第 3 章　ソフトウェア危機の原因

第 2 章ではソフトウェア危機の実例の 1 つとして、デンバー国際空港の手荷物取り扱いシステムの開発の経緯を見た。この開発では、システムがサービスする規模や範囲の縮小は行われたが、システムそのものは遅ればせながらも稼働を開始し、幸いなことに開発が途中で中止させられることはなかった。

第 3 章ではソフトウェア危機の原因についての最大公約数的な要因を明らかにし、第 4 章以降で、現在のソフトウェア工学の成果に基づいたその個々の要因への対応方法を議論することとしたい。

ソフトウェア危機の症状と原因

第 1 章で、今のソフトウェア危機の症状は次の 4 つの形で現れると述べた。

- 開発が、当初に立てたスケジュールから遅れる。
- 開発費用が、当初の予算を超過する。
- 開発した製品の品質が悪い。

さらに

- 開発プロジェクトが、最終の製品を作り出す前に解散させられる。

一般にソフトウェア開発の失敗の原因を特定することは、必ずしも容易ではない。それぞれのプロジェクトには、それぞれの固有の状況がある。開発がうまくゆかない原因は多岐にわたり、それらが相互に関連しあって、一般にはたいへん複雑である。しかしそれでも、失敗するプロジェクトには共通する最大公約数的な要因が存在する。

つまりある開き直った見方をすれば、ソフトウェア危機の症状を引き起こす直接の原因は、まず最初の段階では次の 3 つにまとめることができる。

- ソフトウェアの品質に関わる問題
- 作業の手戻りに伴う問題
- プロジェクトの見積りに関わる問題

以下で、この 3 つの原因について、それぞれ個別に見てみたい。

ソフトウェア危機の原因・その 1　－　ソフトウェアの品質に関わる問題

ソフトウェア危機の症状の 3 つ目は、「開発した製品の品質が悪い」というものである。この「開発されたソフトウェアの低品質」は、それ自体がソフトウェア危機の症状であると同時に、スケジュールの遅延や予算の超過、あるいはプロジェクトの解散という他の症状の原因にもなる。

後の節で、ソフトウェア開発プロジェクトの見積りの不備について述べる。プロジェクト管理が適切に行われていれば、この見積り不備からくるスケジュールの遅れは、プロジェクトの進行とともに着実に積み重なってくる。毎週 1 日ずつとか、1 ヶ月毎に 1 週間とかといった形である。フレドリック・ブルックス（Frederick P. Brooks, Jr.）は、プロジェクトの遅れは「1 週間に 1 日ずつ」発生する遅れが積み重なると述べている[BRO75]。これはこれで深刻な問題であり、軽く見てよいというものではない。開発の途中でこのような形でスケジュール遅れが発生すると、仕事の分担やスケジュールを変更することなく、担当者の「がんばり」だけでそれを埋めようとすることが多い。サービス残業が行われることもある。しかし私の経験では、これには限度がある。担当者はすでに目一杯のスケジュールを抱えており、特に遅れがある程

度以上の大きさになれば、この遅れを担当者のがんばりだけで吸収できる可能性は低い。

ソフトウェアの低品質からくるスケジュール遅れは、この見積り不備のケースとは異なり、ある時点で一気に表面化するという特徴がある。ある時点とは、テストを開始してしばらく経ったところである。一般にこのようなプロジェクトからは、要件定義から設計、プログラミング、そしてテストの初期まで「作業は順調で、進捗は予定通り」という報告が上がり続けることが多い。つまりこの間は欠陥をどんどん作り出し、それを発見して取り除くことがなされないままソフトウェアの中に蓄積し、外部にはその欠陥の存在が明らかにならないことが多い。そして、テストがある程度進んでその品質不良が一気に明るみに出て、問題が表面化することになる。まさに、フレドリック・ブルックスのいう「オオカミ人間への変身」がここで起きる[BRO87]。

そしてその後このプロジェクトは、これもブルックスがいう「コールタールの沼」に足を取られて、もがき続けることになる[BRO75]。こうなるとスケジュールと予算はなかったに等しく、問題がどんどん深刻になり、誰にも出口が見えなくなって、いつ製品が完成するかのめどが全く立たない、という状況になる。この場合この問題を解決するための最も良い方法は、これまで作り上げてきたものを全部放棄して､今度は同じ間違いをしないように充分に注意して、もう一度最初から全部やり直すことである。しかしこれを実行することは、現実にはたいへんに難しい。

欧米ではこの状態を見て、やがてユーザの中には新しいソフトウェアの入手をあきらめて、開発プロジェクトの解散を決心する人が出てくる。その場合そのプロジェクトは、最終の製品を作り出すことなく解散させられる。ソフトウェア危機の、4 つ目の症状である。ファンクションポイント[1]が 10,000（COBOL で 100 万ステートメント相当）を越えるような大規模なプロジェクトでは､65 パーセントが最終の製品を作り出せずに解散させられているという報告がある[JON96a]。アルビン・トフラー（Alvin Toffler）は、その著「パワーシフト」の中で、1980 年代のバンク・オブ・アメリカでの信託業務システム開発について述べた箇所で、このようなケースに触れている[TOF90]。

日本の場合欧米とは国民性が違うためか、プロジェクトの途中解散のケースは、これまでのところさほど多くはないように見える｡しかしいずれにせよソフトウェアの品質が悪いことは、それ自体がソフトウェア危機の症状であるだけでなく、他の要因を通してソフトウェア危機を引き起こす重要な原因でもある[2]。

ソフトウェア危機の原因・その 2　－　作業の手戻りに関わる問題

作業の手戻りは、前節で述べたソフトウェアの品質不良から発生することが多い。

例えば、要件定義の作業の結果が、ユーザを含むステークホールダの要求を明確に記述していないとする。これは要件定義の作業での品質不良の 1 つの典型的な症状であるが、この場合後続作業の設計とプログラミングの作業が適切になされたとしても､「正しくない」要件定義に基づいてなされた「正しい」設計と「正しい」プログラミングは、結果として「正しくない」プログラムを生み出す。テストの段階でそれが発見されたとすれば、要件定義から作業をやり

---

[1] ファンクションポイントとは、ソフトウェアの大きさ（規模）をそのソフトウェアの「機能」を基に見積もる方法である。ファンクションポイントについては、第 10 章で述べる。

[2] ソフトウェアの品質の問題への対応は、第 2 部の 3 つの章で取り上げる。

直さざるを得ない[3]。これが成果物の品質不良に基づく、作業の手戻りの典型的な例である。要件定義の誤りに加えて、類似の問題として「要件定義書への記述不充分」という問題もある。

上流工程を担当する技術者がその都度不明確な箇所をユーザなどに確認して、作業を進めればこの手戻りは避けることができる[4]。しかし一般に上流工程の技術者も多忙を極めており、推測や思い込みで作業を進めてしまうことがある。そしてユーザなどが必要と考えているものは、ここで推測されたものとは一般に異なっている。これが結果的に手戻りの発生に繋がる。しかし責任はテスト工程を担当する技術者にあるのではなく、その前の要件定義を担当した技術者と、その技術者のあいまいさを見逃してしまったレビュー担当者にあることはいうまでもない。たいへん恐ろしいことにこのような場合、レビューが行われていないというようなこともあり得る。

しかし作業の手戻りは、このような単純な品質不良に基づくものばかりではない。そのような場合の例の 1 つとして、仕様などの変更によって起きるものがある。デンバー国際空港の例でも見たように、仕様そのものが「連続的」とでも呼べるほど頻繁に変更されることがある。変更そのものが手戻りを発生させるのは当然だが、その徹底を欠くことで相互につじつまの合わない製品を作り出し、その修正対応で別の手戻りを発生させることがある[5]。大規模なプロジェクトではプロジェクト内部での情報の徹底を欠く場合が多々あり、それが原因となって手戻りが頻発し、問題を引き起こすことが多い。

ソフトウェア危機の原因・その 3　–　プロジェクトの見積りに関わる問題

ソフトウェア危機の症状の最初の項目の「開発スケジュールからの遅れ」と 2 番目の項目の「開発予算の超過」は、何からの遅れであり、何から超過なのだろうか。開発当初に立てたスケジュールや予算が唯一無二のものならば、それからの遅れであり、超過である。それでは、開発当初に立てたスケジュールや予算は本当に妥当なものだったのだろうか。そのスケジュールや予算は、誰が、どのようにして決めたのだろうか。

開発プロジェクトとして、客観的に見て高い生産性を上げ、妥当なコストで開発を完了した場合でも、開発当初のスケジュールや予算との比較でスケジュール遅延や予算超過があれば、「失敗」の烙印が押されることがある。全てのプロジェクトのうち 10 パーセントものプロジェクトが、非現実的な期待を満足させられないために中止させられているという報告もある。これは開発プロジェクトのメンバーにとって、たいへん不幸なことといわざるを得ない。

私の個人的な経験では、プロジェクトの管理者がその開発プロジェクトのスケジュールと予算に完全な決定権を持っていたことは、一度もない。実質的な決定はユーザや経営者が行い、プロジェクトはその決定を受け入れざるを得ない立場に置かれる。この時期ユーザや経営者は、速やかなシステムの開発が企業にとっていかに重要であるかを力説する。それは多分、正しい。だからスケジュールがトップダウンで決められることも、仕方がないことであるかもしれない。しかし私にいわせると、その前の時期、つまりシステムの企画段階やその企画の結果を決裁す

---

[3] このような事態を回避する方法として、「レビュー」がある。レビューについては、第 18 章で述べる。

[4] 要件定義を開発の当初に確定することが難しい場合、ウォータフォール型の開発手順ではなく、アジャイル型の開発手順を採用することができる。ウォータフォール型開発手順については第 13 章で、アジャイル型開発については第 14 章で、それぞれ述べる。

[5] 変更の管理は,「ソフトウェアの構成管理」と呼ばれる作業で行われる。構成管理については、第 8 章で述べる。

る段階で、ユーザや経営者が本来なすべき事柄を充分に行わず、そのしわ寄せを開発段階に入って開発部門に押しつけることが頻発している。

一般にユーザ部門は怠慢で、問題提起が遅れる上、問題の指摘にも的確性を欠き、企画作業もはかどらない。それにも関わらず時には、ユーザが「政治的」に動くこともある。この場合、「政治は禁物」である。純粋の「技術」だけでことを進めたい。デンバー国際空港の当初の開港日も、政治的に決められた。そしてその結果が、不幸な事態を招いた。

さらに経営者は一般に優柔不断で、必要な決裁をずるずると引き延ばす。このようにして、本来ならソフトウェアの開発に使えるはずの貴重な時間が空費される。しかしユーザも経営者も、この怠慢や優柔不断を外部から糾弾されることはない。

しかし悪いのは、ユーザや経営者だけではない。最初から無理と分かっているスケジュールを受け入れてしまうソフトウェア技術者の側にも問題がある。私は、これはソフトウェア技術者が一般に持っている「特性」によるものと考えている[6]。

ユーザや経営者も、押しつけたスケジュール通りにシステムが開発できると本音では思っていないことがある。この場合ユーザや経営者は、開発部門にタイトな目標を与えることによって、そうではない場合よりも早くシステムを入手できるはずだとの期待を持っている。この考えが間違いであることを、プロジェクトの見積りについての章[7]で、改めて述べる。

誰が具体的な見積り作業を行い、誰がスケジュールや予算を決めたかに関係なく、開発プロジェクトが適切なスケジュールと予算を持っていないことを、私はここでは「見積り不備」と呼んでおく。仮に見積りが的確に行われたとしても、それをプロジェクトの計画に反映できなければ結果的に見積りが正しく行われなかったのと同じになる。多発しているスケジュール遅れや予算超過はこの見積りからの遅れであり、超過であるから、この「見積り不備」は、ソフトウェア危機の 1 つの原因である。

これはプロジェクトの見積りについての章で改めて述べるが、情報システムの開発では、いくら技術者を投入してもそれ以上開発期間を短縮できない「最短の開発期間」がある。システム開発のワークロードに「人月」という単位を使うことが一般的である。技術者 1 人が 1 ヶ月間の作業を必要とする仕事の大きさが、「1 人月」である。この表現は、技術者の人数と時間が交換可能であるという誤解をもたらす。つまり技術者の数を倍にすると期間は半分になり、数を 10 倍にすると期間は 10 分の 1 になる、ということである。これがやはり間違いであることも、その章で述べる。

最短の開発期間よりも短い期間で開発するべくスケジューリングされたプロジェクトを、エドワード・ヨードン（Edward Yourdon）は「デスマーチ（死の行軍）プロジェクト」と呼んでいる[YOU98]。デスマーチとは、太平洋戦争中にフィリピンで起こった、旧日本軍が米軍捕虜に課したたいへん厳しい徒歩による移動を指す。デスマーチという言葉は、「たいへん苦しい思いをしなければならないのに、その結果が報われることがない」ことを意味している。余談だが参考文献にあげたヨードンの本は、そのようなプロジェクトに配属された技術者が、いかに傷を負わずに生き延びるかについての指針を書いたものである。こういう本が出版されたということは、ある見方をすれば、「デスマーチ」の名に値するような不合理なプロジェクトが、この世界にはたいへん多いということの証明である。

当初のスケジュールや予算策定の重要性について、すでに充分に述べた。それからも明らか

---

[6] ソフトウェア技術者の特性については、第 42 章で議論することにしたい。
[7] ソフトウェアのスケジュールについての問題は、第 50 章を参照のこと。

なように、見積りの問題はたいへん重要である。しかし仮に見積りが的確にできて、それに基づいてスケジュールや予算が決められたとしても、それで開発が順調に進むわけではない。的確なプロジェクト管理が、やはり重要である。それについて記述する部の最初の章ではプロジェクトの管理に関する問題[8]を、3 つ目の章では見積りの問題を議論したい。

現実のプロジェクトでの問題

私はここで、プロジェクト管理に関わる問題とソフトウェアの品質に関わる問題を分けて、別々に論じた。しかし現実のプロジェクトでは、これらの問題を同時に持っていることが多い。見積りは不備で、プロジェクト管理にも適切さを欠き、さらに品質上にも深刻な問題がある、というケースである。

このようなプロジェクトでは、状況は一般にたいへんに深刻であり、プロジェクト・メンバーを始め関係者一同が、たいへんな苦労をすることになる。そこでのプロジェクト管理者の責任は、たいへんに重い。

次の段階

最初に掲げた 3 つの原因のうち、プロジェクトの見積りに関わる問題の解決がもっとも容易かもしれない。しかしソフトウェアの品質の問題と作業の手戻りに関わる問題の解決は、必ずしも容易ではない。この解決のためには、ソフトウェア工学の全ての成果を活用することが必要になる。しかもその成果は技術面だけでなく、技術者の仕事への取り組み方やチームワークの問題など、人と組織に関わる領域まで拡大されることになる。

他の「工学」で、その技術的な側面に加えて、人や組織の問題がその工学の一部として議論されているものがあるのかどうかを、私はしらない。もし仮にそのようなものがないとしたら、これはソフトウェア工学の最大の特徴の 1 つであろう。

ここにある意味での、ソフトウェア開発の難しさがある。

キィワード

オオカミ人間、ソフトウェアの品質、ソフトウェア危機、デスマーチ・プロジェクト、プロジェクトの管理、見積り不備、最短の開発期間、作業の手戻り

人名

フレドリック・ブルックス（Frederick P. Brooks, Jr.）、エドワード・ヨードン（Edward Yourdon）

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

[8] プロジェクト管理のポイントについては第 50 章を、プロジェクトの見積もりについては第 52 章を、それぞれ参照のこと。

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.
[BRO87] Frederick P. Brooks, Jr., “No Silver Bullet,” IEEE Computer, 1987.
この論文は、以下の書籍に邦訳が転載されている。
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.
[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.
[TOF90] アルビン・トフラー著、徳山二郎訳、「パワーシフト（上、下)」、フジテレビ出版、1990 年.
この本の原書は、次のものである。
Alvin Toffler, “Powershift Knowledge, Wealth, and Violence at the Edge of the 21st Century,” Bantom Books, 1990.
[YOU97] E.ヨードン著、松原友夫訳、「デスマーチ：なぜソフトウェア・プロジェクトは混乱するのか」、トッパン、1998 年.
この本の原書は、以下のものである。
Edward Yourdon, “Death March The Complete Software Developer’s Guide to Surviving “Mission Impossible” Projects,” Prentice Hall, 1997.
またこの本は第 2 版が発行されている。第 2 版は、以下のものである。
エドワード・ヨードン著、松原友夫／山浦恒央訳、「デスマーチ 第 2 版：ソフトウェア開発プロジェクトはなぜ混乱するのか」、日経 BP 社、2006 年.
さらに、この本の原書は以下のものである。
Edward Yourdon, “Death March 2nd Edition,” Pearson Education, 2004.

（2003 年（平成 15 年）10 月 3 日　初稿作成）
（2017 年（平成 29 年）1 月 4 日　一部修正）