第 33 章　ソフトウェアの保守

ソフトウェアの保守という仕事

後で詳しく述べるが、ソフトウェアの保守を対象にした JIS 規格がある。JIS X 0161：2008 であり、その元の ISO 規格は ISO／IEC 14764：2006 というものである。それによるとソフトウェアの保守とは、「ソフトウェアの完全性を維持しながら、既存のソフトウェア製品に対して修正を行うこと」と定義されている[JIS08a]。ここで、「完全性を維持しながら」という文言に留意しておきたい。

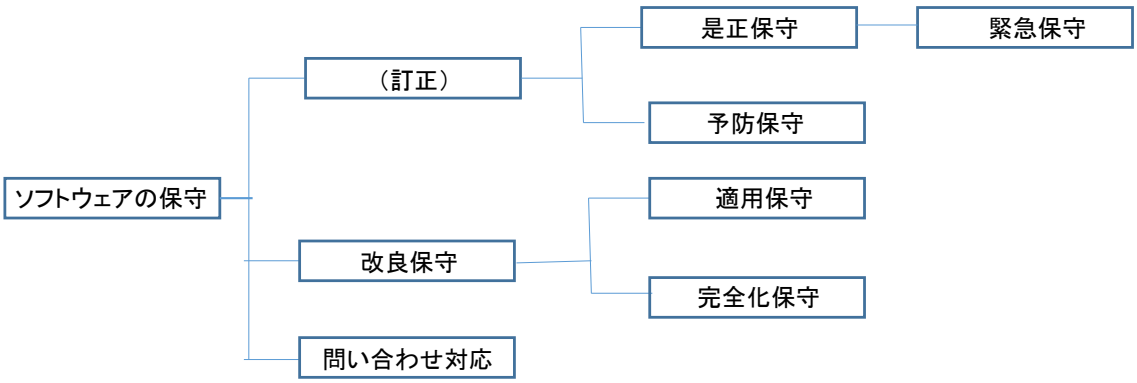
すでに記したように、ソフトウェアの保守の作業そのものはその対象のソフトウェアの新規開発が成功裏に終わったところから始まる。そしてそのソフトウェアが使われ続けている限り、継続しなければならない。ソフトウェアの新規開発は長くても 2 年もあれば普通は終結するが、保守の場合は 10 年を超えることも珍しいことではない。日本情報システム・ユーザー協会（JUAS）の調査では、開発から 21 年も使用している基幹系業務システムの例が報告されている[JUA08c][1]。

一時期よくいわれていたことであるが、広い意味でのソフトウェア開発のパワーの 70%見当が保守の作業に従事しており、新規開発のためのパワーは全体で 30%程度しかないという現実があった。その状況は、今も大きくは変わっていないと考える。

このようなことを総合すると、保守にかける経費はたいへん大きなものになり、もっと的確に、効率的に、保守という作業を行うことが必要となる。このような観点をベースに、この章ではソフトウェアの保守について議論してみたい。

保守の種類

前記 JIS 規格によると、ソフトウェアの保守には 6 つの種類がある[JIS08a]。その全体像を図表 33-1 に示す。



図表 33-1　保守の種類（[JIS08a]の図を修正）

「是正保守」とは、「ソフトウェア製品の引き渡し後に発見された問題を訂正するために行う

[1] この原稿を修正している時点（2016 年（平成 28 年）6 月 29 日)で、この調査から 10 年近くが経過している。いまでは、保守の期間はここで記したものより長くなっているかもしれない。

受け身の修正」である。この保守はないことが好ましいが、問題が起きればそれへの対応は避けることができない。このうち、特に緊急に対応する必要があるものを「緊急保守」と呼ぶ。

「予防保守」は、「引き渡し後のソフトウェア製品の潜在的な障害が運用障害になる前に発見し、是正を行うための修正」である。

「適応保守」は、「ソフトウェアの引き渡し後に、変化した、または変化している環境において、ソフトウェア製品を使用できるように保ち続けるために実施するソフトウェア製品の修正」をいう。普通ここでいう「環境」は運用環境、つまりハードウェアや基本ソフトウェアなどの変更をいう場合が多い。しかしソフトウェアの稼働環境の一部には、組織や社会環境も含まれている。つまり組織構造の変更や、そのソフトウェアの処理内容に関わる法令、制度等の変更への対応も、適応保守の一部となる。ビジネス・アプリケーションに限れば、これが圧倒的な割合を占めている。

「完全化保守」は、「引き渡し後のソフトウェア製品の潜在的な障害が、故障として表れる前に検出し、訂正するための修正」である。

この適応保守と完全性保守を、「改良保守」と呼ぶ。改良保守はソフトウェアの訂正、つまり「問題への対応」ではない。

この他に、図表 33-1 には「問い合わせ対応」がある。問い合わせへの対応は、厳密に言えば保守ではない。しかし件数で見て保守担当者の作業の 30%以上をこの問い合わせへの対応が占めているという調査結果があるので、ここに含めた[JUAS16]。

これ以外に保守担当差の作業としては、後で述べる「業務プロセス改善の提案」や「古くなった文書の作り直し」などの重要な作業がある。

ソフトウェア保守の作業

保守の作業は、ソフトウェア・ライフサイクル・プロセス全体を定めた「共通フレーム 2013」では、保守プロセスで定義されている。そのプロセスには、以下の 5 つのアクティビティがある[2][IPA13a]。

① 保守に必要な成果物の引き継ぎ
② 問題把握及び修正の分析。他の解決法の検討・提案
③ 修正の実施
④ 保守レビュー及び／または受入れ
⑤ 運用テスト及び移行の支援
⑥ 業務プロセス改善の提案

このアクティビティ間の関係を、図表 33-2 に示す。

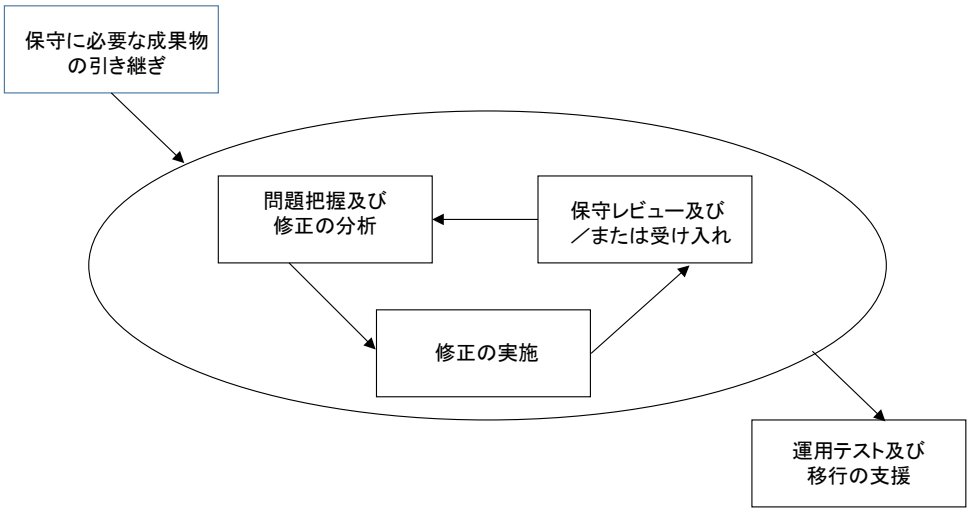
「保守に必要な成果物の引き継ぎ」は、あるソフトウェアについての保守作業を始めるに当たって、文字通り最初に 1 度だけ実施される。

個別の保守作業のきっかけになる「問題報告」や「修正依頼」が起票されると、実施が決定されたものに対して、1 件毎に「問題把握及び修正の分析」、「修正の実施」と「保守レビュー及び／または受入れ」が原則として 1 回ずつ実施される。問題報告に対してこれらの作業を実施しないということは原則ありえないが、修正依頼についてはソフトウェア構成管理に関わる

[2] JIS X 0161：2008 では、保守プロセスの中に「廃棄」のアクティビティがあった。共通フレーム 2013 では、廃棄のアクティビティは「廃棄プロセス」という別のプロセスで定義されている。

「ソフトウェア構成委員会（SCB）」が対応を拒否することがある[3]。

ここには「レビュー」という言葉はあるが、「テスト」という言葉はない。このテストは「修正の実施」のアクティビティの中で実施される。なおプログラムの修正や追加、削除の作業もこの「修正の実施」のアクティビティで行なわれるが、共通フレーム 2013 ではこのために、「ソフトウェア開発プロセス」または「ソフトウェア実装プロセス」を呼んで行うという考え方を取っている。



図表 33-2　保守プロセス（[IPA13a]より）

「運用テスト及び移行の支援」は、「ソフトウェアを新しい環境の下で稼働させるために行う」作業である。

そして最後に、「業務プロセス改善の提案」がある。これはずっと以前から経営者が IT 部門に期待し続けてきたことであるが、なかなか IT 部門はその期待に応えてこなかった。ここでこの作業は、保守担当者の 1 つの領域として定義したい。

そして必要がなくなったソフトウェアの廃棄は、新たに独立した「廃棄プロセス」で行われる。ソフトウェアの破棄で、そのソフトウェアのライフサイクルが終了する。

保守作業の特徴

それではソフトウェア保守の作業には、どんな特徴があるのだろうか。

ウォータフォール型開発で新規開発を見た場合、要件定義、設計と作業が進むにつれて徐々に作業量が増え、プログラミングでそれがピークになる。その後テストが進むにつれて徐々にそのワークロードが減少して移行作業から本番開始を迎える。ISO 14764 : 2006（JIS X 0161 : 2008）の規格について丹念に解説した「ソフトウェア保守開発」という本では、この新規開発のパターンを「ひとこぶラクダ」と表現している[SMS07]。

それに対して保守は、対応が決まった場合には対象のソフトウェアのどこにどのような形で手を入れなければならないのかを調査するところから作業が始まり、まずその部分に大きなワークロードがかかる。修正するべき範囲と修正内容が明らかになると、一般に修正作業のワー

[3] ソフトウェアの構成管理については、第 8 章で述べた。

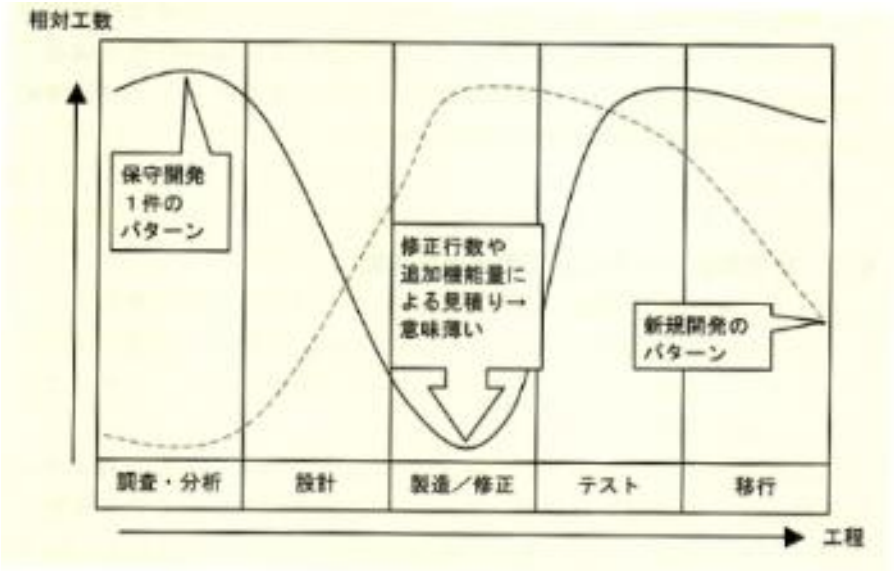
クロードはそんなに大きくはない[4]。しかしテストは、ソフトウェアに手を入れた部分だけを対象に行えばよいのではなく、全体のソフトウェアを対象に実施しなければならないので、またワークロードがかかる。

単に修正の対象になった部分が正しく修正されたかを確認するだけでなく、今回修正の対象にならなかった部分の機能が今回の修正で損傷を受けていないかの確認も必要である。この目的で行うテストを、「回帰テスト（英語では“Regression Test”）」[5]という。修正時にミスをして、以前は順調に稼働していた機能を損なうということは、テストの後半や保守作業でソフトウェアを修正するときにまま起きることである。JIS X 0161 : 2008 の定義にある「ソフトウェアの完全性を維持する」ためには、この回帰テストの実施を避けることはできない。この回帰テストまで含めると、テストのワークロードはまた大きくなる。

このため保守のワークロードは、期間の最初と最後にピークがあるという形になる。それを前述の本の著者たちは「ふたこぶラクダ」と表現している[SMS07]。

その「ひとこぶラクダ」と「ふたこぶラクダ」のモデルの図を、図表 33-3 として示す。

よほど小規模な開発でない限り、新規開発は普通チームで行われる。それに対して保守は、一人の技術者が単独で全ての作業を実施することが多い。1 件あたりの作業完了までの時間も普通は短い。つまり新規開発の場合と比較すると、保守の場合担当する技術者の作業範囲は広く、それに応じてより広く、深い技術力が必要になる。しかし IT スキル標準（ITSS）には、「保守技術者」という区分は設けられていない[6]。



図表 33-3　「ひとこぶラクダ」と「ふたこぶラクダ」のモデル（[SMS07]より）

保守の作業を円滑／的確に行うために

---

[4] 最近は、保守でソフトウェアに手を入れる作業は当初開発を担当してプログラマとは別の人が担当することが多い。このためプログラムの修正作業の負荷はそんなに小さくならず、「ふたこぶラクダ」は「ひとこぶラクダ」に変わったといわれている。

[5] 回帰テストについては、第 31 章で述べた。

[6] IT スキル標準については、第 47 章で述べる。

それではその「ふたこぶラクダ」で表されるソフトウェア保守のワークロードを、全体としてどうすれば軽減できるのだろうか。

ソフトウェアの品質を定義した JIS X 25010：2013（元の ISO 規格は ISO／IEC 25010：2011）に、システム／ソフトウェア製品品質の 1 つとして「保守性」が挙げられている[JIS13a]。つまり保守のしやすさが、ソフトウェアの良い品質についての 1 つの要件となっている。

この規格では「保守性」について、「意図した保守者によって、製品またはシステムが修正することができる有効性及び効率性の度合い」と定義されており、「修正には、環境における変更、並びに要求事項及び機能仕様における変更に対するソフトウェアの訂正、改良または適応を含むことができる」との補足説明がある[JIS13a]。

そして、その副特性として以下の 5 つのものが挙げられている[JIS13a]。

- モジュール性： 1 つの構成要素に対する変更が他の構成要素に与える影響が最小になるように、システムまたはコンピュータプログラムが別々の構成要素から構成されている度合い
- 再利用性： 1 つ以上のシステムに、または他の資産作りに、資産を使用することができる度合い[7]
- 解析性：製品もしくはシステムの 1 つ以上の部分への意図した変更が製品もしくはシステムに与える影響を総合評価すること、欠陥もしくは故障の原因を診断すること、または修正しなければならない部分を識別することが可能であることについての有効性及び効率性の度合い
- 修正性：欠陥の取り込みも既存の製品品質の低下もなく、有効的に、かつ、効率的に製品またはシステムを修正することができる度合い
- 試験性：システム、製品または構成要素について試験基準を確立することができ、その基準が満たされているかどうかを決定するために試験を実行することができる有効性及び効率性の度合い

国際規格らしい難しい言葉が使われているが、ここではまずプログラムが適切なモジュールの組み合わせで作成されていて、ソフトウェアの修正を的確に行うために、内容の理解が容易にでき（解析性）、変更が簡単で、変更時に間違いが起きにくく(修正性)、テストによる確認が容易で（試験性）あることが必要、と述べている。全くそのとおりである。

それでは、どうすればこのようなソフトウェアを手に入れることができるのだろうか。そのためには、ソフトウェアの要件定義を提示する段階で、「非機能要求」の一部にこの「保守性」に関わる要求を明記するのが良い。

しかしここでの要求が「保守のしやすいソフトウェアを開発してください」というものでは、全く意味がない。要求は具体的で、しかもレビューやテストの段階で「検証」できるものでなければならない。例えばソース・プログラムに適切にコメントが入っていることがソフトウェアを理解する上で役立つということであるなら、「ソース・プログラムにコメントを記入すること」を要求しなければならない。あまりコメントの数が多いと結果としてプログラムが読みにくくなるとか、上のような要求の仕方ではたくさんコメントを入れる人とほとんど入れない人が出て、不統一になるということであれば、コメントの割合を例えば全体の 10%程度と決めて、「ソース・プログラムへのコメントの挿入率を、10%を目処にする」とすればよい。

ソース･プログラムへのコメントの挿入は「解析性」に関わるものの 1 つだが、解析性につい

[7] ソフトウェアの再利用の問題については、第 35 章で述べる。

てさらに他の要求があるかもしれない。それ以外の修正性、試験性についても、必要ならこのような指標を作って、具体的に要求する必要がある。そして先でも述べたように、要求したものが実現されているかを、レビューやテストで必ず確認しなければならない。これは要件定義書上で「機能」として要求したものがソフトウェアに的確に組み込まれているかどうかを、レビューやテストで確認することと全く同じである。

ISO は「技術報告書（英語では “Technical Report”、省略して単に “TR” と記述されることが多い）」の 1 つとして、「保守性」を含むソフトウェアの品質に関連して、多くの具体的な指標を明確にしている[ISO03a] [ISO03b]。また日本情報システム・ユーザー協会（JUAS）も非機能要求に関わる指標をまとめた文献を出しており、それには「保守性」についての指標も含まれている[JUAS08a]。具体的にどんなものを要求すればよいのかについては、これらの文献を参考にすることを勧める。

さらに要件定義書から設計書を経てプログラムまで、双方向にトレースできる機能を持つことが望まれる。具体的には、要件定義書のある部分に記述された「機能」が設計のどの部分に記述され、さらにそれがどのモジュール／クラスで実現されているのかを追跡でき、逆にプログラムから設計書、要件定義書へと遡っても行けるという機能である。この仕組みは「トレーサビリティ」と呼ばれていて、最近のソフトウェア工学上の重要な成果の 1 つである。これが的確にできていれば、保守作業が大いに効率化されるだろう。

冒頭にも書いたように、ソフトウェアの保守作業そのものは新規開発の決着が付いた後で行われる作業である。しかし保守の対象になるソフトウェアが実際に開発される前からそれに関する対応を始め、開発中も着実に準備を重ねておかないと、実際の保守の段階で円滑に、かつ的確に対応することできない。

ソフトウェア保守の計画

この原稿で何度も引き合いに出している JIS X 0161：2008 では、ソフトウェアの保守のための戦略の立案と計画の策定は、やはりそのソフトウェアの開発中に、しかもなるべく早い段階で行わなければならないとしている[JIS08a]。

計画策定の過程で最も難しいのは、どれくらいの人数とどういうスキルの人を特定のソフトウェアの保守のために用意しなければならないかということだろう。ソフトウェアの品質が悪い場合は緊急保守が多く、そのためのワークロードを特別に見積もらなければならない。しかし品質が一定レベル以上であれば緊急保守はそれほど多くはなく、ビジネス・アプリケーションではむしろ法令や制度の変更による適応保守が多い。

したがって開発過程の終盤あたりで残存欠陥数の把握を行ってソフトウェアの品質に目処を付け[8]、適応保守の方から必要な人数とスキルの算定を行うのがよい。

誰がソフトウェアの保守を担当するのがよいか

私がまだ現役だった頃、ソフトウェアの保守はそのソフトウェアの新規開発に参加した技術者の中から何名かが引き続き担当するのが普通だった。しかもその頃は、新規開発と保守はチームを分けず、同じチームが両方を担当するのが普通だった。

しかし、1 つのチームが新規開発と保守の両方を担当するのは良くない。保守作業は常に、新規開発よりも優先順位が高い。したがってこの両方を同じチームが担当していると、新規開

[8] 残存欠陥数の把握方法については、第 18 章と第 32 章ですでに述べた。

発が保守作業の影響を受けてスケジュール遅れをきたすことが多い。これが原因して、品質面に影響が出ることもある。これらはいずれも、新規開発では避けなければならないことである。

すでに述べたように、新規開発と比較して保守にはより幅の広い業務知識と高い技術力が必要になる。チームを分けた場合、このような高い知識と技術力を持った人は保守を担当することになり、新規開発のチームには新人など知識や技術力の劣後している人たちが多く担当することになる。これは新規開発を担当するチームのリーダには負担になるが、組織としては技術者を育成するための格好の機会を提供することになる。

保守は、新規開発を担当した人以外の人が担当することもできる。同じ人に担当させるのは、新規開発時にソフトウェアの開発と並行して行わなければならない文書類の整備が充分ではない場合など、文書作成のワークロードや引き継ぎの手間を割愛しようとする、どちらかといえば好ましくない動機から生じることが多い。仮に文書類が的確に作成され、それに基づいて引き継ぎもしっかりと行うことができれば、新規開発を担当したメンバーが引き続き保守を担当しなければならないということはなくなる。

仮に新規開発とは別のメンバーでも保守が担当できるということなら、この保守の作業をアウト・ソースすることも可能になる。新規開発をアウト・ソースしたとしても、その受け手とは別の企業でも保守を担当できることになり、選択の幅が広がる。つまり保守のための準備を怠りなく的確に行っておけば、保守の作業は新規開発を担当した技術者以外の人でも担当することができ、選択の幅が広がる。

今ソフトウェア開発のアウト・ソースといえば、海外へのアウト・ソースを意味することがある。しかし保守を海外に持って行くことには、まだ問題が多い。海外にアウト・ソースするためには、保守用の文書を英語で作成しなければならないかもしれない。またさほど多くないとはいえ、緊急保守の場合の対応に不安がある。さらにソフトウェアには法令や制度など明文化されたもの以外の、例えば「文化」や「ものの考え方」に関連するような日本人なら普遍的に持っているある意味での「常識」ともいうべきものも、盛り込まれているかもしれない。海外に保守作業をアウト・ソースする場合、これらへの対応はたいへんに難しい。これらの問題を解決できるまでは、保守をアウト・ソースするにしても日本国内が適切なアウト・ソース先であるといわざるを得ない。

「ソース・プログラム以外、信用できるドキュメントがない」

これから開発するソフトウェアについては、上で述べたような方策を講じて保守のし易いソフトウェアを開発することができる。しかしすでに開発後に長い年月が経っていて、その間緊急の対応を何度も繰り返してきたことなどもあって、ソース・プログラム以外の設計書や要件定義書などの文書が、仮に存在していても現状から全く乖離してしまっていて使い物にならない、というような状況もあり得る。

このようなソフトウェアの保守は、たいへんに難しい。どこかのタイミングで作り直しをして保守のし易いソフトウェアに変えることが、最終的な回答である。私が現役時代に保守作業を含むソフトウェアの開発部門全体の管理を担当していた頃、そのようなソフトウェアのいくつかを、関連する部分の保守の機会を見つけて順次新しく書き直すことを担当者と進めていた時期があった。

もし仮に担当者に時間的な余裕があるなら、ソフトウェアを解読して少しずつでも良いからドキュメントを整備してゆくことが、時間はかかるがやはり 1 つの回答である。

さらに別の方法として、市販されているツールを使うという方法もある。次章でリバース・エンジニアリングについて述べるが、それについてのあるツールではその企業が持っている全てのソフトウェアのソース・プログラムと JCL やデータベース定義などを読み込んで分析し、「ある表のある部分に表示されている情報は、どこで入力されたどのデータと別のところで入力されたどのデータを基に、どのプログラムでどのような加工を行なって、どのデータベース（ファイル）のどの部分にその情報が格納されている」、というような詳細な情報を取り出すことができる。プログラム上の記述だけの情報なので、コーディング・ステップが何を意味しているのかをその情報を基に判断しなければならないけれど、前述のような状態での 1 つの回答になりうる。

当然のことながら新しいソフトウェアを保守するときには、JIS X 0160：1996 にも記述されているように、ソース・プログラムの修正と合わせて関連するドキュメント類の修正も的確に行わなければならない。「これを行わない保守担当者は犯罪者である」といってもいい過ぎではないと、私は考えている。

ソフトウェアの構成管理との関係

ソフトウェアの構成管理の目的は、ソフトウェアについての変更を管理することである。保守はソフトウェアを変更する作業であるから、ソフトウェアの保守の作業はソフトウェアの構成管理の機能と緊密な連携がなければならない。具体的には、ソフトウェアの保守の作業で変更されるものは全てソフトウェアの構成管理での管理の対象になっていなければならず、その変更の作業は全て構成管理の手順に則って行われなければならない[9]。

ソフトウェアの保守とソフトウェアの構成管理との間の関係は、構成管理委員会（SCB）が行う「変更を実施するか否かの決定」だけではない。

保守についての思い出

私も現役時代は長い間ソフトウェア技術者として勤務していたので、保守の作業も長い期間担当していた。個人的な話で恐縮だが、その間での保守についての思い出などを少し書いてみたい。この思い出には、楽しい思い出とつらい思い出、そのどちらでもないものがある。

まず、楽しい思い出から。私が勤めていた会社が第 2 次オンラインシステムを開発したとき、私がアプリケーション・プログラムの開発チームのリーダをつとめた。その時同僚と語らって、いくつかの領域での共通モジュールの作成と、全てのシステムでのそれらの共通モジュールの使用を取り決めた。その共通モジュールの 1 つに、和暦年号（昭和／平成、など）と西暦年号を相互に変換する機能があった。

不謹慎な話で恐縮だが、昭和天皇の健康状態がすぐれないという話がマスコミから報じられた後、私の同僚の 1 人がこのルール通りにプログラムが作成されているかどうかを全てのシステムを対象に調査し、勝手に年号の変換を行っているプログラムを見つけて手直しして、そのモジュールを使うように変更した。後はもう想像していただけるだろうが、昭和天皇が亡くなられた時、そのモジュール 1 つだけに手を入れて、後は週末にリンク・エディットの大作業を行って、和暦年号の変更に対応したことがある。もう 30 年近く前の話である。

つらい話に移る。私はある時期、ある範囲のアプリケーションを担当するプログラマ集団のリーダを勤めていた。そのチームの責任範囲にあるプログラムが夜間に障害を起こして翌日の

9 ソフトウェアの構成管理については、第 8 章ですでに記した。

オンラインの立ち上げに支障が出る場合、時間にかかわらず私の自宅にオペレータから電話を入れてもらうことにしていた。電話があるとオペレータから状況を聞いて担当者を選び、その担当者に電話して、緊急の出勤とソフトウェアへの対応を要請するのも私の仕事の一部だった。しかし時々、担当者が捕まらないことがあった。今のような、携帯電話というものがまだなかった時代の話である。その時はやむなく、私自身がタクシーで緊急出勤をして対応しなければならなかった。

中間の話は、こういうものである。私が管理者を務めていた組織の担当のソフトウェアに、所得税や法人税などの税制に関わるものが多くあった。その頃は毎年年が明けて通常国会が開かれると、私は国会での税制関係の議論を注意深くフォローしていた。予算編成に関連して、たいてい毎年税制のどこかが変更されて、それに伴いソフトウェアの一部に手を入れなければならないからである。税制改定についての変更依頼は別の部署から正式に届けられる仕組みになっていたが、どの程度の作業量になりそうか、誰に担当してもらうのが適切かなどの判断に、このフォローから得られる情報が欠かせなかった。

「保守」と聞くと、「暗い」イメージがあるという。しかし私には、そんなイメージはない。先に述べた「つらい」話はやはりつらい思い出だが、「仕事とは所詮つらいもの」という割り切りの範囲内のものでしかなかった。これは「緊急保守」の範疇に入るものだが、私の理解では緊急保守の割合はそう多くはない。むしろ法律や制度が変わって、そちらの関係からの「適応保守」が圧倒的に多かったという印象が残っている。

私自身が作成したプログラムに原因があって緊急保守が行われたという事態は、幸いなことに私がまだ駆け出しのプログラマの頃に 1 回経験しただけで終わった。「私のプログラムには、絶対にバグを埋め込まない」というのが私のプログラマ時代の最大の目標だったが、幸いにもそれが効果をもたらしたものと考えている。

保守に関わる規格

冒頭にも記したように、ソフトウェアプロセス全体をカバーする JIS X 0160：1996（元の ISO 規格は ISO／IEC 12207：1995）の主ライフサイクルプロセスに、保守プロセスというものがある[JIS96]。その保守プロセスを詳細化したものが JIS X 0161：2008（元の ISO 規格は ISO／IEC 14764：2006）であり、この 2 つの規格の間には親から子に向けての一貫性が保持されている[JIS08a]。具体的には、JIS X 0160：1996 でしか定義されていない（JIS X 0161：2008 には記述されていない）文書化プロセスや品質保証プロセス、妥当性確認プロセス、検証プロセスなどを、JIS X 0161：2008 で記述されている保守プロセスが使用することがある、などがその例である。ISO／IEC 12207：1995 で定義されているプロセスのあるものが、それをさらに詳細化されて別の規格になったというのは、これが始めてのケースである。しかしそれ以降、このようなケースが増えてきている。

以前 ISO には ISO／IEC 14764：1999（それを JIS 化したものは JIS X 0161：2002）という、今のものの 1 つ前のソフトウェア保守について規格があった。一方 IEEE には、IEEE Std 1219-1998 という、やはりソフトウェア保守についての規格があった。ISO がこの 14764 を改訂しようとしているときに、IEEE から働きかけがあって IEEE Std 1219 の内容を ISO の 14764 と合流させた[JIS08a]。これで実質 ISO の 14764 の 2006 年版は、ISO と IEC、そして IEEE の 3 つの組織合同の規格となった。

これに伴い IEEE は、1219 というこれまでソフトウェア保守に使っていた規格番号を改め

て、今ではこの規格を ISO/IEC 14764 IEEE Std 14764-2006 と呼おんでいる[ISO06a]。こういう動きも今回が始めてのケースであり、今後このようなケース増えてくるのかもしれない。

キィワード

ソフトウェアの保守、是正保守、緊急保守、予防保守、適応保守、完全化保守、改良保守、ソフトウェア・ライフサイクル・プロセス、回帰テスト、Regression Test、ふたこぶラクダ、ひとこぶラクダ、保守性、モジュール性、再利用性、解析性、修正性、試験性、非機能要求、トレーサビリティ、リバース・エンジニアリング、JCL、ソフトウェアの構成管理

規格

JIS X 0161：2008、ISO／IEC 14764：2006、共通フレーム 2013、JIS X 25010：2013、ISO／IEC 25010：2011

参考文献とリンク先

[IPA13a] 情報処理推進機構ソフトウェア・エンジニアリング・センター編、「共通フレーム 2013 経営者、業務部門が参画するシステム開発及び取引のために ソフトウェアライフサイクルプロセス 共通フレーム 2013」、オーム社、平成 25 年.

[ISO06a] “Software Engineering – Software Life Cycle Process - Maintenance ISO/IEC 14764 IEEE Std 14764-2006,” ISO/IEC, 2006-09-01.

[JIS96] 日本工業標準調査会審議、「ソフトウェアライフサイクルプロセス JIS X 0160-1996（ISO/IEC 12207：1995）」、日本規格協会、平成 8 年.

[JIS03a] 日本工業標準調査会審議、「JIS ソフトウェア製品の品質－第 1 部：品質モデル JIS X0129-1：2003 （ISO/IEC 9126-1：2001）」、日本規格協会、平成 15 年.

[JIS08a] 日本工業標準調査会審議、「ソフトウェア技術－ソフトウェアライフサイクルプロセス－保守 JIS X 0161：2008（ISO／IEC 14764：2006）」、日本規格協会、平成 20 年.

[JIS13a] 日本工業標準調査会審議、「JIS システム及びソフトウェア製品の品質要求及び評価－システム及びソフトウェア品質モデル JIS X 25010 : 2013 （ISO／IEC 25010 : 2011）」、日本規格協会、平成 25 年.

[JUAS08a] 日本情報システム・ユーザー協会他著、「検収フェーズのモデル取引整備報告書 UVC 研究プロジェクトⅡ報告書 非機能要求仕様定義ガイドライン」、日本情報システム・ユーザー協会、平成 20 年 6 月.

[JUAS16] 日本情報システム・ユーザー協会、「ユーザー企業ソフトウェアメトリクス調査 2016 ソフトウェア開発・保守・運用の評価指標」、日本情報システム・ユーザー協会、2016 年.

[SMS07] ソフトウェア・メインテナンス研究会編、増井和也他著、「～ISO14764 による～ソフトウェア保守開発」、ソフト・リサーチ・センター、2007 年.

（2008 年（平成 20 年）6 月 3 日　初版作成）
（2016 年（平成 28 年）6 月 29 日　一部修正）
（2017 年（平成 29 年）1 月 27 日　一部修正）