて楽天的で、たいへんな自信家である。

まず楽天家の側面から、話を始めよう。

残念ながら、プログラムにバグ（ソフトウェアに内在する間違い、欠陥、など）は付き物である。バグのないプログラムは「究極の理想」であり、それを実現するために多くの人たちが日夜努力を重ねている。しかしその理想は、簡単には実現できそうにない。要件定義や設計段階の間違いなど単純なプログラミング上の間違い以外のものも含めて、不幸なことに「プログラムにバグはつきもの」といわざるを得ない。

しかし普通ソフトウェア技術者は、特にテストを実施している段階では、今存在が明らかになっているバグ以外「一切バグがない」と考えて、行動している節がある。テスト完了時期についての個人的な見解を聞いたときなどに、これが見事に表れる。ソフトウェア技術者たちはこの期待を常に裏切られ続けているが、不思議なことにソフトウェア技術者がそこから「学習」することはほとんどない。

ソフトウェア技術者が一定の経験を積むと、ソフトウェア開発のプロジェクトの管理を任されるケースが多い。このような特質を持つソフトウェア技術者がプロジェクトの見積もりを行うと、特にテスト段階で過少見積もりとなり、これが原因となって開発スケジュールの遅れや開発予算の超過を招くことになる。一般に彼らが立てるスケジュールは「バグが存在しない」ことを前提にしたスケジュールであり、バグが見つかってもそれを取り除くための余裕を時間的にも要員面でも配慮されていないことが多い。

次に自信家の側面について。

プログラミングを始めてまだ間がない人たちの中には、コンピュータが正しく稼働しない場合に、「コンピュータがおかしい。私は正しくプログラミングしている」と主張する人たちがいる。私の 55 年のソフトウェア技術者としての経験の中で、本当にハードウェアが間違っていたケースが一度だけあった。その一度を除き、全てプログラムの方に問題があった。先ほどの主張をする人たちにこの事実を告げると、「今回は、その例外の二回目に当たる」とすら言い切る。そしてこの期待も常に、見事に裏切られる。

さすがに、この間違いを二度三度繰り返す人は少ない。仮に何度も繰り返す可能性のある人はソフトウェア技術者としてふさわしくないと自分で認識するか、外部からそのような烙印を押され、ソフトウェアの仕事から離れてゆくのかもしれない。

余談だが私は、学生時代にライフル射撃のクラブに所属していた。考えようによれば、ライフル射撃ほど厳しいスポーツはない。標的は地球に対して「絶対的に」固定されている。したがってセンター（標的の中心）を撃ち抜けないことの責任は、銃や弾の管理も含めて全て射手の側にある。ソフトウェア作りも、これと同じ「絶対への挑戦」という側面を持っている。

つまり特種な場合を除き、ハードウェアはソフトウェアを作るための「前提条件」の一つであり、ソフトウェアを作るに当たって、一般にそれをあるがままに受け入れていることから始める。したがってコンピュータが正しく動作しないことについての全ての責任は、基本的にソフトウェアにある。しかし残念ながら人間は、「完全」ではない。人間が作ったソフトウェアに、

間違い（バグ）がないことを期待することはできない。我々にできることは、バグを少なくすることだけである。

前述の通り最近コンピュータは、社会のあらゆる側面で使われている。コンピュータの誤動作が人の命を奪うこともありうる[PET95]。そういう事態が起こることを事前に予見できるにもかかわらず、それでもソフトウェア技術者であろうとする人たちは、やはり楽観主義者で自信家といわざるを得ない。

ソフトウェア技術者の能力差は、たいへんに大きい

ソフトウェア技術者が全員フローチャート書いてCOBOLでプログラミングしていた頃の話だが、プログラミングについての能力差が有能なプログラマとダメなプログラマの間で10倍あったとの報告がある[1][BRO75]。これは昔プログラミングの演習を履修した学生を対象に行った実験の結果だが、実社会で人の能力にこれだけ大きな差が出るものは、たいへんに少ない。

例えば、日本で野球をする人について考えると、プロの投手が投げる球は最も速い場合160Km／h 位で、アマチュアの投手の倍少々といったところだろう。私はジョギングの延長でフルマラソンをフィニッシュする（あえて「完走する」とはいわない）ことができるが、私の所要時間は 6 時間程度で、オリンピックの金メダル級の 3 倍見当である。100 メートル競走でも、私は多分 30 秒以下で走ることができるだろうから、ここでも能力差は 3 倍見当ということになる。ゴルフのワンラウンドをいくつで回ることができるかについても、世界的なプロと駆け出しのアマチュアで 5 倍もの開きはないだろう。

もちろんアマチュアの中には、最初からそれを行う能力を持ち合わせていない人たちがいる。水泳では、「カナヅチ」という全く泳ぐことができない人がいる。自分の足だけで自分の身体を42km 運ぶことができない人は多い。この人たちを基準にとれば、能力差は「無限大」になる。ソフトウェア作りもその通りである。教育／訓練を受けたことがなく、経験も持っていない人たちは、いくら時間をかけてもプログラムを完成させることはできない。ここでの比較はその能力を持っていない人まで含めてのものではなく、すでに何らかの教育や訓練を受けて、レベルはともかくそれについて「行うことができる」人だけを対象にしたものである。

最近は、ソフトウェア技術者の専門化が進んでいる[JON96a]。つまりこれまで単純に「SE」と呼ばれていた人たちの担当していた仕事が、分化し始めている。これまでの SE はソフトウェアを作るために必要な作業を、最初から最後までを通して全て担当していた。しかし今は、アプリケーション・システムのアーキテクチャの設計をする人、データベースの論理設計を行う人、ソフトウェアの品質保証を行う人などが適宜ソフトウェア開発作業に参画し、優れた専門性を発揮して、「良い」ソフトウェアを「早く」作ることに貢献し始めている[2]。

それに伴い、以前にプログラミングの分野で存在していた 10 倍の能力差は、それぞれの専門分野で能力を高めているので、今はもっと拡大していることが考えられる。一例では、「ある特定分野では 100 倍以上の能力差がある」ともいわれ始めている。それ以上に、一般的にはソフトウェア技術者だが、特定の専門領域にはまるで「門外漢」というような技術者が増えてくる

---

[1] どの本で見たかを今は確認できないが、私はこの差について 26 倍という数字を見た記憶がある。私のソフトウェア技術者としての経験からは、26 倍の方が 10 倍より正しいように思える。

[2] ソフトウェア技術者の専門職化については、第 46 章で述べる。

ことも考えられる。このような人は、カナヅチがまるで泳げないように、専門外の分野ではいつまで経っても仕事を仕上げることが出来ない可能性もある。

これまでのようにソフトウェア技術者を単純に「SE」と呼んで、その専門性や能力差に注目することなく一律に扱うことはやめて、これからは個々の技術者が持つ専門性やそれぞれの分野での技術力などを充分に見極めて、仕事を行う際に彼らを適宜使い分けることが必要になってきている。2002 年に経済産業省が発表した IT スキル標準 [3]はこの流れに沿ったものと、高く評価することができる。

ソフトウェア技術者は、自分が作ったプログラムなどを自分の「分身」のように考えている

ソフトウェア技術者は、自分が作ったプログラムなどを「自己」を表現した一種の成果物と捉えていることが多い。場合によれば、「芸術作品」と認識していることもある。それは次のようなことから、推測することができる。

- 成果物を非常に大切にする。
- 成果物の中に、自分の足跡をしっかりと残そうとする（例えばプログラムの中に、自分の署名をコメントにして書き込む、など）。
- ケチを付けられると、たいへんに立腹する。間違いを指摘しただけでも、ケチを付けられたと考えて過剰に反応することがある。
- 成果物を共同管理の下に移そうとすると、たいへん抵抗する。

今やある種のソフトウェアは非常に大規模になり、それを一人だけで完成させることはもはや現実的でなくなった。このようなソフトウェア作りでは、チーム開発が大前提である。しかしソフトウェア技術者のこの特質は、チーム開発でたいへん大きな支障になる。そのようなチームを管理する人たちは、この特性にうち克つ方策を考えなければならない。

さらにこの特質は、品質の高いソフトウェアを作るためにもっと大きな障害になる。品質の高いソフトウェアを作るためには、システム分析の結果や設計書の内容、ソース・プログラム、テストシナリオやテストケースなど、開発作業の各段階での成果物を対象にした「レビュー」の実施が不可欠である[SAN94]。レビューは成果物をグループ討議の対象にし、内在している誤りを発見することを目的とする [4]。つまりレビューはまさに、ソフトウェア技術者のこの特質と真っ向から反するものである。この面からもソフトウェア開発プロジェクトの管理者は、ソフトウェア技術者のこの特質にうち克つ方策を考えなければならない。ワインバーグは、ソフトウェアの品質向上のために「エゴレスプログラム」の重要性を鋭く指摘している[WEI71]。

ソフトウェア技術者は論理的で努力家だが、非社交的

私のこれまでの経験では、任意に選んだ10人のうち9人まではソフトウェアを作ることができる。逆にいえば10人のうち1人だけは、ソフトウェアを作ることができない。それは、専ら「ひらめき」や「思いつき」だけで行動し、論理的に物事を考えることができない人たちである。ソフトウェア作りに「ひらめき」がたいへん大きな働きをする場合がある。特にテスト段階などで、これが顕著に出る。しかし一般に、ソフトウェア作りの基本は「論理の積み上げ」である。

世の中に、仕事の種類は多い。その中でソフトウェア技術者の道を選ぶ人は、「ソフトウェア

[3] 「IT スキル標準」については、第 47 章で述べる。
[4] レビューについては、第 18 章で述べた。

が大好き」という人たちである。その人たちに共通していえることは、「論理的に考えを積み上げることに無上の幸せを感じる」ということだろう。その人たちにとってデバッグ（プログラムの欠陥の除去）は、推理小説を読む以上の楽しみである。難しければ難しいほど、楽しみも大きい。ソフトウェア技術者は例外なく、非常に「論理的」である。

このことからソフトウェア技術者は、激しい感情の表現を嫌う。最も嫌うのは、「怒り」を表に出すことである。論理的な裏付けがある場合はまだ良い。彼らが論理的な裏付けに欠けると判断する激しい怒りは、軽蔑の対象になる。論理的な論争に負けてその結果を怒りで返す人を、彼らはそれ以降「まともな人間」として認めないだろう。

ソフトウェア技術者の間で論争が起きると、それは非常に激しいものになる。論理だけで相手にうち勝たなければならないので、「和を以って尊しとなす」文化で育った人たちからは「異常」とも見えるほどの激しい論戦が展開される。しかしそれで、彼らが感情的になることはない。論争相手との人間関係は、その立場を離れると非常に友好的である。むしろ激しい論争ができる相手ほど高く評価し、一般に尊敬の対象になる。

また彼らはソフトウェアが大好きだから、それに伴う苦労も「苦労」とは認識しない。四六時中今関わっているソフトウェアのことを考え、そのために必要なら、休日出勤や徹夜もいとわない。家族との団らんや恋人との語らいよりも、夜安らかに自分のベッドで寝ることよりも、今関わっているソフトウェアについての問題解決の方が、彼らにとってははるかに楽しい。ソフトウェア技術者は対象がソフトウェアである限り、たいへんな努力家である。

しかし一方で、ソフトウェア技術者は一般に、たいへんに「非社交的」である。論理を積み上げている過程で彼らは、思考を妨げるようなことに一切神経を使いたがらない。たいへん無口になり、最低限必要なこともできれば話さずに済ませたいと願っている。そしてソフトウェア技術者は、彼らのほとんどの時間を「論理の積み上げ」に使っている。

「ゴマをする」などということは、彼らの文化にはない。作業の成果が全てである。良い成果物を作ることができないのに、ゴマをすって管理者に取り入ろうとする同僚は軽蔑の対象である。それを受け入れる管理者も、それだけで軽蔑の対象になる。これが、彼らが「非社交的」であるとされる原因の１つだが、これがそのままソフトウェア技術者の「暗い」イメージにつながっている。

一時期ソフトウェア技術者は、欠陥人間の集まりのようにいわれたことがある。隣の席の人に用事があるときも直接口頭で伝えず、電子メールを出すことなどがその証拠とされた。これはあまりに、ソフトウェア技術者の特性を理解していない議論である。前にも書いたがソフトウェア作りに重要なことは「論理の積み上げ」であり、そのためにソフトウェア技術者は今直面しているテーマに「没頭する」ことが必要である。

没頭している状態からは、次のようなことがあると容易に覚醒してしまう。

- 電話のベルの音
- 話しかけられること
- 人の話し声
- 人の動き

いったん覚醒すると、次に「没頭」状態に移るのに最低でも 15 分程度を必要とするそうである[DEM87]。ソフトウェア技術者は自分の経験から、他人を没頭している状態から覚醒させたくないと考えている。これが、隣の席への電子メールを使った用件の伝達に結びつく。

ソフトウェア技術者を配下に抱えている管理者は、ソフトウェア技術者のこの特性に充分に留意しなければならない。最近は日本でも、ソフトウェア技術者の職場環境はかなり改善され

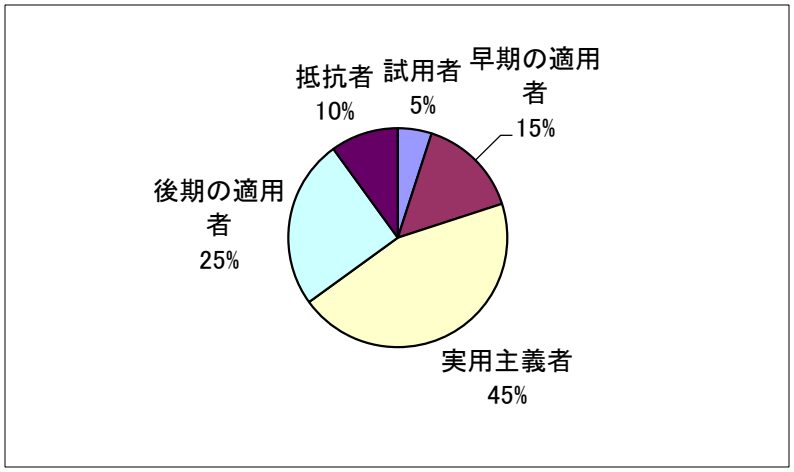
てきた。以前に比べると、格段の進歩といえる。しかしまだアメリカあたりと比較すると、充分とはいえない。今の日本で、ソフトウェア技術者全員に個室を用意することは非現実的な要求であろう。しかしそれでも、まだ多くのことができるかもしれない。できることは、率先して実施してほしい。最低でも、ソフトウェア技術者が仕事をしているところを無意味に歩き回り、「ニコポン」をしてまわることは絶対に止めるべきである。それは彼らのモラルを高めることには決してならず、かえってあなたへの反発を増やすだけに終わる。

ソフトウェア技術者は保守的

技術者は一般に、「革新的」だと捉えられている。技術者が作り出す製品が、世の中の仕組みや仕事の進め方などを変える要因になり、他の人たちに「変化」を強要するからだろうか。しかしソフトウェア技術者は、一般に「保守的」である。

ソフトウェア技術者は、最初の過程でいろいろな技術を習得しなければならない。プログラム言語がその端的な例であり、ソフトウェアの開発方法論やプログラムテストの方法、データベースの設計法など、必要ならいくつでもあげることができる。

ソフトウェア技術者は、最初の技術を習得するために熱心に取り組む。その段階で彼らは、たいへんに謙虚である。自分の技術力を高めるために、懸命の努力をする。その後修得した技術を駆使して、実際の製品作りに参画する。成果が上がるようになり、その技術に自分でも自信を持つようになる。この段階で同じ分野の 2 つ目の技術を習得させようとすると、たいへんな「反発」が待っている。自分がマスターした技術と新しく勉強する技術を事毎に比較し、「最初に修得した技術はすばらしく、新しく勉強中の技術はまるでダメ」と決めつける。その結果よほど強い意志の力や強制力が働かない場合、新しい技術の習得に失敗する技術者が出てくる。この段階を無事に通り過ぎると、3 つ目以降の技術の習得では、一般にこの反発が和らぐ。



試用者　：　何でも新しいことを試して見たがる
早期の適用者　：　新しいものを使って、成功する方法を素早く見つける
実用主義者　：　早期の適用者が成功するのを見て、採用する
後期の適用者　：　それを避けることができない場合に、いやいや従う
抵抗者　：　抵抗し続ける

図表 42-1　ソフトウェア技術者の先進性／保守性の割合（[MART91]より）

なぜ 2 つ目の技術に、ソフトウェア技術者は強い反発を示すのか。私はやはり、ソフトウェア技術者の「保守性」に起因するものと見ている。最初の技術には、保守性を発揮しようがない。まず技術を習得しなければ、ソフトウェア技術者として仕事をしてゆくことができない。最初の技術を使って仕事ができるようになると、その技術に合わせて技術者は、自分の思考パターンを作り上げる。良い仕事をする人ほど、既に良い思考パターンを作っている。新しい技術の修得は、自分が今やベテランと評価されることになった技術や、それを支えている思考パターンの「否定」から始めなければならない。

ソフトウェア技術者に限らず人間は、一般に保守的である。「今日は昨日の続き」であり、「明日は今日の続き」であることを期待している。単純な生活上の習慣についても、これがいえる。思考パターンや価値観の変更を伴うものであるときは、この抵抗はもっと大きい。新しい技術の習得はソフトウェア技術者に、その価値観や思考パターンの変更を強制するものである。これが 2 つ目の技術習得に当たっての、ソフトウェア技術者の反発の原因である。

なおソフトウェア技術者の保守性／革新性の割合について、ジェームズ・マーチン（James Martin）がその著で述べている[MART91]。これをグラフで表したものが、図表 42-1 である。ここでもソフトウェア技術者の保守的な傾向を、読みとることができる。

ソフトウェア技術者は狭い範囲の問題解決に興味を示し、マクロに物事を捉えることが苦手

多くのソフトウェア技術者が日常行っている仕事は、要件定義書を設計書の形にブレークダウンしたり、設計書を基にプログラミングしたり、そのプログラムをテストしたりといった、結果が単純で明確な作業であることが多い。つまりソフトウェア技術者は一般に、身近な問題の解決に常時取り組んでいるといえる。このためにソフトウェア技術者は、自分の「足元」の問題を発見し、その解決をはかることがたいへんに得意である。

しかし一方で、中長期的な視野や高い視点を必要とするトップダウンの問題提起、その問題の解決などは、たいへんに苦手である。ソフトウェア技術者に、企業のソフトウェア戦略の立案を期待する声が聞かれることがある。しかしソフトウェア技術者は、本来の日常の仕事を推進しているだけでは、トップダウンに仕事をするために必要な「ものの考え方」を訓練する機会を与えられていない。技術者一人ひとりの適性の問題もあるが、このような立場を期待する技術者には、別途そのための教育／訓練の機会が必要である。

ソフトウェア作りの基本的な技術の 1 つに、構造化技法（プロセス中心のアプローチ）がある [5]。データ中心アプローチやオブジェクト指向技法がすでに普及しているが、それでもまだ構造化技法でソフトウェアを作っている組織がある。その構造化技法は、技術者にトップダウンの思考を要求する。この場合技法が要求するものと現実の技術者の特質との間に、「ギャップ」が存在する。これがやはり、ソフトウェアに種々の問題を発生させている 1 つの要因になっている。

ソフトウェア技術者にとって、仲間から「認められる」ことが何よりの喜び

ソフトウェア技術者にとっての何よりの生き甲斐は、仲間内でソフトウェア技術者として高い評価を受けることである。そのためにはまず、高い技術力を身につけなければならない。その上でその技術力を駆使して、良いシステムを作らなければならない。この場合に他の手段で生活に必要な所得が得られるのであれば、ソフトウェア技術者にとって経済的な報酬は二の次

[5] 構造化技法については、第 15 章で記した。

であり、それが主目的になることはない。

ソフトウェア技術者のこの特質とオープン・ソース・ソフトウェア[6]の仕組みが結びつくと、たいへんすばらしいソフトウェアが作られることがある[RAY97] [SAN99] [TOB98]。その典型的な例の 1 つが、リーナス・トーバルズ（Linus Torvalds）氏を中心にしたソフトウェア技術者集団がインターネットを利用して作り上げた Linux と呼ばれる OS である。1998 年秋に、Linux が将来 WindowsNT の対抗相手になるのではと分析したマイクロソフトの社内文書が公開されて話題を呼んだ[7]。

これ以外にも同様の方法で良いシステムを作り上げた例がある。エリック・レイモンド（Eric Reymond）氏を中心としたグループが作ったメーリングソフトの fetchmail も、その 1 つである。ネットスケープ社は同社の製品の一部（インターネットのブラウザ）についてオープン・ソース・ソフトウェア制に踏み切り、サン・マイクロシステムズもすでに追随している。IBM も自社で作った Java 言語の開発環境をオープン・ソース・ソフトウェア化した。マイクロソフトもオープン・ソース・ソフトウェアに踏み切ることで、同社の製品の品質がたいへん向上するのではとの指摘もある。

ソフトウェア技術者は、「明確な目標」 を実現するために努力し、多くの場合に成功する

ソフトウェア技術者は、常に良いシステムを作りたいと念願している。その根本の動機が自分の個人的な評価を高めるためだけであったとしても、これは間違いのない事実である。問題は、「何をもって『良い』システムとするのか」の基準が、常に曖昧なことである。

ジェラルド・ワインバーグ（Gerald Weinberg）はこの問題について、その著書の中でやはりおもしろい実験の結果を記している[WEI71]。「早くソフトウェアを完成させる」ことを要求されたチームは、「敏速に動くプログラムを作る」ことを要求されたチームと比較すると、半分以下の作業時間しか使わなかった。しかし出来上がったプログラムは 10 倍も遅かった、とのことである。このことは、ソフトウェア技術者が充分な技術力を持っている場合、実現するべき「明確な目標」が与えられると、それを実現するために懸命な努力をし、多くの場合にその目標を達成できることを示している。

そうであるなら我々は、ソフトウェア技術者に仕事を依頼する場合、どのようなソフトウェアを必要としているかを明確に述べるべきである。これは、ソフトウェアを作る立場と使う立場の両方の人たちに、好ましい結果をもたらすことになる。

ソフトウェア技術者は、「社会的な権威」を認めない

前述の通り、ソフトウェア技術者は良いソフトウェアを作ることを生き甲斐にしている。良いソフトウェアを作って仲間内で高く評価されることを、強く念願している。それは同時に、良いソフトウェアを作ることができる技術者を彼らが高く評価し、尊敬の対象にしていることを意味している。つまり彼らが「尊敬の対象」にする人は、「コンピュータ関係のことで、やり

---

[6] プログラム（ソースコード）の著作権を主張せずに広くそれを公開し、プログラムの使用や修正を自由にできるようにしたプログラム。オープン・ソース・ソフトウェアについては、第 29 章で述べた。

[7] この文書は 1998 年のハロウインの日（10 月 31 日）頃に外部に出たので、「ハロウイン文書」と呼ばれている。その後の経緯をみると、この文書の中でマイクロソフトが心配したことはどうも現実になっているといえそうである。

たいけれど自分にはなかなかできないことを、あっさりとやってのける人」である。「自分では見つけることができない自分のプログラムの『バグ』を、いとも簡単に見つけてくれる」ことなどは、この身近な例の１つである。

彼らは、世の中の人全員がソフトウェアを作ることができるとは勿論思っていない。しかし彼らにとって、ソフトウェアを作らない人たちは彼らの世界とは関係のない世界に住んでいる人たちであって、尊敬や軽蔑の対象ではない。別のいい方をすればソフトウェア技術者は、「社長」や「部長」、あるいは「教授」といった社会的権威を、その権威を持っているということだけでは一切認めない、ということである。その社会的権威を持っていて、同時にその立場上当然持っているべきソフトウェアの能力を充分に持っていない人は、やはりそれだけで軽蔑の対象である。

それでは社会的な権威を持っているけれどソフトウェア作りのバックグランドを持っていない人は、どうすればよいのだろうか。このケースは既に多く見られると思われるが、私自身に答えはない。その立場にいるそれぞれの人たちが、上で述べたようなソフトウェア技術者の特性を考慮して、自分で考えて答えを出すべき事柄である。最低限いえることは、決して「感情的にはならない」ことぐらいであろうか。

キィワード

オープン・ソース・ソフトウェア

人名

ジェームズ・マーチン（James Martin）、リーナス・トーバルズ（Linus Torvalds）、エリック・レイモンド（Eric Reymond）、ジェラルド・ワインバーグ（Gerald Weinberg）

参考文献とリンク先

[BRO75]　F.P.ブルックス Jr. 著、山内正彌訳、「ソフトウェア開発の神話」、企画センター、昭和 52 年.

この本の内容は、以下の本にそっくり含まれている。

フレデリック・P・ブルックス，Jr. 著、滝沢徹、牧野祐子、富澤昇訳、「人月の神話：狼人間を撃つ銀の弾丸はない －原著発行 20 周年記念増訂版－」、アジソン・ウェスレイ・パブリッシャーズ・ジャパン、1996 年.

この本の原書は、以下のものである。

Frederic Phillips Brooks Jr., “The Mythical man-month：essays on software engineering,” Anniversary edition, Addison Wesley, 1995.

[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／長尾高弘訳、「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.

この本の原書は、以下のものである。

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

[JON96a] Capers Jones 著、伊土誠一、富野寿監訳、「ソフトウェアの成功と失敗」、構造計画研究所、1997 年.

この本の原書は、以下のものである。

Capers Jones, “Patterns of Software Systems Failure and Success,” International

Thomson Publishing, 1996.

[MART91] ジェームズ・マーチン著、芦沢真佐子他訳、「ラピッドアプリケーションデベロップメント Ⅰ、Ⅱ」、リックテレコム、1994年.

この本の原書は、以下のものである。

James Martin, “Rapid Application Development,” Mccmillan Publishing Co., 1991.

[PET95] アイバース・ピーターソン著、伊豆原弓訳、「殺人バグを追え」、日経BP社、1997年.

この本の原書は、以下のものである。

Ivars Peterson, “Fatal Defect,” Random House Value Publishing, 1995.

[RAY97] エリック・スティーブン・レイモンド著、山形浩生訳、「伽藍とバザール オープンソース・ソフト Linux マニュフェスト」、光芒社、平成 11 年.

この本はそっくり、次の URL からダウンロードすることができる（確認日：2017 年（平成 29 年）2 月 16 日）。

http://cruel.org/freeware/cathedral.html

またこの原稿のオリジナルは次のものだが、それが出版されたのかどうかを、私はしらない。このオリジナルも、次の URL からダウンロードすることできる（確認日：2017 年（平成 29 年）2 月 16 日）。

Eric Steven Raymond, “The Cathedral and the Bazaar,”

http://www.catb.org/%7Eesr/writings/cathedral-bazaar/

[SAN94] J. サンダース、E. カラン著、原田曄他訳、「ソフトウェア品質向上のすすめ－新しいソフトウェア開発の標準」、（株）トッパン、1996 年.

この本の原書は、以下のものである。

Joc Sanders, Eugene Curran, “Software Quality A Framework for Success in Software Development and Support,” Addison-Wesley Publishing, 1994.

[SAN99] J.サンダース、「オープン・ソースの行方を探る」、日経コンピュータ 1999.1.4 号、pp26-28、日経BP社、1999年.

[WEI71] ジェラルド・M・ワインバーグ著、木村泉他訳、「プログラミングの心理学 または、ハイテクノロジーの人間学」、技術評論社、平成6年.

この本の原書は、次のものである。

Gerald M. Weinberg, “The Psychology of Computer Programming,” Van Nostrand Reinhold Co., 1971.

（1999 年（平成 11 年）3 月　第 1 稿を作成）
（2004 年（平成 16 年）9 月 20 日　第 2 稿を作成）
（2006 年（平成 18 年）9 月 16 日　一部修正）
（2016 年（平成 28 年）9 月 11 日　一部修正）