ソフトウェア・ライフサイクル会計
-ソフトウェア会計の体系的研究-

明治大学大学院経営学研究科

2015 年度

博士学位請求論文

ソフトウェア・ライフサイクル会計

――ソフトウェア会計の体系的研究――

An Accounting for Software Life Cycle

――A systematic study of software accounts――

学位請求者　経営学専攻

長田　芙悠子

凡　例

１．本論文ではソフトウェア用語が頻出するが、多数回使用するもの又は筆者にとって重要と思われる用語は付録の「ソフトウェア用語集」で説明することにした。参照されたい。「ソフトウェア用語集」に収録していない用語に関しては、初出時に脚注で説明することにする。

２．外国人名は、慣用的なカタカナ表記とし、初出時に原語表記を行なうこととする。

３．引用文献（参考文献）に関しては、原則として、本文では著者名・刊行年のみ表記し（例：櫻井通晴編著(1993)）、脚注で該当ページを付加して出典表記とし、併せて「参考文献」の参照を指示することにする。書誌的な詳細は「参考文献」に記載する。

目　次

細　目　次

図表一覧

序論

１．ソフトウェアを巡る昨今の情勢

２．ソフトウェア・ライフサイクル

３．ソフトウェアのプロダクトとプロセス

４．ソフトウェア開発失敗の基底因

５．ソフトウェア会計研究の在り様

序論

本論文は、ソフトウェアのライフサイクル（企画、開発、運用、保守、廃棄）に沿って、どのような会計処理を行なうことが、経済的実態に適合的であり、且つ利害関係者への情報開示として目的適合的であるかを体系的に構成したものである。現行の会計基準（日本、アメリカ、国際会計基準）は、いずれも開発を中心に関連事象を多少取り扱っている程度で、ライフサイクルの全域を取り扱っておらず、規定にない事象は会計慣行に委ねられてしまっている。また、いずれも基準設定以降のソフトウェア分野の市場動向・技術動向（オープンソース、クラウド・コンピューテング等々）に対応した改訂を行なっていないので、任意的な会計処理に任されており、比較可能性が多少毀損しつつある。それに対し、近年のソフトウェア分野の動向までを包含し、且つそれらを適切にライフサイクルに位置付け、会計処理の整合的な体系として提示することを、本論文は企図している。

修士論文は、ソフトウェア会計基準を主題とし、日本基準を中心に、アメリカ基準並びに国際基準をも取り上げて、基準の構成に即して詳細な批判的研究を行なった。それ以降も一貫してソフトウェア会計の研究を継続してきたが、修士論文とは異なり、既存の会計基準に即したものではなく、上記の通り、自らが構想するソフトウェア会計の体系に則ったものである。凡そ半分程度は審査を経て発表することができたが、残り半分程度は未発表であるが、漸く纏まったものとなり、体系を構成できるようになるまでに到ったので、この度学位請求論文として集大成し、提出する次第である。

１．ソフトウェアを巡る昨今の情勢

個別論文でも行なってきたことであるが、冒頭では、ソフトウェアを巡る昨今の情勢において、筆者の特に注目した事象を幾つか取り上げることにしたい。

１つは、2015 年 6 月に表面化した日本年金機構の 101 万人分、125 万件の年金情報流出という事故である[1]。2007 年には「年金記録問題」が大きく取沙汰されたが、その後暫くは鳴りを潜めていた感があったが、今回は情報流出という事故が起きた。このような事故がなかろうと、少子高齢化社会では老齢年金は言わば問題であり続けることであるが、更に情報セキュリティ事故が加重した事態と言える。そして、日本年金機構に限らず、情報セキュリティ対策は間断なく、恒久的に取り組み続けていかなければならない必需的な課題となってしまったのであり、取り分け社会的に注視される組織等が格好の標的となるのである[2]。

もう１つは、特許庁システムの比較的最近の動向である。まずは、「特許庁は 2006 年、基幹系シ

[1] 『日経コンピュータ』2015 年 7 月 9 日号 pp. 18-31 等

[2] 情報セキュリティに関しては、内部統制やシステム監査との関連において、今後主題的に取り組んでいくつもりの課題である、と取り敢えず今は記しておくに留めたい。

ステムの全面刷新に向けて入札を実施。システムの設計・開発業務を東芝ソリューションが、管理支援業務をアクセンチュアがそれぞれ落札した。だが開発は難航し、5年後の2012年1月に開発中止になった」[3]ことに始まる。ところが、「東芝ソリューションとアクセンチュアが、2012年に開発を中止した特許庁システムの開発費に利子を加えた約56億円を、同庁に返納していたことが分かった。特許庁によれば、2013年8月に合意が成立、同年9月に返納金として両社から約56億円が支払われた」[4]、という異例の対応をしていたことが判明したのである（民間企業が顧客であったならば、後続の事象のように、裁判にでもならない限り、「返納」などまず行なわない)。そして、比較的最近の動きである。「特許庁の基幹系システム刷新プロジェクトが頓挫してから３年」[5]。「特許庁は2015年3月、基幹系システム刷新計画の全容を公開した。約700億円を投じ、8年がかりでシステムを順次更新する。現システムは運用・保守に年間250億円を費やしており、システムの刷新で費用の3割減を目指すほか、審査業務のスピードや質、利用者の利便性を高める。入札前にアーキテクチャー案を公開し、専門家の意見を募る。／特許庁はかつて、2006年にシステム刷新を始めた。だが、開発は難航。2012年1月に中止の憂き目にあった。ベンダー依存からの脱却を目指しつつ、組織として開発を主導する体制になっていなかったためだ。／特許庁は、この失敗に正面から向き合った。過去の反省を基に四つの基本方針を設定し、計画をゼロから組み直した。いずれも奇をてらったものではなく、「強いユーザー」を目指す上で正攻法といえるものだ。／「前回は組織としての本気度が足りなかった。今回は全庁を挙げてシステム刷新に挑む」。特許庁ナンバー2である特許技監 特許庁CIOの木原美武氏は力を込める。／同庁の改革は、ベンダー依存の体質から脱却しようと考える多くの企業にとって参考になる」[6]、というのが大凡のアウトラインである。やや長い引用を行なったが、記事は全般的に特許庁の動きを肯定的あるいは「好意的」に取り上げており、筆者とは受け止め方が大きく異なる。

第１に、外庁とは言え、特許庁はＩＴの監督官庁である経済産業省の一翼である。ＩＴに関して指導的な模範となって然るべき立場にあるのであり、それが大失態を仕出かしたのであり、そして漸くごく当たり前の取り組みをするようになったというだけのことである。しかも血税を使ってのことである。「参考になる」などと鷹揚なことを言うべき状況ではなかろう。

第２に、「8年がかりでシステムを順次更新する」とのことであるが、前回の「一括更新」の計画の失敗を回避するためのようだが、随分と悠長な計画である。しかも、「約700億円を投じ」、運用・保守の「費用の3割減を目指す」とのことだが、年間250億円の3割減つまり約75億円を減らすことを9年強変わらず続けて漸く開発費を回収できるという算段である。「利便性」等は除き、あくまでランニング・コスト削減に限ってのことだが、この効果が確実に現れるのは「順次更新」を続け

---

[3] ITPROの記事「2012年の特許庁システム開発中止、開発費全額返納のなぜ」（http://itpro.nikkeibp.co.jp/atcl/column/14/346926/073100025/、2015/08/04,17:54アクセス）
[4] 同上
[5] 『日経コンピュータ』2015年4月30日号p.78
[6] 同誌p.79-80

る８年が経過して以降のことであろう。いずれにしても、足掛け 17 年も先に効果が漸く判明することになるわけだが、その間に新たな事態・事象が起きないという保証はあるのだろうか。大いに疑問である。

第３に、「特許庁は、システムアーキテクチャーの面でも開発の難易度を引き下げた。／かつてプロジェクトが失敗した要因の一つに、採用したアーキテクチャーの難易度が高かったことがあった。「ＸＭＬで全ての業務を管理する」という理想を掲げる一方[7]、実績ある開発手法も、開発ツールもなかった。現場の業務プロセスも、このアーキテクチャーに合わせて書き換える必要があった」[8]のに対して、「今回、既に開発ツールや開発の方法論が存在し、特定ベンダーに偏らないアーキテクチャーの採用にこだわった」[9]とのことであるが、しかし採用しようとしているアーキテクチャーには同様の疑問を呈せざるをえない。「2015 年 3 月までに固まったのが、ＳＯＡ（サービス指向アーキテクチャ）に基づき、ＢＰＭ（ビジネスプロセス管理）ツールを中核に構成したアーキテクチャーである」とのことだが、「ＢＰＭツールは日本ではポピュラーな存在とは言えないが、カシオ計算機やリクルートでは採用実績がある。海外では、米国防総省が業務プロセス可視化の標準としてＢＰＭＮを採用し、業務改革に生かしている」程度のもので、「ＢＰＭツールを大規模システムに導入する難易度は低くない。特許庁の複雑な業務にＢＰＭツールを適用できるかが、システム刷新の成否のカギを握ることになりそうだ。「特許庁の業務には、ＢＰＭには落とし込めない例外処理もある。ＢＰＭツールが使える処理、使えない処理を見極めながら設計する」（特許庁ＣＩＯの木原氏）」[10]とのことであるが、このような状況判断をしていること自体がＢＰＭツールというアーキテクチャー採用が不適切であることを証示して余りある。「成否のカギを握る」？　冗談ではない。予め成功が見込めるアーキテクチャーを採用しないで、どうするのか。２度目の失敗など許されない状況ではないのか。もっと豊富な実績のあるアーキテクチャーを採用すべきであることは言うまでもないことだろう。血税 700 億円を投じる、「成否」の覚束ない開発計画を立てる特許庁、並びにそれを「好意的」に取り上げる御用ジャーナリズム、いずれに関してもタイトでシビアな取り組みが望まれる。

更にもう１つ、7 年に亘り係争中であったスルガ銀 - ＩＢＭ裁判が最近漸く決着が付いたので、取り上げることにする。新たな内容的な付加等はないのだが、2015 年 7 月 8 日、最高裁が両社の上告を棄却する決定を下したので、東京高裁の判決が確定したのである[11]。2008 年 2 月、スルガ銀が新勘定系システム開発の「プロジェクトが頓挫した責任は日本ＩＢＭにあるとし、約 116 億円の損害賠償を求める裁判」を東京地裁に提訴した。2012 年 3 月、東京地裁は日本ＩＢＭに約 74 億円の

---

[7] 業界ではＸＭＬを多用する厄介な事態を「ＸＭＬ地獄」と揶揄しており、それが全面的に正鵠を射ているとは思わないが、一面の真相を表現しているとは言える。そういう事情を踏まえていなかったとすれば、お粗末だったと言わざるを得ない。

[8] 同誌 p. 83

[9] 同誌 p. 83

[10] 同誌 p. 83

[11] 『日経コンピュータ』2015 年 7 月 23 日号 p. 11。なお、当節は引用の形式をとっていない箇所も含めて、内容的には同誌の記事に負っている。

支払いを命じる判決を行なった。プロジェクトの企画・提案段階で、日本ＩＢMが提案した勘定系パッケージ「Corebank」に関して、ＩＢM側の機能検証が不足していたことが、ＩＴベンダーが負う義務「プロジェクトマネジメント（PM)義務」違反に当たるとしたのである。日本ＩＢMが控訴した東京高裁では、2013 年 9 月、賠償額を約 42 億円に減額する判決が下された。日本ＩＢMのＰM義務違反は、パッケージを選定した提案・企画当初の段階ではなく、その後の要件定義を経て両社が最終意見書を交わした 2005 年 9 月以降に起きたと認定し、賠償額をその時期以降の実損害額である約 42 億円としたのである。高裁判決が確定したことの法的意義は次の通りである。ＩＴベンダーが負う「PM義務」という概念が判例として確立したことである。業界特有の多段階契約（元請～再委託等）といった契約形態に関わらず、ＩＴベンダーが損害の責任を負うことになる。それは、プロジェクトの円滑な遂行だけではなく、状況に応じてプロジェクトの抜本的見直しや中止を提言することも、ＩＴベンダーの責務として課せられる。他方、ユーザ企業が負う「協力義務」という概念も、明確になったのである。今回の判例では適用されなかったが、ユーザ企業が要件定義に協力的でない、あるいは適切な意思決定を下していないと認定されれば、ユーザ企業側が責任を問われることになる。アメリカと比べれば、日本では裁判での係争はまだまだ少ないし、今回の判決が直接的にソフトウェア開発実務に影響を及ぼすとまでは言えないとしても、各々の立場での責任（義務）が法的に明確化された意義は少なくないと言える。

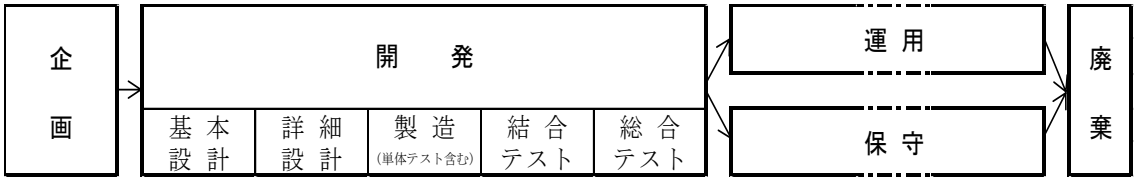
ソフトウェアを巡る昨今の情勢ということで、締め括りとして、所感めいたことを述べておきたい。凡そ 10 年程のことであるが、端的に言えば、ソフトウェアの 70 年近い歴史の中で未曽有の事態だが、失速し停滞してしまっている。スマホやタブレットの小刻みなバージョンアップは続いているが、クラウドやビッグデータ以降、新たなイノベーションと言えるようなことは出現していない。クラウドやビッグデータにしても、業界メディアでは他に話題がないから依然として取り上げられているが、それほど普及しているわけではないし、事業規模等で目覚ましい増加傾向の数値を挙げているわけでもない。それ以降の動きはぱったりと止んでしまっている。ほとんど前例のない事態と言える。これが、筆者のアンテナの感度が鈍く新たな動向を探知できていないだけならば良いのだが、どうであろうか。このまま減速状態が定常化するのかどうかは予測できることではないが、特段のことがない限りは、我が国では遅ればせのマイナンバー対応が消極的に行なわれ、2015 年問題[12]を何とか通過し、2020 年東京オリンピックまではそれなりの需要増があり、それに業界として対応するという既定路線を歩んでいくのであろうか。ソフトウェア業界の構造改善はなされないままに。

２．ソフトウェア・ライフサイクル

[12] メガバンクのシステム統合等のビッグ・プロジェクトで大々的な要員調達が行なわれるので、ソフトウェア業界としては人材不足になることの「通称」である。

ソフトウェアを体系的・総合的に捕捉するには、ソフトウェアのライフサイクルに沿うことが不可欠なことであると言える[13]。ソフトウェアのライフサイクルは、企画、開発、運用、保守、廃棄の５つの大フェーズ(プロセス)で構成されている。それを図示したのが図表Ａ－１-1 である。大フェーズ(プロセス)と称するのは、開発における設計等の工程をフェーズ(プロセス)というのが通称となっているので[14]、それと区別するためである。なお、廃棄に関しては、「システム管理基準」では「保守業務」の１項目と位置付けており[15]、「共通フレーム」では「運用・サービスプロセス」の１プロセスと位置付けており[16]、いずれも独立的な位置付けはしていない。しかし、ソフトウェアを継続的に利用していくための作業である保守又は運用と、利用を止めるための作業である廃棄とでは、その目的・性格が異なるので、たとえ作業量は少なく、短期間の作業ではあるが、廃棄を独立的な位置付けとする方が適切であると筆者は考えている。

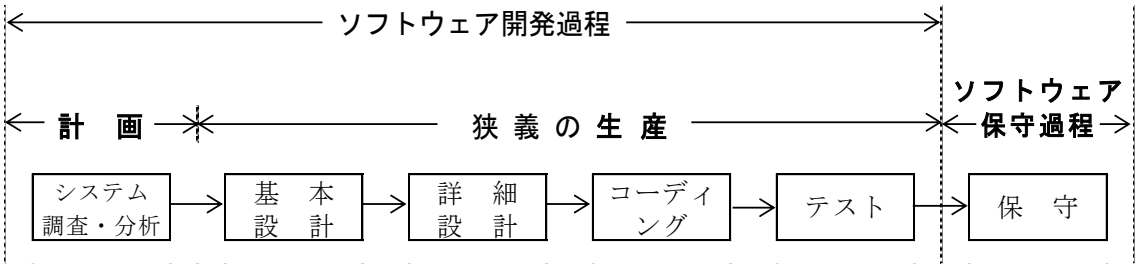
図表Ａ－１-1　ソフトウェア・ライフサイクルのモデル図



(出典：経済産業省商務情報政策局監修(2005)、ＩＰＡ（情報処理推進機構）(2013)を参考に、筆者作成)

それに対し、ソフトウェア会計研究の第一人者とも言える櫻井通晴が捉えているソフトウェア・ライフサイクルは、図表Ａ－１-2 の通りである。業界標準が出現しあるいは形成された時期と突合すると、櫻井の理解はほぼ同期的であり、業界動向をキャッチアップした上で、ソフトウェア会計研究を行なっていることが、如実に窺われる。

図表Ａ－１-2　ソフトウェア・ライフサイクルの各工程（櫻井作成）



(出典：櫻井通晴編著(1993)p. 87（櫻井通晴記))

それでも、多少の不備が看取される。第１に、運用を失念していることである。開発した後に、

---

[13] ソフトウェア・ライフサイクルの詳細は、ソフトウェア基礎論第２章で別途取り上げる。ここでは、エッセンスとも言うべきことと、会計的な関わりに限定して取り上げる。

[14] 開発におけるフェーズ(プロセス)の区分に関しては、主なものだけでも幾通りかあるが（ＩＰＡの「共通フレーム」等参照)、図示したのはその１例である。

[15] 経済産業省商務情報政策局監修(2005)pp. 12, 384-385

[16] ＩＰＡ(2013)pp. 192-194

保守と同時並行的に行なわれる運用を取り上げていないことは、重大な欠落と言わなければならない（販売用ソフトウェアの「客先導入」は取り上げており[17]、これは運用の一環であるが、あくまで初期的・一時的な作業に過ぎず、これをもって運用をカバーしていると見做すのは無理である）。会計的に重要か否か以前に、視野に入れておかなければならないことである。第２に、「計画」（システム調査・分析）は、「ソフトウェア開発過程」の一部としていることからすれば、企画という独立的な「過程」として捉えているのではないと言える。従って、企画が欠落していることになる。第３に、廃棄に関しては、本文では取り上げているけれども[18]、図示はしていない。ソフトウェアを主題とし、且つソフトウェアの設計書等における図表の決定的な重要性を鑑みれば、少なからぬ瑕疵あるいは遺漏と言わざるを得ない。それでも、1990 年代前半に、ライフサイクルを視野に入れて、ソフトウェア会計を構想した先駆性・卓越性は際立っている。

ソフトウェアのライフサイクルをベースとした、櫻井のソフトウェア会計処理案(実務指針案)と対比すると、ソフトウェア会計基準がライフサイクルを視野に入れず、従ってソフトウェアを全域的にカバーしておらず、開発を中心に取り扱うことに留まっていることは明らかである。ソフトウェアの動向との連係という点でも、櫻井の案よりも、後退的であると言わざるを得ない。具体的に確認してみよう。ソフトウェア会計基準等がソフトウェアのライフサイクルをどの程度カバーし、その各々に係る会計処理をどのように設定(規定)しているかを整理すると、図表Ａ－１-3 のようになる。カバレッジという点では、櫻井の案がカバー範囲は(筆者案を除けば)最も広く、我が国のソフトウェア会計基準は最も狭隘な設定であることがわかる。ＩＦＲＳの基準は、研究・開発という枠組みのなかでソフトウェアを捕捉していることからも、主としてソフトウェアの開発という大フェーズ(プロセス)を取り扱っているに留まり、ライフサイクルをカバーしているとは到底言えない。ＩＦＲＳへのコンバージェンスあるいはアドプションという趨勢の中では全く問題視されていないが、IAS 第 38 号「無形資産」はソフトウェアのライフサイクルという観点からみると、カバレッジが不十分なものであり、ソフトウェア会計基準としては現行基準と置換・代替が望まれるほど優良な基準ではない。近年は検討が中断されているが、企業会計基準委員会において、2010 年前後に現行のソフトウェア会計基準に替えて、実質的には IAS 第 38 号「無形資産」を丸呑みにしたソフトウェア会計基準を新設する検討がなされていたが[19]、ソフトウェアに即しての改善にはならないと言わなければならない。いずれにしても、ライフサイクルの全域をカバーしていないということは、ソフトウェアに係る会計処理を全体的には規定していないということである。換言すれば、無規定の部分は会計慣行等に委ねることになるのであり、会計基準としては不備なものである。筆者は、櫻井の構想を批判的に継承し、それらに替わるものとして、ライフサイクルの全域をカバーし、不

[17] ソフトウェアの資産性に関する検討委員会・委員長櫻井通晴、櫻井編著(1993)pp. 37-38
[18] 同書 pp. 61-63
[19] 「無形資産会計基準とソフトウェア会計基準（仮称）の基本的な考え方」(https://www.asb.or.jp/asb/asb_j/minutes/20100311/20100311_06.pdf, 2015/07/25 18:31 アクセス)

備のない会計処理案を提示することを目指している。大フェーズ毎の会計処理の詳細は、本論で展開する段取りである。

図表A－1-3　ソフトウェア・ライフサイクルのカバレッジ並びに会計処理の比較

	櫻井通晴(1993年)のカバレッジ		アメリカ基準(1985)のカバレッジ		会計基準(1998年)のカバレッジ		IAS38(1998)のカバレッジ		筆者のカバレッジ	
企画	△	(1)開発の前段階に限定(P.87) 〔会計処理〕 (1)研究開発費(新奇のパッケージ) (2)販売費(受注前) (3)製造原価(受注後)(全てP.36-37)	△	(1)製品の製作が可能であるということを確定するために必要なすべての計画(Planning)(4)、を企画と解する限り 〔会計処理〕 (1)研究開発費(開発の(1)と同範囲)	×	(1)カバーしていない	×	(1)カバーしていない	○	(1)全般 (2)開発以外に、全体的な戦略ないしライフサイクル 〔会計処理〕 (1)資産計上(ソフトウェアの購入・開発(受注制作を除く)又は運用改善に連動するドキュメント作成を伴う場合) (2)上記以外費用処理
開発	○	(1)全般(P.86-88) 〔会計処理〕 (1)研究開発費(新奇性の調査・分析) (2)資産計上(上記以外の開発全工程)(P.36-37)	○	(1)全般、計画(Planning)、設計(Designing)、コーディング(Coding)及びテスト(Testing)(4) a. 製作過程が、詳細プログラム設計を含む場合(4) b. 製作過程が、詳細プログラム設計を含まない場合(4) 〔会計処理〕 (1)研究開発費(技術的実現可能性が確定されるまで)(5) (2)資産計上(技術的実現可能性が確定した後)(5) (3)購入の場合も、同様の処理(7)	○	(1)全般(三、四１－３) 〔会計処理〕 (1)研究開発費(研究開発、研究開発費に該当する部分)(三、四２) (2)資産計上（資産要件充足）(四２、３) (3)上記以外費用処理	○	(1)自己創設無形資産にとっての研究局面、開発局面(51-64) 〔会計処理〕 (1)研究であれば発生時費用処理(54) (2)開発で、かつ、資産要件を立証できれば資産計上(57) (3)取得(購入等)の場合は資産計上(25-26)	○	(1)全般 (2)再利用、オープンソース利用、プロジェクトの失敗を含む 〔会計処理〕 (1)研究開発費(研究開発) (2)資産計上(研究開発以外、開発の全工程) (3)振替処理(再利用) (4)無償取得のオンバランス化 (5)特別損失(ソフトウェア仕損)
運用	△	(1)客先導入に限定(P.37-38) 〔会計処理〕 (1)期間費用(売上原価または販売費、ユーザー側購入原価、P.37-38)	△	(1)部分的なカバー 〔FASB-ASC Topic350-40におけるPost-Implementation/Operation Stageは、運用の規定と見做すことはできるが、具体的にはtraining costs and maintenance costsだけを挙げていることから察すると(23)、運用の全体を対象としているとは言い難い。〕 〔会計処理〕 (1)費用処理	△	(1)導入に限定(実務指針14,16,38,40) 〔会計処理〕 (1)資産計上(購入の付随作業は取得原価)(実務指針14,38) (2)上記以外費用処理(実務指針16,40)	△	(1)無形の資源の取得、開発、維持又は強化(9)という規定は、運用を想定しているとは思い難い。 (2)稼働状態(28)を運用と解するならば、一応カバーしていることになる。 〔会計処理〕 (1)稼動状態を運用と解するならば、費用処理(29-30)	○	(1)全般 〔会計処理〕 (1)資産計上(運用改善、運用ソフトの取得) (2)上記以外費用処理
保守	○	(1)全般(P.87) 〔会計処理〕 (1)機能の改良、強化は開発と同処理(P.59-60) (2)上記以外、発生時費用(P.59-61)	○	(1)製品の機能強化(Product Enhancement)(2) (2)保守(maintenance)及び顧客支援(customer support)(6) 〔会計処理〕 (1)製品の機能強化は資産計上(ただし技術的実現可能性が確定されるまでは研究開発費)(5) (2)保守及び顧客支援は費用処理(6)	△	(1)市場販売目的に限定(実務指針9,33) 〔会計処理〕 (1)著しい改良は開発と同処理 (2)著しくない改良、資産計上 (3)上記以外、費用処理(全て実務指針9,33)	△	(1)維持又は強化(9) 〔会計処理〕 (1)(機能)強化等が開発に該当し、かつ、資産要件を立証できれば資産計上(57) (2)上記以外(維持)は費用処理と解せられるが、明示的な規定はない。	○	(1)全般 〔会計処理〕 (1)資産計上(維持、改良) (2)費用処理(バグ対応) (3)除却処理(部分的削除)
廃棄	○	(1)全般(P.61-63) 〔会計処理〕 (1)除却処理(P.61-63)	×	(1)カバーしていない	×	(1)カバーしていない	○	(1)廃棄及び処分(112-117) 〔会計処理〕 (1)処分方法に応じた処理を行う(114-117)	○	(1)全般 〔会計処理〕 (1)除却処理(償却残高がある場合) (2)費用処理(償却残高がない場合)

(出典：櫻井通晴編著(1993)p.24-90 (櫻井通晴記)、各ソフトウェア会計基準から抽出、筆者分追加)

３．ソフトウェアのプロダクトとプロセス

経営学におけるイノベーション研究では、イノベーションをプロダクトとプロセスの両面(両軸)から捕捉することが広く普及している[20]。ソフトウェアをプロダクトとプロセスの両面から捕捉す

[20] Tidd, Bessant, Pavitt(1997)等

る筆者の方法は、それを参考にしたものである。

研究開発とイノベーションは同義ではないが、研究開発において、新奇な製品と並び、新奇な製法(生産方法)が追求されることは周知のことである。従って、「研究開発費等に係る会計基準」も、それを踏まえた定義がなされている（一1）。ところが、その基準の一部であるソフトウェアに係る規定では、市場販売目的のソフトウェアの資産認識に関してはプロセスによる捕捉も行なわれているが（意見書三３(3)）、それ以外では専らプロダクトによる捕捉に片寄っており、プロセスによる捕捉は貫徹されていない。しかし、プロダクトとプロセスの両面(両軸)から捕捉することで、ソフトウェアを十全に捕捉することが可能となるのであり、それは会計的にも同様である。

図表A-1-4　ソフトウェアのプロダクトとプロセスの捕捉

	プロダクトの捕捉	プロセスの捕捉
企画	(1)企画における成果物は一般的にはドキュメントである（企画書等）。 (1-1)しかも、企画のドキュメント類は比較的量的に少なく、コンパクトなものあるいはエッセンスといったものであることが多い(提案用であるため)。 (1-2)従って、行間を読むあるいは眼光紙背に徹するようにして、実現像であるソフトウェア（運用を含む）を想到することを要する。 (2)企画では、狭義のソフトウェアは作らず、たとえ作るとしてもプロトタイプくらいである。 (2-1)従って、ドキュメント以上に、完成形を想到する努力による補完的な捕捉を要する。 (3)いずれの場合でも、成果物だけからは、調査の博捜性や分析・検討の必要十分性や内容の充実度等を窺知することはかなり難しい。	(1)企画における、検討に到った経緯、調査・分析・検討のプロセス（中間的な成果物である各種資料を含む）、そして成果物の作成プロセス（推敲・改訂等）を捕捉する。 (1-1)これにより、プロダクトの捕捉では難しい捕捉が可能になり、十全な捕捉となる。
開発	(1)研究開発の場合、製品の新奇性を捕捉する。 (2)機能並びに非機能要件の一部を捕捉する。 (3)ソフトウェア再利用は、精査によって捕捉する。 (4)プロジェクトの失敗は、プロダクトの品質に関しては捕捉可能である。	(1)研究開発の場合、製法(開発プロセス)の新奇性を捕捉する。 (1-1)研究開発の場合、製品の新奇性のための開発プロセス(試行錯誤の組込み)を捕捉する。 (2)プロセスを通じて、非機能要件を捕捉する。 (3)ソフトウェア再利用は、プロセスを通じて容易に捕捉可能である。 (4)プロジェクトの失敗は、プロセスを通じて確実に捕捉可能である（ＱＣＤの指標に顕在化しない事象を含めて）。
運用	(1)プリンタ出力等独自の成果物もあるが(但しソフトウェアではなく、コンテンツというべきだが)、利用が適宜に信頼性を持ってなされることが主たる成果なので、直接的なプロダクトの捕捉は無形的で容易ではない。 (2)運用改善は、各種環境定義類の変更等のプロダクトを伴うので、プロダクトによる捕捉は可能ではあるが、効果等の捕捉はやはり難しい。	(1)運用はプロセスそのものと言えるものなので、プロセスを通じた捕捉こそ本領と言える。 (1-1)より具体的には、ＳＬＡ(Service Level Agreement)の達成状況を計測することにより、定量化が可能である。 (2)運用改善は、利用に関わるプロセスを捕捉することで十分な捕捉となる。
保守	(1)プロダクトの捕捉では、開発との相違は捕捉が難しい(規模の大小くらい)。	(1)プロセスを通じた捕捉により、開発と異なる工程の「二瘤ラクダ」的な様相が容易に捕捉可能である(設計における影響範囲の調査・分析とデグレード回避のための回帰テストという「二瘤」)。
廃棄	(1)廃棄は、少なくとも本番環境における該当ソフトウェアを消去し、利用不可とすることであるから、プロダクトはないに等しい（保管・保存のためにデータを含めてバックアップを取ることはありうる程度）。 (1-1)従って、廃棄に関するドキュメントによる間接的な捕捉に限られる。	(1)計画書や報告書等のドキュメント作成を除けば、廃棄は安全・確実に消去する作業というプロセスが過半を占めるのであり、プロセスによる捕捉が主要な捕捉である。

（出典：筆者作成）

図表A−１−4 は、ソフトウェアをライフサイクルの大フェーズ(プロセス)毎に、敢えてプロダクトとプロセスに分節化し、どのように捕捉できるかを整理してみた。また、図表A−１−5 は、それらの捕捉がソフトウェアの会計処理(具体的には資産の認識及び測定)にどのように適用され、明確化に寄与するかを挙示してみた。これらに関しては、既存の会計基準等がどの程度当て嵌まるかを抽出しようにも、ライフサイクル以上に疎遠なので行なわず、筆者の考想だけを掲示することにした。

図表A−１−5　ソフトウェアのプロダクト及びプロセスと会計処理の関連性

	資産の認識及び測定
企画	(1)企画におけるプロダクトは、ほとんどがドキュメントであり（企画書等）、しかもプログラム等と同期的に作成されることは少ないので、ソフトウェアとは見做されにくい。 (1-1)プロトタイプ等のソフトウェア(プログラム等)が制作される場合には、開発に準ずるものと見做すことができる。 (2)企画**プロセス**における調査・分析・検討を併せて捕捉することにより、企画作業の内容がソフトウェアとしての実現可能性、利用業務に対する適合性・利便性、代替の選択的採用、投資効果等の検討を行なったものであることが確認できる（資産要件の充足性並びに資産の範囲）。 (3)企画に限らず、ライフサイクル全般に共通することだが、資産の測定において工数が必須とも言える項目であり、それ故**プロセス**の捕捉が不可欠である。 (4)企画では、**プロダクト**であるドキュメント（プロトタイプを制作した場合はそれを含む）とその作成**プロセス**に費やした作業工数に基づき、資産額を測定する。
開発	(1)**プロダクト**であるソフトウェアと、その開発における開始から終了までの**プロセス**により認識・測定する。 (1-1)購入の場合は製品の受入、委託開発の場合は成果物の受入により認識・測定する。 (2)プロダクトに関しては、ソフトウェアの実装された機能要件並びに非機能要件が資産要件を充足しているか否かにより、資産として認識する。 (3)ソフトウェア測定に基づき、ソフトウェアの資産額を測定する。基本項目である規模はプロダクト、工数・(人月)単価はプロセスに係る項目である。一般的な原価計算による場合でも、ほぼ同様の計算項目による。 (3-1)購入の場合は購入価格(付随費用含む)、委託開発の場合は請負金額により測定する（規模単価ベースの場合は規模、工数単価ベースの場合は工数により算出する）。 (4)ソフトウェア再利用に関しては、プロダクトにおける既存リソースの取扱い(費用負担等含む)と、再利用のプロセスにより認識・測定する。 (5)ソフトウェアの仕損に関しては、品質は主としてプロダクト（中間の成果物を含む）により、開発費と納期は主としてプロセスにより認識・測定する。
運用	(1)運用のためのツール等の購入又は開発は、**開発と同様の認識及び測定**をする。 (2)運用改善は、**プロダクト**としては各種環境定義類又は運用ドキュメント（運用マニュアル等）、**プロセス**としてはそれらを作成又は変更する作業プロセスにより、運用が改善されたことを認識・測定する。 (a)なお、資産の測定の場合を含め、運用費用の算定は委託における一括請負を除けば、工数と(人月)単価に基づき算定する。
保守	(1)保守は、規模や工程の内容は開発と多少異なるが、**開発とほぼ同様の認識**をする。 (2)保守は、**開発と概ね類似の測定**をする（規模と工数を測定項目とする。但し、母体規模(既存分の規模)を考慮する点が開発と異なる）。
廃棄	(a)廃棄は、資産と成り得る成果物を産出しないので、資産認識は行わない。 (b)償却残高のあるソフトウェアを廃棄する場合は、除却処理を行う。除却額は、償却残高と廃棄の所要諸費用とする。 (c)廃棄作業の費用算定は、委託における一括請負を除けば、工数と(人月)単価に基づき算定する。

（出典：筆者作成）

ライフサイクルの大フェーズによって、あるいは資産認識ないし測定によって、その必要度は異なるけれども、プロセスという側面からの捕捉は有用であり、特に測定に関しては欠かせないことであると言える。図表A−１−4 並びに図表A−１−5 によって、具体的に示し得たのではないかと考える。それに対し、ソフトウェア会計基準には様々な問題が内包されており、その多くはソフトウェ

アに関する理解不足に基因しているが、換言すれば、プロダクトとプロセスの両面(両軸)捕捉が不徹底であることの帰結であると言うことができる。また、プロセスへの着目が稀薄であることが、プロセスの総合的展開であるライフサイクルを射程に入れられなかったこととも相関的・相即的なことである。

なお、研究開発費等会計基準(ソフトウェア会計基準)の定義にある製品と筆者のいうプロダクトは同義と言って差支えないが、製法(生産方法)と筆者のいうプロセスの異同に言及しておきたい。製法は技術（生産方法、製造方法の具体化）を意味するが、それに対してプロセスは製法という技術を中核としているが、その技術を採用し実際にそれを適用して制作する作業(の遂行)までを包含している。それ故、プロセスは技術を実施する場面までを含んでいるので、範囲が広いと言える。

４．ソフトウェア開発失敗の基底因

筆者にとって、ソフトウェアはこれまでに実現してきたこと並びに今後まだまだ様々に切り開いていくであろう可能性によって強い関心の対象であり続けるが、それにもかかわらず「稚拙」とも言えるプロセスが問題含みであることも、興味深い。ソフトウェア開発は、何故失敗するのだろうか。技術的な進歩・改善があるのに、それでも開発の失敗はどうして減少しないのだろうか。会計的な関心と共に、筆者がソフトウェアに関わって、終始念頭にあるのは、このことである。そして、熟慮に熟慮を重ね、遂に思い至った事態は、次のことである。１つの標語のように集約すると、「プロダクト・イノベーションとプロセス・イノベーションの跛行性」ということになる。それを基底因としていることで、開発の失敗はなくならないし、顕著に減少もしないのである。どういうことか。プロダクトのイノベーションは、製品という単体、複合体、もっと精細で共通的な要素技術のレベルまで含め、膨大な量の事象が考案され、出現し、利用されるに到っている。ところが、もう一方のプロセスに関するイノベーションは、70 年ほどのソフトウェアの歴史を通して、おそらく全てを列挙しても、論文の１ページにも満たない、ごく僅かなものが考案され、出現し、利用されるようになっただけである。この凄まじいギャップは恐るべきことである。次々にイノベーティブなプロダクトに取り組まなければならないのに、今に到るまで圧倒的に労働集約的な開発作業のプロセスは、恐ろしく旧式の、旧態依然のやり方で遂行しなければならないのである。この両者が非適合的であることは、判然としているではないか。しかも、事態は何ら固定的ではないのである。改善の努力がなされなかったわけではない。だが、束の間馴染んだとしても、直ちに新たなイノベーティブなプロダクトが出現し、対応しなければならないのである。そうした事態の連続であったと言える。これが、開発に関わる失敗の技術的な基底因である。更に、ＩＴが合理化（効率化）の手段という側面を有しており、そうであるならば、自らもその標的にならないはずはなく、厳しい開発条件を課せられるのである。技術的なことよりも、経済的な条件を中心としたことが多い。これが加重して、ソフトウェア開発の失敗は発生し続けている。こうした事態を抜本的に改善し、安定

的に開発が可能なようにしていくことは、果たして可能なのか。繰返し「特効薬」が喧伝されるが、それが誇大広告ないし虚偽でしかないことは、ソフトウェアの名著『人月の神話』(*The Mythical Man-Month*)で著名なフレデリック・ブルックス（Frederick Phillips Brooks, Jr.）が1986年に発表した論文「銀の弾などない―ソフトウェア・エンジニアリングの本質と偶有的事項」(*No Silver Bullet – essence and accidents of software engineering*)で遍く知られていることである。従って、当然のことながら、地道な改善の積み重ねが必要であることは言うまでもない。筆者にとって、ソフトウェア会計の体系的整備と併せて、微力ながらそうした失敗を軽減していく方策を追求していくことがライフワークとも言うべき課題である（何よりもプロセス・イノベーションの進展、堅実なプロジェクト管理、要求工学の実質的な形成、組織的な情報リテラシーの向上が求められる)。

但し、当論文の内容からは逸脱することなので、これ以上の言及は今は行なわないことにする。

５．ソフトウェア会計研究の在り様

本論と関わることでは、筆者にとっての会計研究の在り様を示唆的に述べておきたい。実際的には、本論の全篇を通じて立証というか、実践して示すべきことであるので、ここでは簡潔に箇条書きのような形で提示するに留めたい。それでも、本論の理解に資するために述べたいと考える。

（1）ソフトウェア会計の研究方法

ソフトウェア会計の研究には、対象であるソフトウェア分野の専門的な知識が必要不可欠である。櫻井通晴が切り開いてきた研究実績が何よりもそれを示している。そういう言い方はしていないが、フィールドワーク的研究方法である。櫻井通晴自身は既にソフトウェア会計研究から離れてしまっているが、筆者はそれを発展的に継承していこうと志している。

（2）ソフトウェア会計研究の目標

筆者にとって、ソフトウェア会計研究の目標は、簡潔に掲げると次の通りである。

①ソフトウェア会計を体系化すること。それには、ソフトウェアのライフサイクルに沿って構成する以外には方策はないのである。

②ソフトウェアの資産範囲を拡大すること。ソフトウェア会計基準は、一応資産として認めているが、まだ極めて限定的でしかない。ソフトウェアの利用実態、制作実態に見合った認識・測定を行なうことにより、資産範囲を拡大することが最重要の実質的な目標である。

③ソフトウェア開発の失敗への対処の一翼を担うこと。会計だけで対処できることではないが、アンカーたる会計が貢献できることはある。それを具体的に行なっていくことである。

④ソフトウェアの情報開示を拡大すること。如何なるソフトウェアであり、どのように取得し、利用しているか、その具体的な態様を可能な限り計数的に情報開示していくことを目指す。現行の情報開示では、財務諸表はもとより、内部統制報告書等でも形式的な定型句表現に留まり、具体性

に欠けており、大幅な刷新が必要である。

⑤ソフトウェア測定の方法を取り入れ、貨幣的な精度を向上させること。対外的な取引の場合には契約に基づく収支が発生し、測定の精度は保証されるが、内部的な作業で収支とは間接的な関係となる場合の精度をソフトウェア測定の方法に則ることで、精度の向上を図ることができる。

これらのことを、本論（ソフトウェア基礎論を含め）を通じて達成していると考えている。但し、情報開示に関しては、開示すべき情報の内容は詳らかにしているが、開示場所や開示の書式・様式等の形式面に関しては、系統立った形では十分に展開していない。これに関しては、当論文とは異なった構成で別途行ないたいと考えており、残された課題とする。

序論の最後として、当論文の構成を簡略に提示しておきたい。序論のあとは、ソフトウェア基礎論を配し、第１章から第３章において、ソフトウェアの定義、ソフトウェア・ライフサイクル、ソフトウェアの分類を各々説示している。これらは、本論におけるソフトウェア会計の考察の基礎となる諸概念であり、本論に先立って定礎しておく必要があり、会計的な議論は極めて少ないが、配置している。そのあと、当論文のメインである本論を配し、最後に結論を配している。本論は、言うまでもなく、ライフサイクルに沿った構成としている。企画に１章、開発に５章、運用に２章、保守に１章、廃棄に１章を配している。開発に最も多い５章を充てているのは、それだけ問題含みだということである。各々必要な範囲で当該大フェーズにおけるソフトウェアに関する概観を行ない、その上で会計的な考察を行なうことにしている。ソフトウェア会計の全ての論点や課題を尽くしているわけではないが[21]、ライフサイクルに沿うことで取り扱うべき事項は全て取り上げているということができる。今後、筆者の研究を踏まえずに、ソフトウェア会計を研究することはもう１度筆者が行なってきた営為を繰り返さなければならないことになるとは言えよう。そのように自負している。

[21] 例えば、組込み系ソフトウェアの会計的な取り扱いがあるが、これに関しては別途独立的な論文で考察している（現時点では未発表）。

ソフトウェア基礎論

第１章　ソフトウェアの定義

１．ソフトウェア会計基準におけるソフトウェアの定義

（1）ソフトウェア会計基準のソフトウェアの定義

（2）櫻井通晴編著(1993)のソフトウェアの定義

（3）ＪＩＳ規格（ＩＳＯ/ＩＥＣ標準）のソフトウェアの定義

２．ソフトウェア会計基準のソフトウェア定義の問題点１――各種環境定義類の欠落

（1）各種環境定義類とは何か

（2）ソフトウェア会計基準にとっての不可欠性

３．ソフトウェア会計基準のソフトウェア定義の問題点２――ドキュメントの規定の不備

（1）ドキュメントの位置付け

（2）ドキュメントの対象範囲

４．ソフトウェア会計基準のソフトウェア定義の問題点３――その他

（1）構造体として捉えるべきこと

（2）「規則」の取り扱い

（3）コンテンツとの境界

５．ソフトウェア会計基準におけるソフトウェア定義改訂案

（1）ソフトウェア定義改訂案の提示

（2）ソフトウェア定義改訂案の会計的含意

補遺　「ソフトウェア工学」における１つのソフトウェア定義

図表Ｂ－１-1　著作権法上のソフトウェアの定義

図表Ｂ－１-2　ソフトウェア・ドキュメント一覧

図表Ｂ－１-3　ソフトウェアの定義

第１章　ソフトウェアの定義

１．ソフトウェア会計基準におけるソフトウェアの定義

ソフトウェアを総合的に捕捉するには、それ相応の枠組みが必要である。筆者は、２つの基軸を据えた枠組みを構想している。１つの基軸は、プロダクトとしてのソフトウェアを捕捉するためのものである。もう１つの基軸は、ソフトウェアのプロダクトを制作ないし利用に供するプロセスを捕捉するためのものである。ソフトウェア基礎論の第１章である本章は、そのプロダクトを捕捉する基軸の基礎である、ソフトウェアの定義を主題的に考察する。予め先取り的に言えば、基礎論の第２章は、プロセスの基軸となるソフトウェア・ライフサイクルを主題的に考察することになる。これらによって、ソフトウェアを総合的に捕捉するための枠組みの基軸を提示することになる[22]。

そうした枠組みに拘らなくとも、一般的に、ソフトウェアの定義は、どこまで精緻にソフトウェアを把握した上で議論をするか、その「試金石」とも言うべきものである。そして、ソフトウェア会計基準においては個々の規定を支えるあるいは成り立たせる基礎である。それ故、例え簡潔なものであっても、周到で且つ十全なものでなければならない。現行基準の定義は、そのようなものであり得ているだろうか。

（1）ソフトウェア会計基準のソフトウェアの定義

ソフトウェア会計基準の基準（本体）では、

「ソフトウェアとは、コンピュータを機能させるように指令を組み合わせて表現したプログラム等をいう」(基準一2)、

と定義している。

実務指針では「ソフトウェアの概念・範囲」という形で、

「ソフトウェアとは、コンピュータ・ソフトウェアをいい、その範囲を次のとおりとする。

①　コンピュータに一定の仕事を行わせるためのプログラム

②　システム仕様書、フローチャート等の関連文書」(実務指針６)、

と定義している。

関連して、実務指針では、「コンテンツ」を取り上げ、「コンテンツは、ソフトウェアとは別個のものとして取り扱い、本報告におけるソフトウェアには含めない。／ただし、ソフトウェアとコンテンツが経済的・機能的に一体不可分と認められるような場合には、両者を一体として取り扱うことができる」(実務指針７）とし、より説明的に、「ソフトウェアがコンピュータに一定の仕事を行

[22] プロダクトとプロセスという捕捉の仕方は、例えば研究開発は新奇的な製品(プロダクト)又は製法(プロセス)を追求することであると集約できるように、取り立てて目新しいものではないが、ソフトウェアを明確にこのような基軸で捕捉することはまだ十分に定着しているとは言い難い。

わせるプログラム等であるのに対し、コンテンツはその処理対象となる情報の内容である。コンテンツの例としては、データベースソフトウェアが処理対象とするデータや、映像・音楽ソフトウェアが処理対象とする画像・音楽データ等を掲げることができる。ソフトウェアとコンテンツとは別個の経済価値を持つものであることから、本報告ではコンテンツはソフトウェアに含めないこととした。／また、ゲームソフトは、一般的にはソフトウェアとコンテンツが高度に組み合わされて制作されるという特徴を有している」(実務指針29)、としている。

これらが、ソフトウェア会計基準におけるソフトウェアの定義である。筆者は大いなる疑問を抱懐してきたが、その問題に直入するのではなく、一旦は迂回的に、会計基準設定前後になされた議論を振り返ることにする。

（2）櫻井通晴編著(1993)のソフトウェアの定義

会計基準設定前のものだが、代表的なものとして、櫻井通晴編著(1993)を取り上げる。

「本指針[23]で、ソフトウェアとはコンピュータ・ソフトウェア（以下、ソフトウェアと称する）のことをいう。ソフトウェアとは、コンピュータに一定の仕事を行わせるためのプログラム、手順、規則およびその関連書類をいう」[24]とし、注記で次のような補足説明を行なっている（少々長いが、重要なので、省略せずに引用する)。

「ソフトウェアの定義については、概ね、次の３つの見解がある。

1) プログラム

2) 1)＋手順、規則、およびその関連書類

3) 2)＋保守

以上のうち、1)は法律的な観点に立脚している。ここでプログラムとは、コンピュータを機能させることを通じて、一定の結果をうるための指令の組合せをいう。

2)は、基本的にはソフトウェアをもって、「データ処理システムの運用に関係する計算機プログラム、手順、規則及びそれに関連する文書」であるとするＪＩＳ規格にもとづく見解およびそれに関連した定義であり、会計・税務の文献ではほとんどこの見解がとられている。関連文書には、システム仕様書、フローチャート、運用マニュアルなどが含まれる。ソフトウェアには、基本ソフトウェアのほか、アプリケーション・ソフトウェアが含まれる。

3)は、アウトプットたる開発されたソフトウェアだけではなく、開発されたソフトウェアへのサ

---

[23] 同書は、「ソフトウェア会計実務指針〔案〕」を提示し、その解説並びに関連する事例を掲示するという形式並びに構成としているので、「本指針」という表現になっているのである。

[24] ソフトウェアの資産性に関する検討委員会・委員長櫻井通晴、櫻井編著(1993)p.26。なお、組み込み系ソフトウェアに関する注記をしており(「注１」、同書p.27)、会計基準でも取り上げられているが(実務指針17、41)、定義にも関係はするが、どちらかと言えばソフトウェアの分類に主として関わることなので、定義を主題とする本章では立ち入らない。

ービス活動である保守（maintenance；メンテナンス）を含むとする見解である[25]」（注3）[26]。

更に、次のように補足している。「本指針ではソフトウェアの資産性などの会計基準を検討しているから、ソフトウェアは財務会計や税務との関連でとらえられている。以上から、本指針でソフトウェアといえば、一般には2)の定義が意味されている。しかし、保守も本指針の対象としているという意味からすれば、本指針でソフトウェアは拡張された意味でとらえられているともいえる」[27]。

一応の結論としては、会計基準に「継承」される2)の定義を採択しているわけだが、末尾の留保的な言い方では、3)を採用しているかのようにも受け取れる。だが、3)はカテゴリー・ミステイクに陥った「定義」である。ソフトウェアの定義は、ソフトウェアというプロダクトを対象とし規定するものである（プログラムや「関連書類」がプロダクトであることは明白であろう）。ところが、「保守」(SFAS86の「機能強化」)はプロダクトではなく、ソフトウェアに関するプロセス（の１つ）である。それらを混同することはカテゴリー・ミステイク以外の何物でもない。ソフトウェアの定義として、3)が成り立つとするのは論理的な誤りであり、「ソフトウェアは拡張された意味でとらえられているともいえる」というのは範疇的な混濁の無自覚的な吐露でしかない。

次に、櫻井通晴編著(1993)が一応採択した 2)の定義と、会計基準の定義は、「微妙に」異なっていることに関説する。会計基準は、2)におけるプログラムと関連書類だけを採択しているといえる。それならば、2)の定義における「手順、規則」とは、何であろうか。櫻井通晴編著(1993)にはそれ以上の具体的な説明はなく、「ＪＩＳ規格にもとづく見解およびそれに関連した定義」としているだけである。そこで、依拠したであろうＪＩＳ規格を確認することにしよう。

（3）ＪＩＳ規格（ＩＳＯ/ＩＥＣ標準）のソフトウェアの定義

櫻井通晴編著(1993)は典拠を具体的に挙示していないので特定し難いが、年代的には JIS:1994年版よりも古い版に依拠していることは間違いないであろう。ISO/IEC 2382-1 Information Technology-Vocabulary-Part1:Fundamental Terms は、1974年に制定され、1984年と1993年に改正されており、JIS X 0001「情報処理用語－基本用語」は、(前身の独自規格におけるコンバージェンスを止めて、ISO/IEC 2382-1 を「翻訳し、その技術的内容を変更することなく作成した日本工業規格である」(同書冒頭)というアドプションの形で)1987年に制定され、1994年に改正されている。

ISO:1974年版のソフトウェアの定義は

‘*Computer programs*, *procedures*, rules, and any associated *documentation* concerned with

---

[25] 櫻井編著(1993)ではアメリカ基準SFAS第86号を取り上げてはいるが(pp.144-161、櫻井通晴記)、ソフトウェアの定義には言及していない。SFAS(1985)第86号のソフトウェアの定義は、「コンピュータ・ソフトウェア製品、ソフトウェア製品及び製品という用語は、コンピュータ・ソフトウェア・プログラム、一群のプログラム及び製品の機能強化を含んで」（２.）いるというものであり、3)の典拠と言うべきものである。

[26] 同書p.27

[27] 同書p.27

the operation of a *data processing system*’(01.04.04)[28]であり、

ISO:1984 年版のソフトウェアの定義は

‘Intellectual creation comprising the programs, procedures, rules and any associated documentation pertaining to the operation of a *data processing system*’(01.04.04)[29]であり、

ISO/IEC:1993 年版のソフトウェアの定義は

‘All or part of the *programs*, *procedures*, rules, and associated documentation of an *information processing system*’(01.01.08)[30]である。

JIS:1987 年版のソフトウェアの定義は

「データ処理システムを機能させるための、プログラム、手順、規則、関連文書などを含む知的な創作」(01.04.04)[31]であり、

JIS:1994 年版のソフトウェアの定義は

「情報処理システムのプログラム、手続き、規則及び関連文書の全体又は一部分」(01.01.08)[32]である。

各版での語句や構文的な異同は少なくないが、実質的には大同小異である。例えば「データ処理システム」(*data processing system*)が「情報処理システム」(*information processing system*)に改正されたのは呼称の時代的な変化と言うべきもので、内容的に特段の変更を意味するわけではない。最も肝心なソフトウェアの構成要素を「プログラム」(*programs*)、「手続き」(手順、*procedures*)、「規則」(rules)、「関連文書」(associated documentation)であるとしていることには全く変更はない(最新版でその範囲を「全体又は一部分」(All or part)と慎重に規定しているが、判断基準を伴示していないので、却って曖昧になってしまっている)。

以上により、櫻井通晴編著(1993)が依拠したであろうＪＩＳ規格（並びに原典であるＩＳＯ/ＩＥＣ標準）の内容を確認することができたし、それをほぼそのまま丸写しにしたことも判明した。なお、内容がまだ不明確なのは、「手続き」（手順、*procedures* 又は procedures)と「規則」(rules)である。その「手順」（手続き）だが、別のＪＩＳ規格では、「ある特定の目的のためにとるべき一連の動作の記述」(01.04.09)、とある[33]。その最新版ではより明確に、「手続き、サブルーチン」（procedure, subroutine）とは、「副プログラム、の一種であって、パラメタ授受の一環である場合を除き、データ値を返さないもの」(JIS X 0015:2002「情報処理用語（プログラム言語）」、15.06.11)

---

[28] ISO(1974)p.11

[29] ISO(1984)p.10

[30] ISO/IEC(1993)p.7

[31] 日本規格協会(1987)p.5

[32] 日本規格協会(1994)p.3。なお、ISO(/IEC)原文で斜字体、JIS 邦訳文で下線となっているのは、他の箇所で定義している用語を意味し、リファレンスを指示している。また(　)内の番号は用語の番号である。

[33] 情報処理学会編(1983)p.12(ＪＩＳ情報処理用語解説編集委員会編(1990)p.245 も、同文)。これは旧の JIS C 6230-1981 の定義である。

[34]、とある。つまり、「手続き」（手順）とは、プログラムが独立的なメイン・プログラムであるのに対して、非独立的な比較的小規模なサブ・プログラムのことである。そういう意味では、(広義の)プログラムの一部であり、技術的に詳細に立ち入る場合には識別する意義があるとしても、それ以外ではプログラムと分けて別個に挙示するほどのものではないことがわかる。「手続き」（手順）を包含したプログラムと解すれば済むことである。従って、「微妙な」差異のうち、手続き（手順）を挙示していない会計基準の定義に特に問題はないと言える。

「規則」に関しては、ＪＩＳ規格（ＩＳＯ/ＩＥＣ標準）でも全く説明がないので、何を指示しているか定かではないが、例えばプログラム標準（プログラムの作り方を規定したもの）や命名規則（ソフトウェア資源の命名の仕方を定めたもの）等を指示していると解するのが穏当であろう。これを挙示していない会計基準の定義が妥当か、それともＪＩＳ規格ないしそれに依拠した櫻井通晴編著(1993)の「実務指針〔案〕」の定義が妥当かということだが、これに関しては筆者が問題とすることに関説するところで取り上げた方がよいと考えるので、ここでは取り敢えず保留とする。

２．ソフトウェア会計基準のソフトウェア定義の問題点１――各種環境定義類の欠落

ソフトウェア会計基準におけるソフトウェア定義の第１の問題点は、重大な欠落があることである。それは、１（節）で取り上げた櫻井通晴編著(1993)において掲示された1)、2)、3)案（但し3)はカテゴリー・ミステイクにより成立不能なことは指摘済み）あるいはそれと会計基準との「差異」とは大きく性格の異なる欠落である。予め抗弁を封じておくと、基準（本体）では「プログラム等」（一 2）という「等」によって含みを持たせているので、それの解釈如何によって欠落はないと言い得る可能性がなくもないが、実務指針で敷衍的に①「プログラム」と②「関連文書」（６）としていることから、基準（本体）の「等」が「関連文書」を意味していることは明瞭であり、他の解釈の余地はない[35]。つまり、会計基準のソフトウェアの定義は「プログラム」と「関連文書」を意味しているのである。それ故に、筆者からすれば、重大な欠落があるのである。

（１）各種環境定義類とは何か

まず単純明快なことから始める。プログラムはそれ自体では動かない。何らかの手段により起動して始めて動くのである。例えば、メインフレーム系であれば、センターコンソール（マスターコンソール）からコマンド入力により起動させることが１つの方法である。ところが、センターコンソールから一連の情報を入力する煩雑さ等から臨時的な場合以外は、もう１つの方法であるＪＣＬ

[34] 日本規格協会(2013)p. 224。‘procedures’を、1987 年版(それ以前の旧版を含む)は「手順」としていたが、1994 年版は業界の通用語である「手続き」に改訳している。
[35] 基準(本体)と実務指針に齟齬があるという抗弁の可能性があるであろうか。もしあるならば、実務指針の改訂がなされて然るべきだが、この規定に関する改訂が全くなされていないことから、設定機関(企業会計基準委員会並びに公認会計士協会)では齟齬がないと認定していると言える。

（Job Control Language：Job 制御言語）による起動が一般的に行なわれるようになったのである。ＪＣＬで、プログラム、使用ファイル、ワークエリアのサイズ等を指定して、バッチ処理（Ｊｏｂ）の実行を指示するのである。つまり、プログラム単独では動かず、コンソールからのコマンド起動以外では、バッチ処理ではＪＣＬによる起動が不可欠なのである。しかも、ＪＣＬはプログラムではない。ＯＳからみれば、データ（制御情報）とも言えるものだが、アプリケーション・プログラムの入出力となるデータ（ソフトウェア会計基準でいうコンテンツ）ではない。そういう意味で、プログラムとは異なるソフトウェアの要素なのである。そうであるにも関わらず、ソフトウェア会計基準のソフトウェアの定義では、これらの各種環境定義類[36]が欠落しているのである[37]。

メインフレーム系のバッチ処理の起動（方法）というのは、１例である。いわゆるオープン系のバッチ処理起動ならば、ジョブ・スクリプト（Job Script）を必要とする。メインフレーム系のオンライン処理ならば、メッセージ起動でプログラムを動かすが、一般的にはＯＳの配下にオンライン・コントロール・ソフトを搭載し、それに対して予めトランザクション定義（オンライン定義）を行なっておくことで、メッセージを制御し、プログラムを動かすのである。ジョブ・スクリプトやトランザクション定義（オンライン定義）も、各種環境定義類である。

プログラムに関することは、プログラムの起動に限らない。ＤＢは、ＤＢＭＳに対してＤＢ定義（スキーマ定義）を予め行なっておかなければ、プログラムがＤＢに対するデータの処理を行なえないのである。このように各種環境定義類は、プログラムと共に、「コンピュータに一定の仕事を行わせるため」に必要不可欠のものである。しかも、繰り返すが、プログラムではない。従って、これをソフトウェアの定義に含めなければならない。主要な定義類として、具体的には、画面定義、帳票定義、ＤＢ定義（スキーマ定義）、トランザクション定義（オンライン定義）、ジョブ定義（ＪＣＬ、いわゆるオープン系ではジョブ・スクリプト）、その他（フレームワークへの定義等）がある。

（２）ソフトウェア会計基準にとっての不可欠性

ソフトウェア会計基準のソフトウェアの定義では、プログラム以外に、ソフトウェアに関するドキュメントもソフトウェアに含めているが、それはあくまで副次的ないし外延的なものであり、ハードウェア上で作動するわけではない。それに対して、各種環境定義類はまさしくハードウェア上で作動し、且つプログラムが作動するためには不可欠なものなのである。そうであるにもかかわらず、ドキュメントを含めておいて、各種環境定義類を欠落させていることは、余りにも均衡を欠いた定義ではないか。各種環境定義類を、正当に、ソフトウェアの定義に含めなければならない。

なお、ソフトウェア会計基準におけるソフトウェアの定義で、明示的ではないが、「プログラム」

---

[36] 実は、筆者の知る限り、ソフトウェア実務で通用している総称がないので、「環境定義」という用語を拡張的に複数形にして、そう称呼しておく。比較的近年では「設定ファイル」という用語がＷｅｂシステム等で使われているが、総称とするには限定的なので、「各種環境定義類」としておく。

[37] 日本基準だけではなく、アメリカ基準でも欠落していることは前掲のソフトウェアの定義(FASB２.)から明らかである。また、櫻井編著(1993)も、同類である。

という中にそれらも含まれているのだ、という見解が考えられるだろうか。しかし、各種環境定義類[38]は、ＪＩＳ規格で言うならば、「プログラムの外部環境」[39]の一部であり、プログラムとのインターフェースとも言うべきものである。従って、「プログラム」という中に含まれているというのは、無理である。プログラムとは明らかに異質なものであり、プログラムの一部又はサブ・カテゴリー又は亜種とすることは不適切であり、プログラム概念を混濁した不明瞭なものとしてしまうことになる。あくまでも、ソフトウェアという包括概念ないし上位概念の中に包摂することが適切であり、妥当である。

更に、プロセス等の観点から、議論を補強しておく。例えばＤＢ設計だが、開発プロジェクトの規模等により、必ずしも専任の担当者が配置されるとは限らないが、ＤＢを使用する限り、ＤＢ設計を行ない、ＤＢ設計書を作成する作業は必ず設定される。そして、実装のフェーズではプログラムを作成することと並行的にＤＢＭＳに対する実装レベルのＤＢ定義を行なう。プログラムかＤＢ定義かという形態的な違いはあるが、設計から実装までを通して、同等の且つ異なった作業（アクティビティ）があるのである。そして、開発が完了し、運用に入る際には、プログラムと併せて各種環境定義類がソフトウェア資源として運用部門に引き継がれるのである（受託開発であれば、当然に、納品・検収物件である）。

３．ソフトウェア会計基準のソフトウェア定義の問題点２――ドキュメントの規定の不備

ソフトウェア会計基準におけるソフトウェア定義の第２の問題点は、ドキュメント[40]の規定の不備である。大別すると、２つある。１つは、プログラムとドキュメントを同等に取り扱っていることである。もう１つは、「システム仕様書、フローチャート等の関連文書」（実務指針６）という簡略な例示をしているだけなので、対象範囲が不明確なことである。問題点１のような重大な欠落ではないが、具体的な会計実務の次元では、後述するように、少なからず会計処理に影響するのであり、軽視してよいことではない。

（１）ドキュメントの位置付け

ドキュメントをソフトウェアに含めることは、妥当であろうか。その前にまず、ドキュメントをプログラム並びに各種環境定義類と同格に位置付けることが妥当かどうかを問題にする。これはや

---

[38] ＪＩＳ規格（ISO/IEC 標準）には総括的な用語はないが、例えば「データベース スキーマ」（JIS X 0017:1997「情報処理用語（データベース）」、17.01.13）（日本規格協会（2013）P.242）のように、個々には記載がある。

[39] JIS X 0007:2001「情報処理用語―プログラミング」、07.11.07（日本規格協会（2013）P.115、下線は原文）。

[40] 現行の会計基準では「関連文書」としているが（櫻井編著（1993）では「関連書類」、ＪＩＳ規格では「関連文書」）、本章では引用を除き、ソフトウェア実務で最も馴染みのある「ドキュメント」という呼称を使用する。

はり不適切と言わざるを得ない。例えば1992年に刊行された大部の専門用語事典である『情報処理用語大事典』を会計基準設定の検討に際して参照していれば、次のような位置付けを知ることができたはずである。

ソフトウェアとは、「計算やデータ処理および計算機を動作させるためのプログラム、関連する手順、操作法などの総称。ハードウェアと対比される」[41]ものであり、

ソフトウェアシステムとは、「ソフトウェアを含んだトータルなシステムのこと。プログラム、プログラムの内容を説明したドキュメント等を含む」[42]ものであり、

ソフトウェアドキュメントとは、「ソフトウェア生産物の各種マニュアル類や仕様書などの総称。ユーザのためのマニュアルおよびソフトウェアの管理・保守のための仕様書の重要性がますます高まっており、これらドキュメントの作成技術の開発が急務となっている」[43]ものである。

ドキュメントは、狭義のソフトウェアには含まれないが、外延を拡げた「ソフトウェアシステム」には含まれる、と解することができたのではないだろうか。また、ドキュメントは、プロダクトであるソフトウェアを技術的に捉えれば不可欠の要素とは言えないが、ソフトウェアを開発する等のプロセスを考慮すると、必要性を増すものである、と解することもできたであろう。

もう１つ、格段に有力な参照すべき規範があったはずである。それは著作権法である。1986年にコンピュータ・プログラムを「著作物」に加える法改正が行なわれ、1987年にはデータベースを加える法改正が行なわれた[44]。ソフトウェア会計基準設定より、10年以上前の法改正であるから、時期的に参照することは十分に可能であった。著作権法では、プログラムは「定義」として「電子計算機を機能させて一の結果を得ることができるようにこれに対する指令を組み合わせたものとして表現したものをいう」（第二条十の二）のであり、「プログラムの著作物」（第十条九）という独立的な著作物として著作者の権利を有するものと規定されている。それに対して、「設計書やフローチャート等」はどうか。加戸守行(2013)によると、まず「定義」として「プログラム作成過程においては、その前段階としてシステム設計書、フローチャートが作成されるし、また、プログラムの利用についての説明書が作成される。これらもプログラムとともにコンピュータ・ソフトウェアといわれることがあるが、これらは、プログラムの作成、利用のための文書であって、電子計算機を作動させることを直接の目的とはしていないので、プログラムには当たらない」[45]としている。次に、「著作物（の例示）」として「広い意味でコンピュータ・ソフトウェアには、プログラムの他に、その作成過程における産物であるシステム設計書、フローチャートや、利用者のためのマニュアルが含まれるが、プログラム以外のものについては、通常の言語、数式、図形などで表現されており、人間

41 情報処理用語大事典編集委員会編(1992)p. 428
42 同書 p. 429
43 同書 p. 430
44 植松(2000)p. 2 参照
45 加戸(2013)p. 47。なお、同書は「です・ます」調の文体だが、引用時に「である」調に改変させて頂いた。

への伝達を目的とするものであるので、著作物であることについては従来から異論はない。また、権利の働き方の点でも何ら他の言語、図形等の著作物と区別すべき特殊な点はない。したがって、これらのソフトウェアについては、第１号〔引用者注：「小説、脚本、論文、講演その他の言語の著作物」〕の言語の著作物あるいは第６号の図形の著作物に当然含まれるものとして、特段の規定を置くこととしなかったものである」[46]、としている[47]。要するに、著作権法では、プログラムと関連ドキュメントは各々独立の「著作物」と規定されているということである[48]。もちろん、目的も性格も異なる規範であるから、例え同一対象に関してであっても、同一のカテゴライズをしなければならないわけではないであろう。それでも、先行的な規範がプログラムと設計書等の関連ドキュメントをどのように規定しているかは、参考になったのではないだろうか[49]。

---

[46] 同書 p. 127

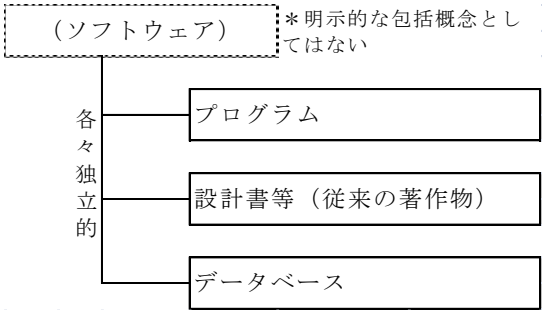
[47] こうした条文解釈は加戸だけではなく、植松(2000)でも、「設計書、仕様書、フローチャート等は(中略)プログラムではないから、「プログラムの著作物」ではない」(p. 36)。それらは「いずれも従来の形式の著作物となる」(p. 34)、ということである。「それではフローチャートの著作物とプログラムの著作物とはどういう関係に立つか、いいかえれば、プログラムの著作物はフローチャートの著作物の翻案か、という問題については、著作権法が「プログラムの著作物」として独立に規定している以上、プログラムはフローチャートとは別個独立の著作物になるのが原則と考えられる」(p. 35)、ということである(なお、この引用文では専ら「フローチャート」と言っているが、プログラム関連のドキュメントの代名詞と解せられる。また、同書からの引用も全て「である」調に改変させて頂いた)。

[48] なお、データベースに関しては、「定義」として「論文、数値、図形その他の情報の集合物であって、それらの情報を電子計算機を用いて検索することができるように体系的に構成したものをいう」(第二条十の三)のであり、「データベースの著作物」(第十二条の二)という独立的な著作物として著作者の権利を有するものと規定されている。これは、データベースをコンテンツとしてソフトウェアとは別個に取り扱うソフトウェア会計基準と符合する。

[49] 補足的に、注記とするが、ソフトウェア会計基準では殆ど論点になっていないが、著作権法では論点・争点となっていることを記しておきたい。①ソース・プログラムとオブジェクト・プログラムの取り扱い、並びに②モジュールの取り扱いである。①のソース・プログラムとオブジェクト・プログラムの取り扱いに関しては、「米国では、1980 年の著作権法改正後もオブジェクト・プログラムには著作物性がないという議論がなされていたが、わが国の著作権法は米国のように「書いたもの」とか、有体物に固定されることを著作物の成立要件としていないので、「プログラムの著作物」という規定をおいた以上、ソース・プログラムはもちろん、オブジェクト・プログラムも、「プログラムの著作物」として著作物になると考えられる」(植松(2000)pp. 34-35)、とのことである。そして、「ソース・プログラムをコンパイラによってオブジェクト・プログラムに変換すると、高級言語から機械語またはそれに近い言語に変換されるので、著作権法上は翻訳物となるが、変換は機械的に行なわれ、その過程に人間の創作活動がないため新たな著作者は生ぜず、結局ソース・プログラムの著作者がオブジェクト・プログラムについても唯一の権利者ということになる」(同書 p. 43)、とのことである(言うまでもないことだが、プログラマーがプログラムの権利者に相即的になるという意味ではなく、業務システムに関しては一般的には開発した企業が(委託を含む)権利者となる)。会計基準では、特に意識することなく、包括的にプログラムないしソフトウェアと見做しており、それで差し支えない。②のモジュールの取り扱いに関しては、既述したＪＩＳ規格(ＩＳＯ/ＩＥＣ標準)の「手続き」(手順)に大凡相当することだが、「プログラムである場合と、ない場合と両方がありうるといえるであろう。本来モジュールという言葉は技術用語であって、法律用語ではないから、それが著作権法上のプログラムに該当するか否かは、個々に法律的な見地から検討しなければならず、一概にはいえないわけである。しかし、かなり多くの場合、モジュール単位で法律的にもプログラムといえると予想される」(同書 p. 35)、とのことである。会計基準ではプログラムの大小

念のため、著作権法の改正は確かにソフトウェア会計基準設定前だが、このような法解釈が定着したのは時期的にそうとは言えないのではないか、という疑義が起こり得るので（引照文献が遥かに後のものでもあるので)、基準設定前の文献で確認をしておく。東季彦監修(1996)に拠ると、「プログラムを設計する段階で書かれる設計書、仕様書、フローチャートなどはいずれも従来の著作物の要件を満たせば、著作物として保護を受け得るものである。これらは、いわば機械に対する設計図に当たるもので、理論としてはこれらに著作物性があるのであってこれらによって生まれるプログラムは著作物ではないということも可能である。しかし、著作権法はプログラムを独自の著作物として規定し、かつ「書いたもの」であるとか「有体物に固定する」ことを要件としていないので、プログラムは設計書などの有無にかかわらず著作物になると考えることになる。／すると、これらの設計書などはプログラムとは別個独立に著作物となることになり、プログラムが設計書などに従って初心者のプログラマーでも当然に完成しうるものであるような場合には創作性がない、あるいは設計書などの二次的著作物もしくは翻案であると考えることになる。しかし、設計書などとプログラムは表現形式が異なり、また著作権法がプログラムを独自の著作物と認めた趣旨から考えれば、プログラムが当然に設計書などの複製物になると考えるべきではない」[50]、ということである。時期的に解釈のブレや変化はなく、ソフトウェア会計基準設定前後を通じて、著作権法上の捉え方は定着していたと言えるであろう。なお、参考までに著作権法上のソフトウェアの定義を図示すると、次のようになるかと思う。

図表B－１-1　著作権法上のソフトウェアの定義



（出典：筆者作成）

ソフトウェア開発では、一般的には、設計を行ない、それを設計書として纏め上げ、それに基づいてプログラム並びに実装レベルの各種環境定義類を作る、という作業の進め方をする。そういう意味では、プログラム並びに各種環境定義類とドキュメントはかなり不可分的なものと言えるであろう。但し、不可分的とは言い切れない事情も少なからずあることを総合的に捉えないと、一面的な片寄った捉え方になる。開発スケジュールがタイトになると、ドキュメントは後回しにして、「物

は特に問題にせず、プログラムと見做しており、それで差し支えない。なお、①・②とも、あくまで著作権法上の著作物並びに著作者の権利に関わる規定としてのことであり、会計基準に考慮を求めるようなものではないが、他分野の議論として記しておく。

[50] 東監修(1996)pp. 257-258。なお、同書は尾中・久々湊・千野・清水の「共同執筆」(p. ｖ)であるが、引用箇所は清水幸雄が「執筆担当」(p. 326)である。

作り」（プログラム作成）を優先することは、開発プロジェクトにおいてありふれた対応である。あるいは、保守において、ドキュメントが最新状態に更新されていない等のために、ドキュメントに依拠できず（あるいはせずに）、プログラムを直接解読して仕様を確認し作業を行なうことはありふれたことである。いずれも、是非を問題にするならば、決して「是」とは言えないことだが、例外的と見做すほど、少ないわけではない。また、これらはいずれもネガティブに捉えてのことだが、アジャイル型開発では発想の転換がなされ、もっとポジティブに「動くソフトウェア」を作ることを優先し、ドキュメントは最低限のものしか作らないことをモットーとしている。これらは、ドキュメントの必要性や意義を否定するものでは些かもないが、ドキュメント→実装という順序性や作業における密接不可分性はある範囲の条件下では成立しているが、常態ではないことを如実に示している。

こうしたことから、ドキュメントはやはりソフトウェアそのものとは言い難く、狭義のソフトウェアの定義には入らないとするのが妥当である。ドキュメントは、実行時にハードウェア上でプログラム並びに各種環境定義類と一緒に作動するわけではないし、開発時等においてもドキュメントがなければプログラム並びに各種環境定義類が作れないというほど必要不可欠のものではないからである。従って、プログラム並びに各種環境定義類と同格に位置付けることは妥当ではない。

次に、ソフトウェアの定義に含めることが妥当かどうかということだが、その必要性により広義のソフトウェアとして含めることは差し支えないと考える。大規模開発において全般的にはドキュメント抜きの開発はまずあり得ない（例え部分的には上記に例示したような事態になったり、あるいは最終的にフィックスした正式ドキュメントは後付けで整備されるとしてもである）。あるいは、今日では当たり前に行なわれているオフショア開発は、ドキュメントなしでは成り立たないであろう。但し、あくまで副次的なものであって、ソフトウェアの主たる要素であるプログラム並びに各種環境定義類と同等のものではないことは弁えておかなければならない。このことは、具体的な会計処理に直接影響することではないが、識別しておいた方が良いことである。

（2）ドキュメントの対象範囲

ドキュメントに関するもう１つの問題は、ソフトウェアとして定義するドキュメントとは何なのか、その対象範囲はどこまでなのか、ソフトウェア会計基準では不明確なことである。基準（本体）は「プログラム等」（一2）としているだけであるが、実務指針では「システム仕様書、フローチャート等の関連文書」（6）、としている。櫻井通晴編著(1993)では「関連文書には、システム仕様書、フローチャート、運用マニュアルなどが含まれる」[51]と例示している。しかし両者とも、カテゴリー・ミステイクに陥っている。「システム仕様書」や「運用マニュアル」は、ドキュメントの種類である。それに対して、「フローチャート」は、ドキュメントの種類ではない。フローチャートは、表

[51] ソフトウェアの資産性に関する検討委員会・委員長櫻井通晴、櫻井編著(1993)p. 27

現形式であり、記法である。「システム仕様書」を文章記述と併せて、その一部ないし大半をフローチャートで記述する場合もあるだろうし、「運用マニュアル」も同様である。「システム仕様書」や「運用マニュアル」と別に、独立的なドキュメントの種類として「フローチャート」があるわけではないのである。

そして、記法ならば、フローチャート以外に、ＨＩＰＯ（Hierarchy with Input-Process-Output）[52]、ＤＦＤ（Data Flow Diagram）、デシジョン・テーブル（decision table：決定表）、ＣＲＵＤ図（CRUD matrix）[53]等があり、今日ではUML（Unifiied Modeling Language：統一モデリング言語）やマインド・マップ（mind map）もある。これらは、特定のドキュメントの種類を意味するわけではなく、様々な種類のドキュメントで記法として使われるものである。要は、性格の異なるものが列記されている。言うまでもないことながら、例え例示であっても、同じ性格のものを並記しなければならないのであり、ここではドキュメントの種類を挙示すべきなのである[54]。

それでは、どのような種類のドキュメントをソフトウェア・ドキュメントとすることが妥当であろうか。

図表Ｂ-１-2　ソフトウェア・ドキュメント一覧

大フェーズ	正式ドキュメント	テンポラリー・ドキュメント
企 画	ＲＦＰ （提案書） 企画書 各種標準 各種規程	各種検討資料
開 発	開発計画書 設計書 テスト計画書 テスト仕様書 （テスト結果報告書） （各種テスト結果）	各種検討資料 各種管理資料 進捗報告書 レビュー議事録 問題票
運 用	環境構築計画書 環境構築手順書 （環境構築結果報告書） 移行計画書 移行手順書 （移行結果報告書） 運用マニュアル 操作マニュアル	年間運用計画書 資源管理台帳 ＪＯＢ依頼書 ＪＯＢ手順書 （ＪＯＢ実行結果報告書） オペレーション日誌 レビュー議事録
保 守	設計書（更新） 差分設計書 テスト仕様書 （テスト結果報告書） （各種テスト結果）	各種検討資料 各種管理資料 進捗報告書 レビュー議事録 問題票
廃 棄	廃棄計画書 廃棄手順書 （廃棄結果報告書）	レビュー議事録

（出典：筆者作成）

[52] Hierarchical Input-Process-Output、Hierarchy plus Input Process Output とも表記する。
[53] データ(エンティティ)が、どの機能で作成(create)、参照(read)、更新(update)、削除(delete)されるかをマトリクス形式で表現したものである。
[54] 情報処理用語大事典編集委員会編(1992)(p.430)は、カテゴリー・ミステイクに陥ってはいないが、挙示しているドキュメントの種類は大差なく、貧弱で、参考にならない。

ソフトウェア基礎論第２章で主題的に考察するソフトウェア・ライフサイクルを先取り的に使用し、それに沿った形で一覧に纏めた。個々のドキュメントの説明は省略するが、留意すべき幾つかのことを箇条書きの形で簡潔に説明する。

①ドキュメント名は筆者に馴染みのあるものとしたが、例えば設計書は仕様書と言い換えても同じことである。

②例えば設計書は、更にフェーズ毎に基本設計書、概要設計書、詳細設計書と分かれるが、包括的に設計書とした（テスト仕様書も同様である）。

③「結果報告書」をカッコで括ったのは、必ず作成するとは限らないし、あるいは例えばテスト仕様書に結果を記載する欄を予め設定し（空白で）、テスト実行後に記載するという様式もあることを意味する。類似的なことでは、「計画書」と「手順書」は一応別立てとしたが、両方の内容を一本化（一体化）して、計画書とする場合もある。

④「各種テスト結果」をカッコで括ったのは、それを必ず正式ドキュメントとするとは限らないし、またテスト結果の中にはドキュメントではなく、テスト実行結果のデータ・ファイルといったものもあるからである。

⑤保守における「設計書」を（更新）と段下げで差分設計書としたのは、本体の設計書を保守の都度更新する場合と、本体は（初期）開発のままとし、変更した内容（追加、削除も含む）だけを「差分」という形で作成する方式があり、何れを採用するかは一律ではないからである。

⑥企画における提案書をカッコで括ったのは、ＲＦＰや企画書は自社で作成するが（委託を含む）、提案書は必ずソフトウェア業者側が作成するものなので、別扱いした方がよいので、そのようにした（先取り的なことを言えば、ドキュメントを資産計上するとした場合、自社が作成したものでないドキュメントを自社の資産とするか否かは他のドキュメント以上に熟慮しなければならないからである、詳細は本論で取り上げる）。

⑦最後になったが、もっと大きな括りとして、正式ドキュメントとテンポラリー・ドキュメントに区分したが、正式ドキュメントは一般的には廃棄まで保管するが（廃棄後も保存する場合もあるし、少なくとも長期に保管することは間違いない）、それに対してテンポラリー・ドキュメントは一般的には長期に保管せず、作成目的が達成されれば用済みとなるものである。例えば、設計時の各種検討資料は仕様が確定し、設計書にそれを記載すれば、用済みとなる。但し、各種代替案等を記載した検討資料が特定の案が採択されても、資料として独自の意義あるいは有用性を持しており、その一部が設計書に盛り込まれても、将来的な利用価値を考慮して保管するといった可能性もある。そうした意味で、テンポラリー・ドキュメントの中にも、正式ドキュメントと同等の取り扱いをすべきものもあり得る。それでも、基本的には、正式ドキュメントが副次的なソフトウェアとする範囲である、と筆者は考える。

なお、仮にこの一覧が十分に網羅的ではないとしても、これだけの挙示をしておけば、追加対象とすべきか否かの判断を行なう目安（規準）には十分になり得るであろう（例示というのは、そう

でなければならない)。

４．ソフトウェア会計基準のソフトウェア定義の問題点３――その他

ソフトウェア会計基準の問題点として、各種環境定義類の欠落、ドキュメントの規定の不備を取り上げてきたが、それ以外のややマイナーな問題を幾つか取り上げる。

（1）構造体として捉えるべきこと

ソフトウェア会計基準では、単に「プログラム」というだけで（「関連文書」に関しても同様)、構成や構造として捉えていない。独立的な最小のソフトウェアとして１つのモジュール（プログラムの最小単位）ということもあり得るが（組み込み系等)、今日では殆どのソフトウェアはプログラムに限定しても複数のモジュールで構成されている。そして、それらは単なる集合ではなく、順序性や階層や連係的な関係として構成される構造体[55]として存立しており、作動するのである。従って、そのように定義しなければ、不完全である。各種環境定義類にしても、ドキュメントにしても、同様である。例えば、Ｊｏｂネットワークを構成する定義類である、あるいは設計書において基本設計書―概要設計書―詳細設計書という階層的な構成になっている、というようなことである。

このことは、具体的な会計処理に直接影響することではないが、ソフトウェアの実態を捕捉するということでは必要なことである。

（2）「規則」の取り扱い

保留にしておいた「規則」の取り扱いを、これまでの行論を踏まえて、ここで取り上げる。結論を予め提示すれば、プログラムやドキュメントと同列的に独立的に取り扱うのではなく、ドキュメントの一部として含めればよいと考える。理由は、次の通りである。規則というのは、範囲が曖昧であり、広義には不文律や慣行律といったものも含まれ得る。しかし、それらが形成されるプロセスを考慮すると、プログラムやドキュメントと大きく異なる。プログラム等は、明示的な計画を立て、目的意識的な具体的な作業として遂行し、その結果として具体的な成果物となるものである。それに対して、「規則」は必ずしもそのような形で形成されるとは限らない。プログラム等と同様に具体的な作業の結果として形になるのであれば、それは必ずドキュメントという形態を取ることになる。それ故、「規則」という範囲が曖昧なものはソフトウェアとは見做さない。より具体的な成果物として、例えば「プログラム標準」といった名称の標準書に成形された場合には、ソフトウェア・

---

[55] 「構造体」は、ソフトウェア用語としては、Ｃ言語系の派生的データ型の１種の名称であるが、それを「流用」した。原語はstructureであり、一般的には「構造」と訳されているが、プログラミング言語の訳語ではデータ「型」を意識して、「構造体」とされている。なお、「構成体」でもよい。

ドキュメントとしてソフトウェアと見做すことにする。そのような取り扱いとする。

もっと外延的な例えば開発部門や運用部門の業務分掌や職務分掌といったものであれば、それは組織全体の業務分掌や職務分掌の一部であり、独立的に取り出す謂われは乏しいし、ましてそれをソフトウェアと看做すことは拡大解釈に過ぎると言うべきである。ソフトウェアのライフサイクルに沿った「規則」に限定すべきであるが、明示的なものであれば、それは必ずドキュメントという形態を取るから、取り立てて「規則」という性格付けを意識する必要はなく、ソフトウェア・ドキュメントとして具体的な作業の成果物として出来上がり、そして有意義且つ有用であるかを判断すればよい、と考える。

（3）コンテンツとの境界

会計基準におけるコンテンツの取り扱いは、基本的にはソフトウェアとは看做さないが、ソフトウェアと「一体不可分」の場合にはソフトウェアと看做すというものである。一見明解なようだが、「一体不可分」の意味がやや曖昧である。「コンテンツはその処理対象となる情報の内容である」（実務指針 29）にしても、技術的には截然と区別できるかどうか、やや疑問である。ソフトウェアの定義として、「ソフトウェアの中心となるものはプログラムであるが、関連するデータも含まれる（フォンノイマン方式計算機ではプログラムとデータは本来区別が難しい〔引用者注：そして今でも大半のコンピュータはフォン・ノイマン式である〕）」[56]、というものもあるくらいである。また、それまでのプログラミングは出来る限りデータと処理を分離する方向で進んできたが、オブジェクト指向ではある意味逆行するように、クラス（一般的には最小のプログラム単位と解してよい）をデータメンバーとメソッド（処理）で構成するようにしている。こうしたことからも、今日的な観点で再考の必要があると考える。

簡潔に言えば、筆者はデータ（コンテンツ）の性格と併せて、ここでもプロセスという視点を追加的に導入して捕捉すればよいと考える。利用の段階で生成（入出力）されるものか、それとも開発・保守・運用において提供側が生成するものか、という識別である。しかも、例えばデータ・テーブルの事前設定（「例えば、財務会計ソフトの科目マスターの設定」（実務指針 38(2)）が該当する）は利用者の代行作業として行なうものなので、例え提供側が実際の作業を遂行してもソフトウェアとは見做さないデータであり、旧システムのデータを新システムに移行することも、提供側が遂行するとしても、元々利用者が作成した元データの変換等であるから、同様である、という分別を併せて行なう必要がある。

もう１つ今日的に考慮する点は、ソフトウェアにおいてプログラミング言語とは異なる「ソフトウェア言語」[57]が出現してきていることである。古くはＪＣＬがそうであるが、呼称に Language が

[56] オーム社編(2001)p. 385

[57] このような用語が通用しているわけではなく、筆者が仮称しているものである。しかし、略号の末尾に「L」の付いた言語類が多く出現してきており、筆者の知る限り纏まった研究は殆どなされ

付いていることが１つの目安である。但し、記法であるＵＭＬ等もあるから、要注意ではある。例えばＷｅｂシステムで多用されているＨＴＭＬ（HyperText Markup Language）やＸＭＬ（Extensible Markup Language）が、そうである。これらが使われている部分は、データに近いが、提供側が作成する部分であり、ソフトウェアと見做すのが妥当である[58]。

今日的な事態として、ソフトウェアとコンテンツ（データ）の境界は変動しており、個々に精査し、ソフトウェアか否かの判断を行なう必要があると考える。その際に、上記のような追加的な判断規準が有用であろう。そして、現行会計基準と同様、コンテンツ（データ）は基本的にはソフトウェアとは見做さない、ということでよいと考える。

関連して、データベースの取り扱いに言及する。まずデータベースが多義的なので、対象を特定する必要がある。データベースは、大別すると、①データを格納する、一般的なファイル形式とは異なる特有の器、②ＤＢＭＳというＤＢを管理するソフトウェア、③あるカテゴリーの多大なデータの集積、という意味に分かれるが、国民経済計算や著作権法で対象としているのは③である。③が固有の価値を有しており、「無形固定資産」と見做すことに特段の異議はないが、上記のコンテンツ（データ）の取り扱いの延長上で取り扱えばよい、と考える。ソフトウェアの一部ではないし、同等のものと見做すことも適切ではない。ＤＢＭＳというソフトウェアによって、処理することが可能な依存物に留まるのである。

５．ソフトウェア会計基準におけるソフトウェア定義改訂案

これまでの考察を纏める形で、ソフトウェアの定義に関する筆者の改訂案を提示する。対比的に、現行会計基準の定義をも図示する。また、ソフトウェア定義の改訂案が会計にとってはどのような意義があるのかを説示する。

（1）ソフトウェア定義改訂案の提示

ソフトウェアの定義は、次の通りである。

「ソフトウェアとは、コンピュータを機能させるように指令を組み合わせて表現したプログラム並びに各種環境定義類の構造体、副次的なものとして企画、開発、保守、運用、廃棄の各種関連ドキュメントをいう」。

実務指針では、「ソフトウェアとは、コンピュータ・ソフトウェアをいい、その範囲を次の通りと

ていないように視えるし、興味深いテーマではあるが、会計と関わるのは今のところ本文で取り上げた範囲に留まるように思われる。

[58] 例えばＰＤＦ(Portable Document Format)は、作成・閲覧するソフトはソフトウェアだが、それによって作成・閲覧するものはデータであり、ソフトウェアとは見做せない。会計で馴染みのあるＸＢＲＬ(eXtensible Business Reporting Language)も、同様に、作成・閲覧するものはデータであり、ソフトウェアではない。

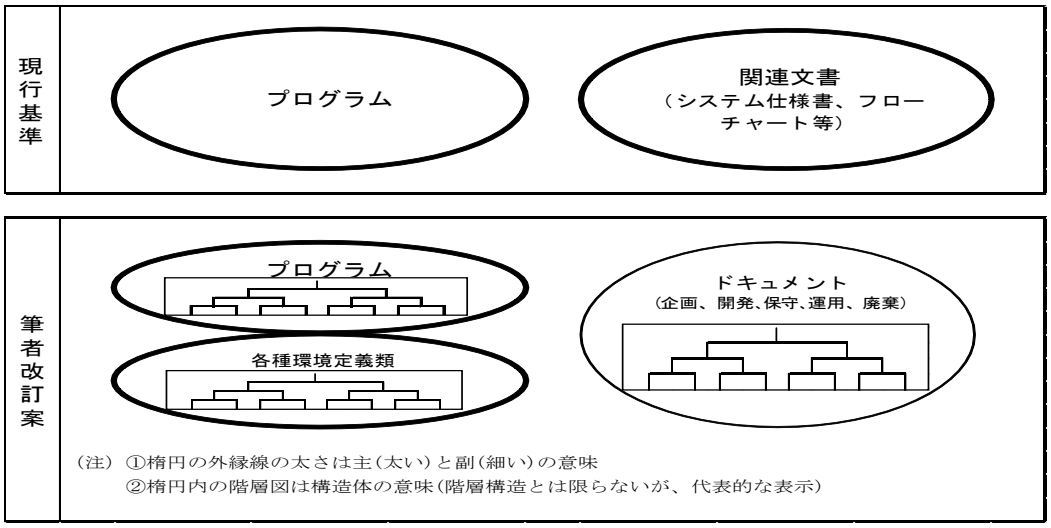
する。

① コンピュータに一定の仕事を行わせるためのプログラム並びに各種環境定義類の構造体

② 副次的に、企画、開発、保守、運用、廃棄の各種関連ドキュメント」、

とする。「コンピュータを機能させるように指令を組み合わせて表現した」とか、「コンピュータに一定の仕事を行わせる」といった表現は、如何にも旧式な感が否めないが、敢えてそのままにした（替えるとすれば、「情報処理」といった表現にするのが無難であろう）。理由は、多少は装いを新たにすることが出来なくはないが、こうしたことの表現に十分と言えるようなものを考案することは存外に難しいからである。また、今日ではネットワークがコンピュータ・システムの重要な構成要素なので、それは是非とも明示的に加えるべきだという意見もあり得るであろうが（ＩＴに替えてＩＣＴ（Information and Communication Technology）とするように）、ハードウェア並びにソフトウェアの各々に含まれると解すれば済むことなので、明示的には加えなかった。コンテンツの取り扱いに関しては、現行基準の定義でも解釈的な幅を持たせており、定義としてはそのままとした。なお、若干の注釈を施せば、１つはドキュメントには「構造体」という規定を明示しなかったが、「関連」という規定にそれが含意されると解し、煩雑さを避ける意味で、明記しなかった。副次的というのは、順序性や原本とそれからの派生といった意味はなく、「主たる」に対する「副たる」（あるいは従たる）という意味としている。企画から廃棄までを明記したのは、第２章で詳論するプロセスとしてのソフトウェア・ライフサイクルを意識してのことである。

図表Ｂ－１－３　ソフトウェアの定義



（出典：現行基準の内容は基準参照、図化並びにそれ以外は筆者作成）

（２）ソフトウェア定義改訂案の会計的含意

これまで会計の論文としては異例なほどの紙幅を費やして、ソフトウェアの定義を考察してきた。行論の途上で必要な範囲では会計的な含意に言及してきたつもりだが、必ずしも十分ではなかったであろう。このような議論が、会計的に意味があるのか、疑問を呈せられるかもしれない。しかし、

筆者には即応的に会計的な意義を唱えるのではなく、一旦は対象たるソフトウェアに沈潜し、これまでの浅薄なソフトウェア理解を超えて、ソフトウェアを必要十分な深度と拡度で捕捉し、そうした基礎の上でソフトウェア会計を考察することが、後進の会計学徒がソフトウェア会計研究を前進させるために不可欠な基礎作業と考えているからである（従って、本論ではなく、その前提とも言うべきソフトウェア基礎論という位置付けとした)。

ソフトウェアの定義を上記のように改訂すると、明確に前景化してくることがある。ソフトウェアは現行基準でも資産要件を充足すれば、資産計上することになっているが、「研究開発」という大枠の中で限定的であり（そもそもが「研究開発費等に係る会計基準」であることは言うまでもなく、ソフトウェアに係る独立的な会計基準ではない)、資産範囲も未だに狭い。それが、ソフトウェアの定義を改訂することによって、資産化の範囲はかなり拡大するのである。これまで専ら費用処理とされてきた企画や運用改善が資産化の可能性を有することになる。具体的に立ち入ることは本論各章においてであるが、その基礎固めとなるのである。

補遺　「ソフトウェア工学」における１つのソフトウェア定義

ソフトウェア基礎論本文では敢えて取り上げなかったが、もう１つ「有力な」ソフトウェアの定義がある。微妙な問題を孕んでいるので、主題的な考察に混乱を招くのを避けるため取り上げなかったのだが、ソフトウェア会計基準のソフトウェアの定義の「援軍」ともなり得るものなので、公平を期する意味でも、補遺として取り上げておいた方がよいかと思う。

監訳者によれば「欧米をはじめ世界各国で用いられており、定評の高い教科書」[59]とのことであるロジャー・プレスマン(2005)における、ソフトウェアの定義である。それは、

(1)実行されることによって必要な特性、機能と性能を提供する命令語群（コンピュータプログラム)

(2)プログラムが十分に情報を扱えるようにするためのデータ構造

(3)プログラムの操作や使用法を記述したドキュメント[60]

というものであり、これまでの定義と異なるのは、３要素を同列的に取り扱っていることである。(3)は、ソフトウェア会計基準がドキュメントを同列に取り扱っていることを支持するソフトウェア分野における定義と言うことができるであろう。それに対して、(2)は筆者の各種環境定義類というソフトウェア定義の重要な要素を明示的に支持する定義と言えなくもない。それ以上の詳細な説明がないので、「データ構造」がデータ定義を意味しているとは断定できないが、データとせずに、「データ構造」としているのは、単なるデータ類（コンテンツ）のことではないと解せられる。仮に筆者の意味でのデータ定義のことだとしても、それに限定しているということでは、環境定義の総体

[59] Pressman(2005)邦訳 p. vii
[60] 同書 p. 3

を定義に加えているわけではないので、筆者の定義と同等ではない。いずれにせよ、(2)の要素を含めた3要素の定義という点では、ソフトウェア会計基準の定義を全面的に支持するものでもないし、3要素を同列的に並べているという点では筆者の定義とも異なるものである。

プレスマンは、定義を提示しておきながら、続けて、「定義は他にもあるが、特に形式ばった定義は必要ないだろう」[61]と言って、定義の詳細な議論に踏み込もうとしていない。この点、プレスマンは少々見当違いの振る舞いをしている。ソフトウェア自体がある意味では各種定義の塊といってもよいものであり、しかも定義にごく微細な瑕疵があっても、バグとなり、正常に動作しないのがソフトウェアではないのか。それを軽視するようでは、「実践的アプローチ」と言いながら、有用性に悖るのではないか。ソフトウェアにとって、定義には徹底的に拘らなければならない。

3要素の並列的な定義ということでは会計基準の定義と全同ではないが、「ソフトウェア工学」におけるソフトウェアの定義で、プログラムと関連ドキュメントが並列的な要素となっているという意味では、会計基準にとって有力な「援軍」と言うべきで（専門分野の「お墨付き」）、定義が取り立てて不備や不完全さを指弾される謂われはないと言えるのではないか。ところが、そうではないのである。「ソフトウェア工学」は、ソフトウェアに関して、余りにも未熟なのである[62]。事情に通じていないと、誤解を招くかもしれないが、ソフトウェアは一般的には工学的な分野と思われているであろうし、そして一応間違いではないが[63]、そうであるからといって、「ソフトウェア工学」を基礎にして構築されているわけではないのである。「ソフトウェア工学」は、公式的には1968年に初めて提唱され[64]、徐々に研究が進められてきたが、有態に言って、顕著な成果は余り出ておらず、

---

[61] 同書邦訳p.3

[62] 「定評の高い教科書」と称されるPressman(2005)は、筆者が容易に指摘できるような誤りが散見される程度の水準の「教科書」である。例えば、「アルファテスト(alpha testing)は、開発者のもとでユーザ自身が行うテストである」とか、「ベータテスト(beta testing)は、エンドユーザのもとで行われるテストである」(p.266)などという、ソフトウェア業界の「常識」をも弁えぬ出鱈目な説明を行なっている等である。

[63] 留保的な言い方をしているのは、プロダクトとしてのソフトウェアを工学的と見做すのは間違いではないが、プロセスに関しては工学的アプローチは一面的な有効性しかないのである。ソフトウェア開発の失敗が目立って減りもせず、相変わらず問題を抱えていることは、それを明かして余りある。例えば、比較的近年、要求工学(requirement engineering)という分野が形成されつつあり、遅ればせながら要求定義(要件定義)の重要性とそれには適合的な方法を必要とするということが共通認識となりつつあることは歓迎すべきことであるが、工学的アプローチだけで要求定義を十全且つ適宜に成案化することは無理である。要求工学の実際の内容には、既に工学的ではない内容も含まれており、それをなお「工学」と称している点で、自覚的ではなく、自ずと限界を示している。それを自覚的に捉え直し、工学的アプローチを含む、更に統合的なアプローチを採ることで、初めて発展する可能性が切り開かれるのである。ソフトウェアに関わるプロセスは、工学だけで解決を見い出すことはできないのである。

[64] ＮＡＴＯ(North Atlantic Treaty Organization：北大西洋条約機構)のSoftware Engineering Conferenceにおいてである。なお、ソフトウェア工学を広義に捉えれば、実務において集積されてきた開発技法等を全て包含したものを意味するが、カッコ付きの「ソフトウェア工学」はもっと限定した狭義のもので、ＮＡＴＯでの提唱を起点とし、主として大学等で「研究」が行なわれてきたものを指している。広狭の区別をしないと、議論が混線する。

ソフトウェア実務への貢献はそれほど多くない。従って、ソフトウェアを専門分野としている「ソフトウェア工学」の高名な「教科書」に書かれているからといって、自らが実態に即して検証することなく、それに依拠することは、ただの権威主義に過ぎない。ソフトウェア分野の「通念」にしても、定義が実務の何かに影響することはないから（例えば契約条項はもっと具体的に定めるから「定義」が係争事項になることはない等）、無反省のまま通用しているだけのことである。

ケーパーズ・ジョーンズの近著[65]は、「ソフトウェア工学はその名に値せず、ソフトウェア開発は単なる工芸であって真の職業ではないのである」[66]という思いが基調となって全編に貫かれており、邦訳の副題である「ソフトウェア工学の真の工学への発展をめざして」はさすがにその真意（意図）を的確に汲み取ったものとなっている。これが、ソフトウェアに精通した者の真っ当な現状認識である。こうしたことから、例えソフトウェア分野で「通用している」からといって、鵜呑みにはできないのであり、反省的に、批判的に実態に迫らなければならないのである。

[65] Jones（2010）邦訳
[66] 同書邦訳 p. ｘｘvii

ソフトウェア基礎論

第２章　ソフトウェア・ライフサイクル

１．各種「標準」におけるソフトウェア・ライフサイクル

（1）ＪＩＰＤＥＣ「システム管理基準」

（2）ＩＰＡ「共通フレーム」

（3）ISO/IEC(JIS)「ソフトウェアライフサイクルプロセス」

２．ソフトウェア・ライフサイクルの概容

（1）ソフトウェア・ライフサイクルの全体像

（2）個々の大フェーズの概容

図表Ｂ−２−1　ソフトウェア・ライフサイクル

図表Ｂ−２−2　個々の大フェーズの概容

第2章　ソフトウェア・ライフサイクル

１．各種「標準」におけるソフトウェア・ライフサイクル

ソフトウェア基礎論第１章ではプロダクトとしてのソフトウェアを捕捉する基軸の基礎であるソフトウェアの定義を考察したが、それに引き続きソフトウェア基礎論第２章ではソフトウェアのプロセスを捕捉する基軸となるソフトウェア・ライフサイクルを主題的に考察する。

ソフトウェア・ライフサイクルとは、文字通りに、ソフトウェアの「一生」のことである。企画において開発が決定すれば、開発が行なわれ、開発が完了すれば運用に入り、並行的に保守が行なわれ、それらがある期間継続し、陳腐化等により運用を終えることになれば、廃棄される。購入する場合には基本的には開発は行なわれないが（カスタマイズを除き）、それは既に提供元で開発済みだからである。フェーズの分け方等は多少の違いがあるとしても、ライフサイクルという捉え方はソフトウェア業界（実務）では当たり前のことになっている。

ソフトウェア・ライフサイクルに関する標準的なものを幾つか取り上げる。

（1）ＪＩＰＤＥＣ「システム管理基準」

ＪＩＰＤＥＣ（経済産業省）の「システム管理基準」は、2004 年に策定されたが（追補版は 2007 年[67]）、次のような構成になっており、ソフトウェア・ライフサイクルに沿った構成である。

大項目は、

「Ⅰ．情報戦略」

「Ⅱ．企画業務」

「Ⅲ．開発業務」

「Ⅳ．運用業務」

「Ⅴ．保守業務」

「Ⅵ．共通業務」

という６つであるが[68]、「Ⅰ．情報戦略」は企業あるいは「組織体の情報システム」[69]全体に関わることであり、「Ⅵ．共通業務」は各大項目に共通的なことを一括的に取り出したものであり、「Ⅱ．企画業務」から「Ⅴ．保守業務」までが直接的にライフサイクルに関わるものである。「業務」と称呼しているが、筆者のいう大フェーズ[70]に相当する。廃棄は、大項目にはなっておらず、独立的で

---

[67] 追補版は、その名の通り、副題とする「財務報告に係るＩＴ統制ガイダンス」を追加したものであり（経済産業省(2007)参照）、元の「平成 16 年基準策定版」に対する改訂はないので、ここでは取り上げない。

[68] 経済産業省商務情報政策局(2005)P. 39-502。なお、個別項目は 287 項目である（同書 p. 4）。

[69] 同書 p. 3

[70] 大フェーズ（large phase）と称呼するのは、開発（内）のフェーズ（phase、工程分割）がより馴染み

はないが、「Ｖ．保守業務」の中で「６．情報システムの廃棄」として位置付けられている[71]。しかし、この位置付けは、筆者には賛同できない。確かに管理項目として「ユーザ、運用及び保守の責任者の承認を得て廃棄すること」（Ｖ．６．(1)）[72]とあるように、廃棄はユーザ、運用、保守の各部門に関係しているが、(本番) 運用を行なっている限り、資源等の管理を主として管掌しているのは運用部門であり、(本番)環境へのアクセスは保守部門には通常認可されていないことからしても、廃棄作業は運用部門主導で行なうものである。従って、独立的な大項目とするべきであるが、そうしない場合に次善の策としては「運用業務」の一環として位置付けるべきである、と筆者は考える。このように、廃棄の取り扱いは支持できないが、ソフトウェア・ライフサイクルにほぼ沿った構成となっており、ライフサイクルを視野に入れた管理が定式化されていることは確かである[73]。

（2）ＩＰＡ「共通フレーム」

ＩＰＡの「共通フレーム」は、1994 年に旧通産省が策定し、1998 年に改訂が行なわれ、ＩＰＡが 2004 年に継承し、2007 年に改訂を行ない（2009 年に第 2 版改訂）、最新版が 2013 年版、という経緯で今日に到っている。1998 年版以降は、国際標準「ソフトウェアライフサイクルプロセス」ISO/IEC 12207（JIS X 0160）をベースに作成され、日本独自の内容を追加したものである。

最新の 2013 年版でみると、プロセスは、

「1. 合意プロセス」

「2. テクニカルプロセス」

「3. 運用・サービスプロセス」

「4. 支援プロセス」

「5. プロジェクトプロセス」

「6. 組織のプロジェクトイネーブリングプロセス」[74]

「7. プロセスレビュー」

「8. テーラリング（修整）プロセス」

のあるものなので、それと区別するためである。

[71] 同書 pp. 12, 384-385。なお、この位置付けは、「システム管理基準」で独自に考案したのではなく、後掲の「共通フレーム」等を思慮なく踏襲したものであろうと筆者は捉えている。

[72] 同書 pp. 12, 384

[73] 「システム監査基準」は「システム管理基準」策定以前から策定されていたが(1985 年策定、1996 年並びに 2004 年改訂)、「システム管理基準」策定以降は「システム監査は、本監査基準の姉妹編であるシステム管理基準を監査上の判断の尺度として用い、監査対象がシステム管理基準に準拠しているかどうかという視点で行われることを原則とする」(経済産業省商務情報政策局(2005)第２分冊 P. 3)、とされており、そうした意味で「システム管理基準」は一定の「規範性」を有しているのである。

[74] 「イネーブリングとは、あるもの(例えば対象システム)の実現を可能にするための資源や基盤をいう」（ＩＰＡ監修(2013)p. 50)、そして「組織のプロジェクトイネーブリングプロセスは、プロジェクトの開始、支援及び制御によって製品又はサービスを取得及び供給するための組織の能力を管理する」(同書 p. 252)、ということである。

という８つであるが[75]、「4. 支援プロセス」から「8. テーラリング（修整）プロセス」までは共通的ないし横断的なプロセスであり、「1. 合意プロセス」は「取得者」と「供給者」という「二つの組織の間の合意を確立するために必要なアクティビティを定義する」[76]ものなので、いわばソフトウェア・ライフサイクルを成り立たせる前段階のプロセスであり、「2. テクニカルプロセス」から「3. 運用・サービスプロセス」が直接的にライフサイクルに関わるものである。「プロセス」と称呼しているが、筆者のいう大フェーズに相当する[77]。「2. テクニカルプロセス」の小プロセスは、

「2.1　企画プロセス」

「2.2　要件定義プロセス」

「2.3　システム開発プロセス」

「2.4　ソフトウェア実装プロセス」

「2.5　ハードウェア実装プロセス」

「2.6　保守プロセス」

となっている。「3. 運用・サービスプロセス」の小プロセスは、

「3.1　運用プロセス」

「3.2　廃棄プロセス」

「3.3　サービスマネジメントプロセス」

となっている。独特の用語（呼称）に拘泥せず、内容の理解に努めれば、概ね一般的なライフサイクルが設定されていることが了解できる。廃棄は、やはり独立的なプロセスとはされていないが、「システム管理基準」とは異なり、運用のプロセスの中に位置付けられている。次善の策としては、順当である。但し逆に、保守の取り扱いが独立的ではなく、軽微な扱いとなっている。「テクニカルプロセス」という大枠の中で、企画に始まる一連のプロセスの１つという位置付けは間違いとまでは言えないが、例えば「ソフトウェア実装プロセス」と同格の位置付けは不適切である。何故ならば、保守は開発ほどの規模ではないとしても、要件定義からソフトウェア実装のプロセスまでを包含する場合が少なくないからである。運用を独立的なプロセスとしているのであるから、少なくともそれと同等の独立的なプロセスとすべきである。そのような問題含みではあるが、ソフトウェア・ライフサイクルに概ね沿った構成となっており、「システム管理基準」と同等の意味で、ライフサイクルを視野に入れた管理が定式化されていることは確かである。

「共通フレーム」の歴史的な変遷（異同）を逐一トレースすることは当文脈で取り上げる目的か

---

[75] 同書 pp. 90-296

[76] 同書 p. 90

[77] 用語に関することで、断っておきたいことがある。「共通フレーム」等では「プロセス」(process) という用語が、筆者の大フェーズないしフェーズに相当するものとして使われている。プロセスは非常に多義的で、様々な対象や様相に対して使われており、どちらが適切かといったことは一概に言えるものではないが、筆者は当論文全体で、プロセスをプロダクトとの対比で使っているので、基本的にはその用法に特定することとし、つまり制作し利用する過程を総体として捉える場合に使用し、それを区分する場合には(大)フェーズという用語を使用することに統一する。

らは逸脱するので行なわず、最初の 1994 年版を確認する（最新の 2013 年版が却って改悪となっていることもあるからである)。プロセスは、15 のプロセスが設定されており[78]、「システム管理基準」や 2013 年版（2007 年版も同様）のように共通的ないし横断的なプロセスを別に括り出すのではなく、同列的に構成されているようにみえるが、「共通フレームのプロセス相関図」では、

「契約作業」として「1. 購入プロセス」
「2. 供給プロセス」
「システム開発作業」として、
「共通プロセス群」の「7. 管理プロセス」並びに
「8. 環境整備プロセス」
「9. 教育訓練プロセス」
「10. 文書作成プロセス」
「11. 構成管理プロセス」
「12. 品質保証プロセス」
「13. 問題解決プロセス」
「基本プロセス群」の「3. 企画プロセス」
「4. 開発プロセス」
「5. 運用プロセス」
「6. 保守プロセス」

が位置付けられている[79]。「相関図」には図示されていないプロセスとして、

「14. 組織の確立・評価・改善のためのプロセス」
「15. システム監査プロセス」

がある[80]（それで計 15 プロセスである)。1994 年版では、企画・運用・保守が「システム開発作業」という大きな括りの中に位置付けられており、これは「開発」の拡大解釈に過ぎ、明白に誤りであるが、それを除けば、企画・開発・運用・保守が同等に位置づけられている点は妥当であり、2013 年版が改悪となっている[81]。廃棄は、「6. 保守プロセス」の中の「6.6　旧システムの廃棄」に位置付けられており（2007 年版までは同様)、「共通フレーム」において一定（一貫）していないが、2013 年版で変更されたことは改良とは言える。但し、独立的でない点では次善の策に留まる、と言っておかなければならない。

---

[78] 共通フレーム検討委員会編(1994)pp. 33-106。なお、副題にある「SLCP-JCF」は Software LifeCycle Process - Japan Common Frame の略号である。
[79] 同書 p. 17
[80] 「ソフトウェアを中心としたシステムの取引に関する共通フレーム体系(1994 年版)」という図でも、「枠外」的な位置付けになっている(同書 p. 19)。
[81] 1998 年版、2007 年版(第 2 版を含む)までは 1994 年版と同等の位置付けになっていたが(SLCP-JCF98 委員会編(1998)、ＩＰＡ編(2007)、ＩＰＡ編(2009)参照)、2013 年版では企画と保守が独立的ではなくなっている。

1994 年版は、上記の通り大小幾つかの問題点はあるが、企画・開発・保守・運用・廃棄というソフトウェア・ライフサイクルを曲がりなりにも捉えていることは確かなことである[82]。

（3）ISO/IEC(JIS)「ソフトウェアライフサイクルプロセス」

ISO/IEC の‘System and software engineering-Software life cycle processes’（ソフトウェアライフサイクルプロセス）ISO/IEC 12207 は、1995 年に制定され、2002 年、2004 年、2008 年に改訂されている。それに対応する JIS の「ソフトウェアライフサイクルプロセス」JIS X 0160 は、1996 年に制定され、2007 年（ISO/IEC12207:2002,2004 対応）、2012 年（ISO/IEC12207:2008 対応）に改訂されている[83]。

JIS 規格の最新 2012 年版[84]に拠ると、「6　システムライフサイクルプロセス」は、

「6.1　合意プロセス」

「6.1.1　取得プロセス」

「6.1.2　供給プロセス」

「6.2　組織のプロジェクトイネーブリングプロセス」

「6.3　プロジェクトプロセス」

「6.4　テクニカルプロセス」

「6.4.1　利害関係者要求事項定義プロセス」

「6.4.2　システム要求事項分析プロセス」

「6.4.3　システム方式設計プロセス」

「6.4.4　実装プロセス」

「6.4.5　システム結合プロセス」

「6.4.6　システム適格性確認テストプロセス」

「6.4.7　ソフトウェア導入プロセス」

「6.4.8　ソフトウェア受入れ支援プロセス」

「6.4.9　ソフトウェア運用プロセス」

---

[82] なお、ＩＰＡ編(2007)並びにＩＰＡ編(2009)に、「共通フレーム 98 では、産業界の要望に応えるべく国際規格になかった企画プロセスやシステム監査プロセスを追加、拡張していました」（各 p.ｖ、引用者注:原文の「国際規格にかなった」という誤記ないし誤植は改めた）という記述があるが、本文で掲示した通り、それらは 1994 年版にも既にあり、明白な事実誤認である。「共通フレーム」を継承したＩＰＡが、旧版をまともに読解していないことを露呈した「まえがき」を記すことは頂けない。

[83] 関連した標準(規格)として、上位標準とも言うべき‘Systems and software engineering ― System life cycle processes’（システム及びソフトウェア工学－システムライフサイクルプロセス）ISO/IEC 15288(2002 年制定、2008 年改訂)とそれに対応する JIS の「システムライフサイクルプロセス」JIS X 0170(2002 年制定、2013 年改訂)があるが、ソフトウェアを主題とする本論文では取り上げない。

[84] ＪＳＡ編(2013)pp. 139-227

「6.4.10　ソフトウェア保守プロセス」

「6.4.11　ソフトウェア廃棄プロセス」

となっている。「7　ソフトウェア固有プロセス」は、ライフサイクルとしては補足的なものだが、

「7.1　ソフトウェア実装プロセス」

「7.2　ソフトウェア支援プロセス」

「7.3　ソフトウェア再利用プロセス」

となっている。「共通フレーム」で既に触れたことは省略し（但し、見比べれば判然としているが、「共通フレーム」が当標準（規格）をベースとしながら、相当程度アレンジしている)、且つ粒度の異なるプロセスを同列に並べていることを逐一指摘することは省略し（例えば長期間に亘る運用や保守のプロセスと、導入や受入れ支援のプロセスといった短期間の限定的な作業を同列扱いすることは到底受け入れられない等)、これまでと同様の点検を行なう。企画は独立的に位置付けられていないだけではなく、一連のプロセスの１つとしても取り扱われていない[85]。「6.4.1　利害関係者要求事項定義プロセス」や「6.4.2　システム要求事項分析プロセス」は、作業的には類似的ではあるが、決定的に異なる点がある。これらは既に開発することが決定して以降のプロセスだが、企画は開発するかどうか未決の段階での検討であり、検討結果として開発する場合もあれば、しない場合もありうる。その意味で、これらのプロセスが企画に相当するというのは無理がある。それ以外は、「6.4　テクニカルプロセス」の一連のプロセスとして、開発の内訳的なプロセスと同列に取り扱われていることは支持できないが、運用・保守・廃棄が各々順当に位置付けられている。

遡及して、最初の 1996 年版[86]を確認する。「5. 主ライフサイクルプロセス」は、

「5.1　取得プロセス」

「5.2　供給プロセス」

「5.3　開発プロセス」

「5.4　運用プロセス」

「5.5　保守プロセス」

となっている。「6. 支援ライフサイクルプロセス」は、

「6.1　文書化プロセス」

「6.2　構成管理プロセス」

「6.3　品質保証プロセス」

「6.4　検証プロセス」

「6.5　妥当性確認プロセス」

---

[85] このことは、筆者の解釈というだけではなく、ＩＰＡ監修(2013)に「共通フレーム 2013, 共通フレーム 2007(第２版)と ISO/IEC 12207:2008(JIS X 0160:2012)の対比」が「付属資料」として掲載されており(同書 pp. 55-86)、その「企画プロセス」の JIS X 0160:2012 の対比欄が空白となっていることが有力な証拠である(同書 pp. 58-60)。

[86] ＪＳＡ編(2011)pp. 137-216

「6.6　共同レビュープロセス」

「6.7　監査プロセス」

「6.8　問題解決プロセス」

となっており、「7. 組織に関するライフサイクルプロセス」は、

「7.1　管理プロセス」

「7.2　環境整備プロセス」

「7.3　改善プロセス」

「7.4　教育訓練プロセス」

となっている。「6. 支援ライフサイクルプロセス」は共通的なプロセスであり、「7. 組織に関するライフサイクルプロセス」は組織運営上のプロセスであるから、「5. 主ライフサイクルプロセス」に着目すればよい。企画が独立的に位置付けられていないだけではなく、一連のプロセスの１つとしても取り扱われていない。最初から 2012 年版に到る迄、改善されていないということである。廃棄は、独立的なプロセスにはなっていないが、「5.5　保守プロセス」の中で「5.5.6　ソフトウェア廃棄」として位置付けられている[87]。以上のことから、企画は欠落しており、廃棄は保守の一部と看做されている点で支持できないが、開発・運用・保守・廃棄が取り扱われていることが確認できる。

幾つかの標準（規格）と言えるものにおいて、ソフトウェア・ライフサイクルがどのように捉えられているか（規定されているか）を概観した。多少の無視しえない違いはあるが（独立的な設定としているか否か、廃棄の位置付け等）、それ以外は大同小異であり、筆者の言う大フェーズとして企画・開発・保守・運用・廃棄を設定していることが確認できたであろう。そして、時期的に重要なことであるが、日本におけるこれらの初版が 1994 年ないし 1996 年には策定（制定）され、公表されていることである。これが、ソフトウェア会計基準にとって持つ意義は、後述する。

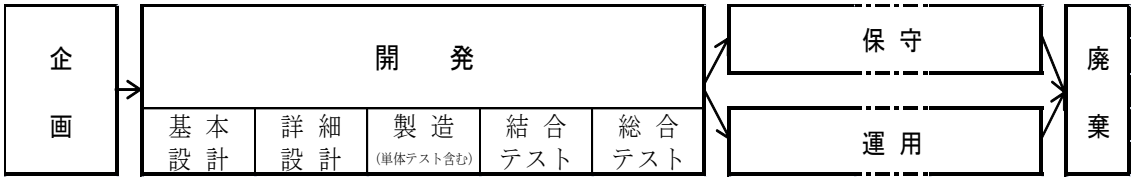
２．ソフトウェア・ライフサイクルの概容

（1）ソフトウェア・ライフサイクルの全体像

ソフトウェア・ライフサイクルの全体像に関して、各種標準における捉え方に関説する中で筆者の見解は必要な範囲で述べてきたが、問題点の指摘という形式でのことなので、必ずしも明解ではないかもしれないので、改めて直截に提示する。取り立てて独自のものではなく、「集合知」と言えるものだと考えている。図示すると、次のようになる。

[87] 同書 p. 159

図表B-2-1　ソフトウェア・ライフサイクル



（出典：筆者作成）

ソフトウェア・ライフサイクルを構成する大フェーズは、企画・開発・保守・運用・廃棄の５つである[88]。期間の長短、費消する工数等は相当に異なるが、作業内容が大きく異なるので、独立的な大フェーズと見做すことが妥当である。廃棄を、保守又は運用の一環として位置付ける「標準」が少なくないが、不適切である。保守や運用の最終工程と言えなくはないが、存続を前提に遂行する作業と存続を打ち切るための作業が性格的に異なることは明白である。企画を、開発と地続きで捉える、あるいは明示的に捉えない「標準」があるが、企画は開発に直結するわけではなく、開発しないという選択肢もあるのであり、開発とは一線を画す必要がある。なお、開発（内）のフェーズを参考までに図示したが、様々な分割の仕方があり、あくまで１例である。

（2）個々の大フェーズの概容

個々の大フェーズの内容に関しては、本論各章の該当箇所で詳細に取り上げるので、ここでは簡潔に触れるに留める。これまで取り上げてきた各「標準」から主要な作業（アクティビティ）を抽出し、それを表形式で纏め、本文では補足的な言及を行なう。形態という点からみると、開発はほとんどプロジェクト制であるが、それ以外は概ね定常的な組織で行なうものである。これが外形的にみた大きな特徴である。開発以外でも、案件によって（大規模もしくは高度に専門性を有する場合等）、プロジェクト制を採用する場合もあるが、一般的には定常的な組織がキャパシティに応じて案件を消化していくという遂行の仕方を採用するものである。

①企画

組織的なことで言えば、例えば大企業でシステム部門を子会社化することはありふれたことだが、その場合も企画部門は親会社に残すことが多いと言える。企画部門は、システムに関する予算編成機能を有するのが一般的である。その中で、企画対象は大別すると、A．全くの新規案件／B．既

[88] オーム社編(2001)は、ソフトウェア・ライフサイクルを「ソフトウェアの立案、開発から廃棄までの工程」(p. 388)としており、保守や運用を明示していないが、「立案」(企画の別名と解せられる)から「廃棄まで」と明記することでほぼ過不足なく捉えている。但し、その後に「この工程には、いくつかの類型があり、ライフサイクルモデルと呼ばれる。ウォーターフォールモデルやスパイラルモデルなどが提案されており、ライフサイクルモデルに沿って開発を進めていくことで、進捗管理や品質管理を系統的に行うことができる」(p. 388)とあるが、これは「開発」工程に関するモデルであり、せいぜい保守にまで適用可能な程度で、ライフサイクル全体のモデルではない。企画、運用、廃棄に「ウォーターフォールモデル」等は適用できるものではない。

存システムへの追加（修整）案件／C．既存システムの全面再構築（機能追加を含む）案件に分かれる（それ以外に、保守や運用の業務改善、開発標準の改訂等の、プロダクトではなく、プロセスに関わることもあるが、当文脈では立ち入らない）。そして、検討結果として、a．対処しない（却下、保留、先送り）／b．現行システムの保守で対処／c．（パッケージ・ソフト等の）購入／d．開発、という選択肢（代替案）がありうる。こうしたことからも、企画→開発というのは1つの選択肢（代替案）に過ぎず（単純に余り起こらない組み合わせを合わせれば、3×4の 12 通りのうちの1つ）、それらを直結させることは短絡的と言わざるを得ない（各「標準」はその傾向が強い）。企画を独立的な大フェーズとすべき所以である。

図表B-2-2　個々の大フェーズの概容

大フェーズ	システム管理基準	共通フレーム2013	JIS X 160:2012
企 画	開発計画 　システムライフの条件の明確化 　実現可能な代替案作成検討 分析 　導入効果の定量的・定性的評価 　パッケージのニーズ適合性検討 調達 　要員の必要スキル明確化	システム化構想の立案 　調査分析 　対象の選定と投資目標の策定 システム化計画の立案 　サービスレベルと品質の基本方針明確化 　実現可能性の検討 　費用と投資効果の予測	
開 発	開発手順 システム設計 プログラム設計 プログラミング システムテスト・ユーザ受入れテスト 移行	要件定義 システム開発 　システム要件定義 　システム方式設計 　実装 　システム結合 　システム適格性確認テスト 　システム導入 　システム受入れ支援	利害関係者要求事項定義 システム要求事項分析 システム方式設計 実装 システム結合 システム適格性確認テスト ソフトウェア導入 ソフトウェア受入れ支援
運 用	運用管理ルール 運用管理 入力管理 データ管理 出力管理 ソフトウェア管理 ハードウェア管理 ネットワーク管理 構成管理 建物・関連施設管理	運用テスト及びサービスの提供開始 業務及びシステムの移行 システム運用 利用者教育 業務運用と利用者支援 システム運用の評価 業務運用の評価 投資効果及び業務考課の評価	運用の開始及び終了 実運用 顧客支援 運用上の問題解決
保 守	保守手順 保守計画 保守の確認 移行	問題把握及び修正の分析 修正の実施 保守レビュー及び／又は受入れ 運用テスト及び移行の支援	問題及び修正 修正の実施 保守レビュー及び／又は受入れ 移行
廃 棄	情報システムの廃棄 　廃棄計画の策定 　不正防止及び機密保護対策	システム又はソフトウェア廃棄計画 廃棄の実行 　新旧環境並行運用と利用者教育訓練 　関係者全員への廃棄通知 　廃棄関連データ保持･安全性確保	ソフトウェア廃棄計画 ソフトウェア廃棄の実行

（出典：項目は経済産業省商務情報政策局(2005）、ＩＰＡ監修(2013)、ＪＳＡ編(2013)から抽出、表化筆者）[89]

[89] 各「標準」間で、また同一「標準」内でも、同一項番レベルが必ずしも同等の作業粒度ではないので、各大フェーズの概容理解に資する目的で適宜抽出した。特に一般的には十分に知られている

②開発

ソフトウェア・ライフサイクルにおいて、中心的な大フェーズと見做されており（保守や運用に比べれば、期間的には遥かに短いにもかかわらず)、最も注目されていると言えるだろうが、今でも失敗が少なくない。対策本も枚挙に暇がないくらい次々に出されているが、効能は乏しい。筆者は、ソフトウェア開発プロジェクトのアドホック性を特に強調したい。詳細は後続の論考のなかで関説する。

③運用

運用は、これまで出来上がったものをただ動かすだけといった消極面が強かったが、ＩＴの広範な普及並びにそれに伴うリスクの増大により、重要度が顕著に増してきた。更に、未然防止には限界があり、継続的な運用改善の必要性が増してきた。

④保守

保守は、開発と共通することも多いが、一般的には比較的小規模であり、且つ既存システム（母体）があることが、開発とは大きく異なることである。従って、作業内容（アクティビティないしタスク）が開発とは異なるのである（回帰テストの必要性、仕様の劣化へのリファクタリング対応等)。

⑤廃棄

廃棄は、作業的には、既存環境を消去し必要資源は保存するといった比較的単純且つ短期間のものだが、自らの「寿命」によるだけではなく、他律的な要因（機能的な陳腐化、合併等の組織の統廃合に伴うシステムの片寄せのシワ寄せ等）によることも多く、それによって付随的な作業が異なることも少なくない。また逆に、ソフトウェアの「寿命」は一般的に思われているよりも長いもので、ライフサイクルを通して同一担当者が関わり続けることはそれほど多くない。それ故、ＴＣＯ等を記録・捕捉されることはほとんどないと言える（資産管理としては驚くほど杜撰であり、「後半生」は既に減価償却済みであることがほとんどである)。そうした意味で、筆者は廃棄に関して、取り組むべきことが少なくない、と考えている。なお、「標準」で、廃棄の実行で全てが終わるのではなく、アフターケア（アフターフォロー）とも言うべきことを挙示している点は留意すべきであろう。

とは言えない企画と廃棄はやや精細に概容を知り得るように配慮した。

ソフトウェア基礎論

第３章　ソフトウェアの分類

１．ソフトウェアの一般的な分類

（１）ソフトウェアの分類Ａ（エンタープライズ系／組み込み系）

（２）ソフトウェアの分類Ｂ（基本ソフトウェア／ミドル・ソフトウェア／応用ソフトウェア）

（３）ソフトウェアの分類Ｃ（パッケージ・ソフトウェア／カスタム・メイド・ソフトウェア）

（４）ソフトウェアの分類Ｄ（商用ソフトウェア／オープンソース・ソフトウェア）

２．国民経済計算（SNA）におけるソフトウェアの分類

（１）国民経済計算（SNA）におけるソフトウェアの分類

（２）分類上の個々のソフトウェアの取り扱い

３．ソフトウェア会計基準におけるソフトウェアの分類

（１）研究開発的観点からのソフトウェア分類の点検

（２）組み込み系観点からのソフトウェア分類の点検

（３）収益の源泉という観点からのソフトウェア分類の点検

（４）総合的観点からのソフトウェア分類の改訂案

補遺　ソフトウェア分類別のライフサイクルへの関与

第３章　ソフトウェアの分類

１．ソフトウェアの一般的な分類

ソフトウェア業界等で行なわれている、ソフトウェアに関する一般的な分類は、幾通りかある。それらは各々明確な分類基準に基づいたものであり、用途が異なり、有用なものである（A、B等の呼称は便宜的なもので、敢えて特段の意味はないものとする)。

（1）ソフトウェアの分類A（エンタープライズ系／組み込み系）

ソフトウェアの分類Aは、

①エンタープライズ系

②組み込み系

という分類である。①エンタープライズ系（enterprise system）は、特に大型の企業システムを限定的に指す場合があり、その場合は「企業の業務システムや情報システム、銀行、証券、病院、鉄道など大規模かつ社会基盤を支える情報システムなどに含まれ、それらのシステムの機能を中心となって実現するもの」（ＩＰＡの定義）[90]ということだが、そのように限定しない場合には、企業の業務システム全般を指す。それに対し、組み込み系（Embedded system)は、特定の機能を実現するために家電製品や機械等に組み込まれるシステムのことである。ソフトウェアを包括し、且つ最も大きく二分した分類であるが、ソフトウェア技術者がほぼ截然と二分される業界の分野という意味でも、大分類としては意義を有している。

筆者としては、その分類基準をより明確に「対応するハードウェアの種類」とすることができるのではないかと考えている。①エンタープライズ系は汎用のハードウェアに対応するソフトウェアであり、②組み込み系は専用のハードウェアに対応するソフトウェアである。少々補足をすると、汎用／専用のハードウェアという区分はその境界領域で多少の不分明さを生じるかもしれないが、概ね区分可能である。しかも専用のハードウェアというのは、コンピュータ・ハードウェア（狭義のハードウェア）に限定するものではなく、それ以外の機械等のハードウェアをも含めた意味である（製造装置や家電製品でもそうである)。そこで、例えば専用のハードウェアである銀行等のＡＴＭ（Automated Teller Machine：現金自動預け払い機の略称）に搭載されるソフトウェアは、組み込み系と言うことができる。それらを含めた総体的な勘定系システムはエンタープライズ系であるが、その一部のサブシステムであるＡＴＭ搭載のソフトウェアは組み込み系である。同じように、銀行等の営業店端末はやはり専用のハードウェアであるから、それに搭載されるソフトウェアは組み込み系ということになる。工場の製造装置に搭載されているソフトウェアは組み込み系であるが、

[90] http://www.ipa.go.jp/sec/softwareengineering/std/ent.html(アクセス：2014/06/05,11:37)

その製造装置もネットワークで当該企業の業務システムと接続していれば、全社的なエンタープライズ系システムにおける一部のサブシステムとしては組み込み系であるということになる。各々完全に独立的・あるいは孤立的ではなく、相互乗り入れ的な関係になる場合があるとしても、区分できるものであり、また特有のアーキテクチャでの構成となるという意味で区分することに意義があるのである。

（2）ソフトウェアの分類B（基本ソフトウェア／ミドル・ソフトウェア／応用ソフトウェア）

ソフトウェアの分類Bは、

①基本ソフトウェア

②ミドル・ソフトウェア

③応用ソフトウェア（アプリケーション・ソフトウェア）

という分類である。これは、階層的な分類で、ハードウェアへの遠近（依存度合）を基準としたもので、①基本ソフトウェアがハードウェアに最も近く、③応用ソフトウェア（アプリケーション・ソフトウェア）が最も遠い位置付けにある。①基本ソフトウェアの代表はＯＳであり、他に各種のドライバー（ハードウェアの制御ソフト、例えばプリンタ・ドライバー）等がある。②ミドル・ソフトウェアの代表的なものはオンライン・コントロール・ソフトやＤＢＭＳ等である。それ以外のソフトウェアが、③応用ソフトウェア（アプリケーション・ソフトウェア）であり、圧倒的に多数のソフトウェアが該当する。特定のソフトウェアが各境界のいずれに属すると見做すかには多少の変動や見解の相違はありうるが、ソフトウェア・アーキテクチャを階層的に捉える上で有用な分類であるし、ソフトウェア技術者がどの階層のソフトウェアの制作等に主に携わるかという分野の性格を有するという意味でも有用である。

（3）ソフトウェアの分類C（パッケージ・ソフトウェア／カスタム・メイド・ソフトウェア）

ソフトウェアの分類Cは、

①パッケージ・ソフトウェア

②カスタム・メイド（オーダー・メイド）・ソフトウェア

という分類である。これは、個々の企業の業務に対応して個別に制作し、独自仕様の②カスタム・メイド（オーダー・メイド）・ソフトウェアと、汎用的な同一仕様の製品である①パッケージ・ソフトウェア（ソフトウェア会計基準の市場販売目的のソフトウェアに一応相当する）に区分する分類である。①パッケージ・ソフトウェアも、カスタマイズということで、部分的に顧客の要望で、機能追加や変更をし、部分的に独自仕様（専用仕様）となることがある。②カスタム・メイド（オーダー・メイド）・ソフトウェアも、市販のソフトウェアを一部組み込んだり、共通モジュール等を組み込むことがあり、全てが独自仕様というわけではないこともある。そういう意味では、両者が重なり合う部分や相互乗り入れのような様相を呈することもあるが、基本的には明確に区別できるも

のである。

（4）ソフトウェアの分類D（商用ソフトウェア／オープンソース・ソフトウェア）

ソフトウェアの分類Dは、

①商用ソフトウェア

②オープンソース・ソフトウェア

という分類である。これは、ソフトウェアの全範囲を包括した分類ではなく、外部から取得するもので、営利のソフトウェア企業が制作・販売する有償の①商用ソフトウェアと、基本的には無償の②オープンソース・ソフトウェアに区分する分類である。包括的な分類とするには、③として内部で制作する（委託開発を含む）自社開発ソフトウェア（内製ソフトウェア）を含めた方がよいかもしれない。全体的には自社開発だが、一部のコンポーネントとして①ないし②のソフトウェアを含むことは少なくない。というより、全体的なアーキテクチャのうち、基本ないしミドル・ソフトウェアの大半は①ないし②のソフトウェアとし、応用ソフトウェア（アプリケーション・ソフトウェア）だけ又は一部ミドル・ソフトウェアを自社開発することが一般的である。

他にも大分類として異なる分類基準によるものがあるかもしれないし、上記分類AからDの更に内訳としての細分類はありうるが、それに立ち入る必要はないであろう。これらの分類に共通して言えることは、分類基準が明解であること、区分の境界領域では特定のソフトウェアがいずれに属するか、多少は不分明であったり人によって見解が異なったりすることがあるにせよ、基本的には個々のソフトウェアは分類のいずれかに属し、且つ専一的に属することである。それが、国民経済計算やソフトウェア会計基準の分類と決定的に異なることである。

２．国民経済計算（SNA）におけるソフトウェアの分類

国民経済計算（national accounting、national economic accounting）は、ＧＤＰ等の算出を行なうが、その計算（推計）方法等を定めた規範とも言うべき国際基準がSNA（Systems of National Accounts：国民勘定体系）である。SNAはこれまで、53SNA→68SNA→93SNA→2008SNA、と15年ないし25年毎に改訂されてきた（冠している数値は制定の西暦年である）[91]。日本は、1970年に53SNAへ、1978年に68SNAへ、2000年に93SNAへ各々移行してきた。2016年には、2008SNAへ移行予定である（但し一部移行済）[92]。その各改訂で、時代の推移に伴い、ソフトウェアの取り扱いも変化してきた。国民経済計算を取り上げるのは、企業会計と大きくは隣接する領域であり[93]、またソフト

[91] 作間編(2003)P.30、内閣府経済社会総合研究所国民経済計算部(2013)p.3

[92] 同上

[93] 但し、両者には入り組んだ関係があり(倉林(1989)p.22等参照)、本格的に取り扱うには別途長大

ウェアの分類として一見ソフトウェア会計基準と似たような分類を行なっており、「他山の石」として参考になることが少なくないからである。

（1）国民経済計算（SNA）におけるソフトウェアの分類

ソフトウェアの分類に入る前に、国民経済計算におけるソフトウェアの定義を瞥見しておこう。93SNA におけるソフトウェアの定義は、次の通りである。

「システムおよびアプリケーション・ソフトウェア双方についてコンピュータ・プログラム、プログラム説明書および補助材料。購入ソフトおよび自己勘定で開発されたソフトの双方が、支出が多額の場合、含まれる。／市場に向けられるものかどうかに関わらず、１年より長く使用されると考えられるコンピュータ・データベースの購入、開発または拡張に対する多額の支出もまた含まれる」[94]。

ソフトウェアに関して、独特の捉え方がなされている、と言える。「補助材料」というだけでは何のことか文脈だけからでは不明だが、データベースのことである[95]（改行後の定義におけるデータベースへの言及に繋がっている）。有力な傍証は、次の 2008SNA では、資産カテゴリーの項目名を「コンピュータ・ソフトウェアとデータベース」に改めたくらいであり[96]、データベースを類同物として重視していることが改名からも判明する[97]。また、「支出が多額」かどうかが、ソフトウェアの定義に「含まれる」というのは如何なものか。推計上の取り扱いとしてならば了解できるが、定義の問題ではないであろう。それと、定義で「プログラム説明書」としているのを、大森徹(1998)では、「仕様書、マニュアル、フローチャート等のドキュメント類」、と敷衍しているが、筆者には少々疑問である。並記で且つ先頭に「仕様書」（あるいは設計書）を挙げるのが通例といえるところを、単独で「プログラム説明書」としている含意は、パッケージ・ソフトウェアで説明書がセットになっているようなことを想定しているかもしれないからである。しかも、ここでも「フローチャート」を挙げることで、ソフトウェア基礎論第１章で既に指摘したことだが、大森徹はカテゴリー・ミステイクに陥っている。

上記のソフトウェアの定義を念頭に置いて、ソフトウェアの分類をみることにする。現行[98]の

な論文を必要とするので、本章ではソフトウェアの分類に限定して取り扱う。

[94] 『1993 年改訂　国民経済計算の体系』邦訳版上巻 p. 356、第ⅩⅢ章付録　資産の定義、「コンピュータ・ソフトウェア」（大森(1998)p. 4 参照）。

[95] 大森(1998)では、「この定義によれば」として、当たり前のように、「コンピュータ・ソフトウェアには、コンピュータ・プログラム本体と、仕様書、マニュアル、フローチャート等のドキュメント類、及びコンピュータ・データベースが含まれることになる」(p. 5、傍点引用者)、と敷衍している。

[96] 木村俊孝、木村早霧(2012)p. 113

[97] 現行のソフトウェア会計基準ではデータベースはコンテンツと見做されるし、著作権法では「プログラム」と「データベース」は別の著作物と見做されており（第二条定義の十の二、十の三）、かなりの相違がある。

[98] 日本が基準として採用しているという意味であり、世界的には 2008SNA に徐々に移行している。

93SNA では、ソフトウェアの分類を資本勘定の総資本形成という脈絡で、次のように行なっている。

「企業が１年を超えて生産に使用することを予定しているコンピュータ・ソフトウェアは、無形固定資産として取り扱われる。そのようなソフトウェアは、市場で購入されたり、自己使用向けに生産されたりする。そのようなソフトウェアの取得を総固定資本形成[99]として取り扱うわけである。市場で購入されたソフトウェアは、購入者価格によって評価する。社内（インハウス）で開発されたソフトウェアは、推定基本価格、もしくは基本価格の推定が不可能な場合は生産費によって評価する」(10.92)[100]。「また、ソフトウェアの総固定資本形成には、企業が１年超の期間にわたり生産に使用することを予定している大型データ・ベースの購入や開発を含む。こうしたデータ・ベースは、上述したようなソフトウェアの場合と同一の方法で評価する」(10.93)[101]。

これを、浜田浩児(2001)はわかりやすい形で整理している[102]。

①「ハードウェア（電子計算機本体）と一体のもの」

②-1 受注型ソフトウェア

②-2 汎用型ソフトウェア

③内製型ソフトウェア[103]（自社開発ソフトウェア、インハウス（in-house）型）ソフトウェア）

（④大型データベース、筆者追加）

①は一般的には組み込み系のソフトウェアのことである。68SNA では、これだけがソフトウェアとして取り扱われ、しかも機械等として有形固定資産（の一部）という取り扱いだったが、93SNA でも引き続き同様の取り扱いがなされている。①以外が、93SNA で初めて国民経済計算において、ソフトウェアとして明示的に取り扱われるようになったものである。ソフトウェア会計基準との対応付けをすると、②-1 は受注制作のソフトウェア、②-2 は市場販売目的のソフトウェア、④は自社利用のソフトウェアに概ね対応している、と取り敢えず看做すことができよう。但し、後述するように、実質的な意味合いはかなり異なるのである。

次に、その取り扱いが少々「錯綜」していることを取り上げ、整理を試みる。内閣府経済社会総合研究所国民経済計算部(2011)では、パッケージ型（②-2）と受注型（②-1）は平成 12 年基準（日本が 2000 年に 93SNA に移行したとき）から「固定資本形成に計上」していたが、自社開発ソフトウ

---

[99] 総固定資本形成(gross fixed capital formation)とは、「生産者による、①新規または既存の有形固定資産(住宅、住宅以外の建物、その他の構築物、輸送機械、その他の機械・設備、育成資産)の取得マイナス」処分、②新規および既存の無形固定資産(コンピュータ・ソフトウェア等)の取得マイナス処分、③土地を含む有形非生産資産の改良」のことである(作間編(2003)p. 330)。

[100] 『1993 年改訂　国民経済計算の体系』邦訳版上巻 p. 267。なお、貸借対照表の記入項目という脈絡でも、同様の処理規定がある(13.44、p. 340)。

[101] 同書 p. 267

[102] 浜田(2001)pp. 69-70(木村俊孝、木村早霧(2012)p. 109 もほぼ同様である)。

[103] 浜田らではなく、中村が「内製型ソフトウェア」という言い方をしており(中村(2010)p. 114)、受注型・汎用型と合わせる意味ではいい用語なので援用することにした。但し、「内製型」とは言え、実態的には大なり小なり委託開発を含んでいることは常に承知しておかなければならないことは言うまでもない。

ェア（③）はまだ「固定資本形成の対象外」であった。それを、平成17年基準から「固定資本形成に計上」するように改定した[104]、としている。ところが、経済企画庁経済研究所(2000)によると、（旧）68SNAでは「受注型ｿﾌﾄｳｪｱ、鉱物探査については、資産には記録されていない」ものだったが、（新）93SNAでは「ｺﾝﾋﾟｭｰﾀ・ｿﾌﾄｳｪｱのうち受注型ｿﾌﾄの部分については、新たに無形固定資産に計上し、表章項目とする。鉱物探査については、無形固定資産に計上するが表章項目とはしない」ことに改訂した、としているのである[105]。つまり、資産計上することに改訂したのは②-1受注型ソフトウェアだけで、②-2汎用型ソフトウェア（パッケージ型）は見送る扱いとしたのである。そして、浜田浩児(2001)でも、そうした事情を次のように説明している。「日本の国民経済計算の93SNA移行においては、（中略）電子計算機と一体のものを有形固定資産、受注型ソフトウェアを無形固定資産として位置付け、総固定資本形成の一部として計上する。なお、電子計算機と一体のコンピュータ・ソフトウェアについては、68SNAにおいても固定資産、総固定資本形成として位置付けられていた。／一方、汎用型ソフトウェアは、総固定資本形成とせず、従来どおり中間消費[106]としている。これは、基礎統計からは、企業の購入した汎用型ソフトウェアを総固定資本形成と中間消費に分離することができないためである。産業連関表[107]においても、汎用型ソフトウェアは、総固定資本形成ではなく、中間消費とみなしている。また、インハウス型ソフトウェアについては、93SNAにおいて、その推定価格または生産費用で評価された投資額を計上することが推奨されている。このため、自社内でソフトウェアを開発するためのプログラマーの雇用者数及びその報酬、中間投入等といった生産コストにかかる統計が必要になるが、企業の研究開発分（R&D）や雇用者報酬との重複計算が除去された包括的な推計を行うことができないため、インハウス型ソフトウェアの総固定

---

104 内閣府経済社会総合研究所国民経済計算部(2011)p.8

105 経済企画庁経済研究所(2000)「(参考資料２)我が国国民経済計算体系における主な変更点とその概要」p.10

106 「中間消費(intermediate consumption)とは、生産の過程で使い尽くされた財・サービスのこと」である(作間編(2003)p.37)。

107 産業連関表は５年毎に更新されるが、浜田の指摘していることの直接的な挙証ではないが、わかりやすい間接的な証憑を示すと、1990年産業連関表では列部門8512-01情報サービス、行部門8512-011情報サービス、品目表示は「ソフトウェア開発、情報システム開発、プログラム作成、受託計算サービス、…」（総務省統計局(1990)p.163）、1995年産業連関表では列部門8512-01情報サービス、行部門8512-011ソフトウェア業、8512-012情報処理・提供サービス、ソフトウェア業の品目表示は「ソフトウェア開発、情報システム開発、プログラム作成」（総務省統計局(1995)p.174）となっていたが、2000年産業連関表からは列コード8512-01情報サービス、行コード8512-011ソフトウェア業、8512-012情報処理・提供サービス、ソフトウェア業の品目表示は「受注ソフトウェア開発、業務用パッケージ、ゲームソフト、その他ソフトウェア」（総務省統計局(2000)p.159、傍点引用者）、2005年産業連関表では列コード7331-01情報サービス、行コード7331-011ソフトウェア業、7331-012情報処理・提供サービス、ソフトウェア業の品目表示は「受注ソフトウェア開発、業務用パッケージ、ゲームソフト、その他ソフトウェア」（総務省統計局(2005)p.236、傍点引用者）というように、汎用型ソフトウェアである「業務用パッケージ」が品目表示に挙げられるようになった。これは、中間消費ではなく、総固定資本形成に計上されるようになったことと相関的なことである。

資本形成への計上を見送っている」[108]。

そして、『SNA 推計手法解説書（平成 19 年改訂版）』で始めて「無形資産は、生産者が 1 年を超えて生産に使用するソフトウェアのうち受託開発分（受注ソフトウェア）及びパッケージ型ソフトウェア、鉱物探査、プラントエンジニアリングから構成される」[109]とし、汎用型ソフトウェアが無形固定資産として総固定資本形成に計上されるようになったのである。

一体どういうことであろうか。少々紛らわしいのだが、浜田や『SNA 推計手法解説書』が取り上げているのは推計方法（資産計上対象とするソフトウェアの指定含む）であり、その制定・改訂年次なのである。それに対し、内閣府経済社会総合研究所国民経済計算部(2011)が問題にしている何年基準とは、推計のベースとなる基準値の設定・改定年次のことなのである（それと併せて、「平成 17 年基準改定」では資産計上の対象変更も行なったということである)。「平成 17 年基準改定」ならば、平成 19 年の推計方法の改訂より前なのではないかというと、そうではないのである。「平成 17 年基準改定」を、平成 21 年～23 年に掛けて検討し決定したのであり、平成 19 年の推計方法の改訂より後なのである。「錯綜」は一応解消したことになる。

続いて、『国民経済計算の作成方法』によれば、「情報通信業」に関して、「『特定サービス産業実態統計』(経済産業省）等を基に推計する。／ソフトウェア業については、受注型ソフトウェア及びパッケージ型ソフトウェアに加え、自社開発ソフトウェアについても推計を行う。自社開発ソフトウェアは、社内で自己使用向けに生産・開発されるソフトウェアであり、市場価格で評価することができないため、開発に取り組んだ労働者の人件費等を基に生産額を推計する」[110]、という取り扱いになったのである。但し、自社開発ソフトウェア（③内製型ソフトウェア）に関しては、ソフトウェア企業分だけに限定していることを注意しておかなければならない。これによって、限定的であった日本の 93SNA 移行はソフトウェアに関しては、全て果たしたことになるであろうか。そうではないのである。自社開発ソフトウェア（③内製型ソフトウェア）に関して尚限定的なだけではなく、ことの是非はともかく、筆者が④で追加掲示したデータベースに関しては、管見の限り議論の俎上にすら乗っていない(2008SNA への移行において、どのような取り扱いとなるか、注目したい)。そうした意味で、統計当局が自負しているような「完全」移行とは言い難いのである。

（2）分類上の個々のソフトウェアの取り扱い

だいぶ長い前提的な確認を踏まえて、分類に関わる問題に取り組むことにする。１で取り上げた分類と、国民経済計算並びに会計基準におけるソフトウェアの分類は、全く性格の異なる分類である。１で取り上げた分類は、対象であるソフトウェアそのものの分類であるが、国民経済計算並びに会計基準におけるソフトウェアの分類はソフトウェア自体の分類ではなく、ソフトウェアとそれ

[108] 浜田(2001)pp. 69-70
[109] 内閣府経済社会総合研究所(2007)p. 68
[110] 内閣府経済社会総合研究所(2011a)p. 9

を制作ないし利用する企業との関係性を含めた分類である。それを承知しておかなければならない。

①組み込み系は、ハードウェアと一体的なので、有形固定資産として取り扱うという点では、ソフトウェア会計基準（基準四 3、実務指針 17）と対応していると言える。しかし、①の組み込み系は、分類としては一応挙げてはいるが、ハードウェアに従属するものとしてソフトウェア自体としては独立的に捉えようとせず、埒外に置いてしまっているというのが真相である。しかし、今日ではハードウェアよりもソフトウェアの方が開発費等におけるウェイトが高まっていると言えるくらいであり[111]、いつまでも旧套墨守で、結局のところハードウェアとして（その中に組み込まれているものとして）取り扱うことは、筆者には大いに疑問である。このことは、会計基準におけるソフトウェアの分類の箇所で再度言及する。

②-1 受注型ソフトウェアは、無形固定資産として計上し、固定資本形成として推計するが、基礎統計を採取し捕捉する箇所は受注したソフトウェア企業であるけれども、実際に資産（資本形成）とするのは発注した企業であると看做してのことである（個々の企業のことが表面化することは統計の性格上ないが）。受注したソフトウェア企業にとって、資産となるわけではないからである。つまり、受注型ソフトウェアと称してはいるが、受注企業の売上高を、発注し取得して利用する企業のソフトウェア資産の代理変数とする意味合いなのである。但し、ソフトウェア業界の多重構造を勘案すれば、丸投げ以外は総額表示なので[112]、同じソフトウェアに対して売上高が重複している可能性が高いので、その重複を適切に排除していなければ、過大計上となる。

②-2 汎用型ソフトウェア（パッケージ・ソフトウェア、ソフトウェアプロダクツ）は、双方に関わるはずである。制作したソフトウェア企業にとっては、複製を販売する原本として資産（資本形成）計上するが、購入した企業にとっては、複製を資産（資本形成）計上する、というようにである。ところが、ソフトウェア企業の売上高を、購入して利用する企業の資産の代理変数として使用し、国民経済計算における無形固定資産→総固定資本形成を推計することは概ね妥当だが、一方でソフトウェア企業の原本としての資産の捕捉が行なわれていないように見受けられる[113]。「基礎統計」のどこにも、これに関する情報がないからである。少し古い時点のものだが、大森徹(1998)によれば、経済企画庁の検討として、「「受注開発型ソフトウェア（特定ユーザーの独自の利用目的で制作するもの)」と「パッケージ型ソフトウェア（幅広いユーザーを想定して汎用的な既成製品として制

---

[111] 例えば、江口(2013)によれば、「財務省貿易統計　概況品別表」に基づくと、2012 年の輸出総額 63.7 兆円のうち、組込みソフトウェア関連製品は 37.9 兆円で比率としては 59.4%を占める、とのことである(p. 29)。また、経済産業省商務情報政策局情報処理振興課(2010)によれば、2009 年(版調査)では、組込み製品開発費においてソフトウェア開発費は 49.0%を占めており、最も多いのである(電子系ハードウェア 14.4%、機構系ハードウェア 10.4%、両者を合わせてもソフトウェアの半分程度)(p. 9)。

[112] 企業会計基準委員会(2006)「ソフトウェア取引の収益の総額表示についての会計上の考え方」(4)

[113] ソフトウェアの資産範囲が限られていた時期の議論ではあるが、大森(1998)での取り扱いの３つの基本的なケースの第一である「コンピュータ・ソフトウェアの原本、コピーとも無形固定資産として取り扱われるケース」(p. 9)に相当する。

作するもの)」のコピーに関しては販売額を使用者側の固定資本形成とすることについては問題ないとしている。しかし、概念的には、「パッケージ型ソフトウェア」の原本を制作者側の固定資本形成する必要があるが、この点については、このような資産をマクロ的に評価する方法が十分検討されていないと指摘している。このため、推計方法（試案）では販売額が把握可能な範囲のコンピュータ・ソフトウェアを使用者側の固定資本形成に計上するとされているのである[114]。つまり、汎用型ソフトウェアと称してはいるが、販売企業の売上高を、購入し利用する企業のソフトウェア資産の代理変数としているのであり、制作・販売企業の資産としては推計の対象としていないのである。

③内製型ソフトウェア（自社開発ソフトウェア（インハウス型））は、本来ならば、ソフトウェア企業でもユーザ企業でもありうるが、前述した通り、ソフトウェア企業が自社で制作し、利用するものとして資産（資本形成）計上することにしているのである。ユーザ企業に関しては、②-1 の受注型ソフトウェアを代理変数としているからであろうが、委託していない内製が欠落することになる。内閣府経済社会総合研究所(2011c)によれば、「自社開発ソフトウェアの固定資本形成への計上」に関して、「生産者が１年を超えて生産に使用するコンピュータ・ソフトウェアについて、９３ＳＮＡに適合するように新たに自社開発ソフトウェアを固定資本形成の推計対象に含める。自社開発ソフトウェアは、社内において自己使用目的で生産・開発されたソフトウェアであり、市場価格[115]で評価することができないため、開発に従事する労働者の人件費等を基に推計する」[116]、ということになった。その際、「アンケート調査も用いて、時間投入割合を推計した」とあるように、既存の「基礎統計」だけを用いるものではない。どのようなアンケート調査を行なったのであろうか。統計当局の公式の情報は筆者には検索出来なかったが、関係者の個人名での論文によると、「自社開発ソフトウェアの投資額を推計するに当たって、単純に生産費用＝ソフトウェア専門労働人数×人件費とするのは適当ではない。なぜならソフトウェア専門労働者が自社開発ソフトウェアの開発にすべての労働時間を費やしているわけではないからである。業務内容の中には自社使用目的のみならず、対顧客用や市販用ソフトウェアにかかる業務があり、さらにそれぞれ「開発」、「メンテナンス」、「間

---

[114] 大森(1998)p. 18。筆者の、ソフトウェア企業の売上高→利用企業の資産の代理変数、という捉え方の傍証となる。

[115] 国民経済計算では３種類の価格を取り扱っており、基本価格(basic price)は「生産者価格から純商品税を控除したもの」（作間編(2003)p. 317）であり、生産者価格(producers' price)は「生産者が出荷する時点の価格で、運賃・マージンを含まない価格」（同書 p. 195）であり、購入者価格(Purchasers' price)は「財・サービスが購入者に渡される時点での市場コストである。それは生産者価格に当該購入者の負担すべき運輸・商業マージンを加えた額に等しい」（同書 p. 320）、ということである。従って、自社開発ソフトウェアに関しては、それらのいずれとも異なる「生産費用」で推計するということである。

[116] 内閣府経済社会総合研究所(2011c)p. 3。『国民経済計算の作成方法』でも、同じように、「情報通信業」に関して、「『特定サービス産業実態統計』（経済産業省）等を基に推計する。／ソフトウェア業については、受注型ソフトウェア及びパッケージ型ソフトウェアに加え、自社開発ソフトウェアについても推計を行う。自社開発ソフトウェアは、社内で自己使用向けに生産・開発されるソフトウェアであり、市場価格で評価することができないため、開発に取り組んだ労働者の人件費等を基に生産額を推計する」（内閣府経済社会総合研究所(2011a)p. 9）という取り扱いをするとしている。

接業務」など分けられる。そこで、ソフトウェア労働者の労働時間のうち自社開発ソフトウェアの開発にかける時間の割合（時間投入割合[117]）を、システムエンジニアやプログラマーを対象としたアンケート調査結果（内閣府）を基に把握した」[118]、とのことである。具体的な調査項目は不明であるが、「図表５－１はアンケート調査から得られた時間投入割合である」[119]として、次のような興味深い数値が掲示されている。

図表B-3-1　システムエンジニア及びプログラマーの時間投入割合（単位：%）

業務内容	ソフトウェア業	非ソフトウェア業
ソフトウェア開発（自社使用）	3.73	18.85
メンテナンス業務（自社使用）	1.56	14.77
間接業務（自社使用）	0.80	12.26
ソフトウェア開発（顧客使用）	60.37	27.51
メンテナンス業務（顧客使用）	18.52	8.96
間接業務（顧客使用）	10.49	8.93
ソフトウェア開発（市販）	2.03	2.00
メンテナンス業務（市販）	0.78	1.91
間接業務（市販）	0.27	1.10
その他業務	1.45	3.72

（出典：木村俊孝、木村早霧(2012)p.111、但し図表タイトルは若干変更）

この調査自体は大変興味深いものであり、フル稼働かどうかを配慮していて、一見周到なように見えるが、問題がある。既に触れたことでもあるが、アンケート調査は限られたサンプル調査であろうが、対象をどのように選定したかである。ソフトウェア業界の階層構造の中で、大手企業から中小零細企業まで相当に業態が異なるので、「時間投入割合」も異なるのである。それを適切に平均値を析出できるような調査になっているか、それ如何に上記図表の数値の信頼性は大きく懸っているのである。また、対象をソフトウェア企業に限定してしまっているが、本来はユーザ企業をも対象としなければならないのであり、それに対する同様の調査が必要である。この場合も、大手企業から中小零細企業まで相当に業態が異なっているので、それに適切に対処したサンプル調査が必要である。それらを行なった上で得た数値であれば、有用となり得るであろう。

いずれにせよ、国民経済計算は推計に継ぐ推計を積み重ねて算出を行なっている。マクロ経済に関することなので、致し方ない面はある。しかし、会計学徒として、どうしても言っておかなければならないことがある。財務報告とその情報開示の役割である。本来ならば、このような多段階の

---

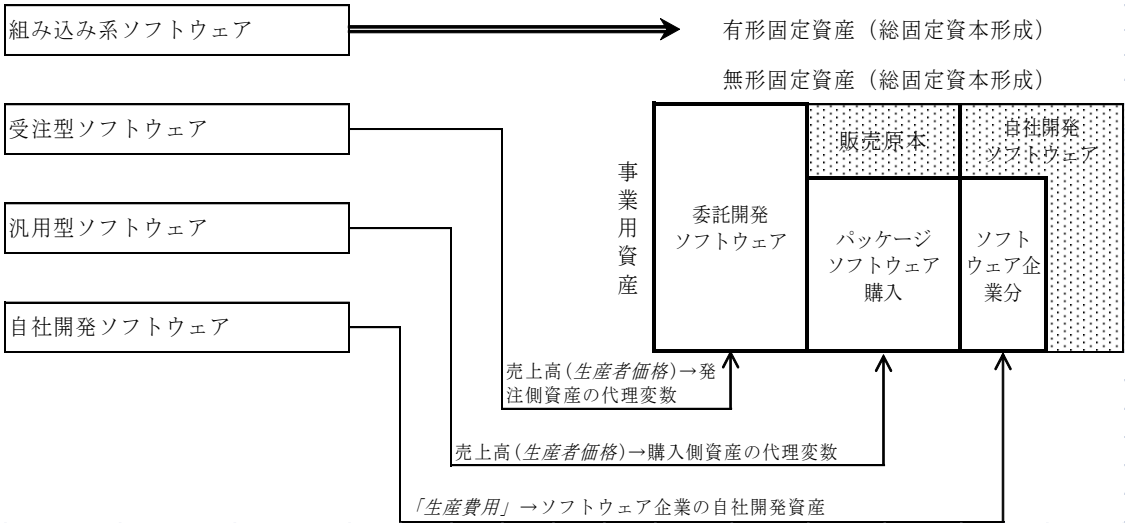
117 この箇所に注番号５が付され、「総労働時間を100%とした場合に自社開発ソフトウェアとして組み入れることが可能な労働時間の割合」（木村俊孝、木村早霧(2012)p.111）、と注記されている。

118 同論文p.111

119 同論文p.11。なお、この箇所に注番号が付され、「実際に推計に用いる時間投入割合は、自社開発ソフトウェアのうち研究開発(R&D)部分は中間消費とみなして控除する等の一定の仮定を置いた調整を行っている。図表５－１は調整前の時間投入割合である」（p.111）、と注記されている。「調整」の具体的な仮定内容は詳らかにされていないので、不明であるが、ここにもソフトウェア会計基準の「研究開発」看做しの悪しき影響が及んでいる。また、図表番号は同論文のものであり、引用では異なるものとなる。

推計を行なわなくとも、企業が財務報告で適切にソフトウェア資産を計上し、情報開示を行なっていれば、それを集計すれば済むはずのことではないのか、ということである。集計の労力は少なくないとしても、集計値の精確度は比較にならないはずである。一般に情報開示しているのは上場企業だけだとしても、確定決算主義の日本では、非上場企業であっても、財務報告に基づいて税務申告を行なうのであるから、税務申告内容によってソフトウェア資産の情報を収集することができる。それを何故使わないのか。代替的な多段階の推計を行なっているということは、国民経済計算における「基礎情報」として、財務情報が利害関係者への情報開示という機能を十分に果たしていない、あるいは国民経済計算の所管省庁に利用されていない事態である、と言わなければならない。

図表Ｂ-３-2　SNA におけるソフトウェアの分類と資産の対応



（出典：筆者作成）

国民経済計算におけるソフトウェアの分類とその取り扱いを、纏めとして図示すると、以上のようになる。網掛け部分は対象外となっている部分であり、模式的な図表なので、範囲の広狭までは示し得ていないが、欠落があることだけは確かなことである。分類並びに使用されている用語は、ソフトウェア会計基準のものと似ているが、その実質的な内容は大きく異なっていることが判然とする。使用する「基礎統計」の数値も、致し方ない面はあるが、売上高と「生産費用」というように異なっている。分類した対象のソフトウェアを包括的に取り扱っているようにみえて、対象外にしているものも少なくない。ソフトウェアに関する国民経済計算の数値を利用する場合には、こうしたことを十分に弁えておかなければならない。

なお、2008SNA では、研究開発を全て資産計上（総固定資本形成）することになっている[120]。今のところ、それに対する企業会計側からの応答（ＩＦＲＳを始め）等は筆者の知る限り見られないが、更に懸隔が拡がることになる。動向を注視していきたい。

[120] 中村(2010)p. 119 等

３．ソフトウェア会計基準におけるソフトウェアの分類

ソフトウェア会計基準におけるソフトウェアの分類は、１．で瞥見したようなソフトウェアの一般的な分類とは性格が大きく異なる分類である。対象の属性が分類基準であり、それによって対象が専一的に分類範疇のいずれかに帰属するのが一般的な分類であるが、ソフトウェア会計基準における分類はそうではない。対象とするソフトウェアと会計主体との関係に基づく分類だからである。同一のソフトウェアが、例えば受託したソフトウェア企業にとっては受注制作のソフトウェアであるが、委託したユーザ企業等にとっては自社利用のソフトウェアなのであり、各々の規定に拠って会計処理を行なうのである。そのような分類であること自体は、何も問題はない。取引における商品・製品一般の取り扱いと同様なことであり、会計的な取り扱いとしては当然なことと言ってよい。問題になるとすれば、その分類の仕方と内容である。

（1）研究開発的観点からのソフトウェア分類の点検

第一に、研究開発的観点から、ソフトウェア会計基準において、ソフトウェアがどのように分類されているか、を点検してみよう。下記の図表は、その通念的理解を表示したものと言える。

図表Ｂ－３-3　ソフトウェア会計基準におけるソフトウェア分類の通念的理解

研究開発			……
研究開発以外	受注制作		……
	市場販売目的		……
	自社利用	サービス提供	……
		購入	……
		社内利用	……

（出典：筆者作成）

一見ソフトウェア会計基準の規定を忠実に図表化したもので、会計基準の理解としては何も問題がないようにみえるかもしれない。しかし、そうではないのである。大きく研究開発と研究開発以外を区分しているが、「研究開発費に該当しないソフトウェア制作費に係る会計処理」（四、傍点引用者）においても、「市場販売目的のソフトウェアに係る会計処理」（四 2）で「研究開発費に該当する部分」（四 2、傍点引用者）があると看做しているのである。従って、少なくとも市場販売目的のソフトウェアに関しては、最初の大区分として研究開発以外としたにもかかわらず、まだ研究開発の基準が継続混入しているのである。市場販売目的のソフトウェアをそのような取り扱いにしているのは、製造業における製品の研究開発と製造との類推に拠るものである[121]。それに対して、自社利用のソフトウェアに関しては、明示的にはそのような取り扱いをしていない。資産性の規準を、「サービスを提供する」場合の「将来の収益獲得が確実であると認められる場合」、「社内利用のソフトウェアについては」「その利用により将来の収益獲得又は費用削減が確実であると認められる場

[121] 「ソフトウェア制作における研究開発と商業生産との区分」（企業会計審議会(1997)参考３）

合」(四 3)、としているからである。しかし、市場販売目的のソフトウェアだけに、「研究開発費に該当しない」のに「研究開発費に該当する部分」があると看做すのは、片寄った看做しであろう。研究開発が新奇性を基準とすることからすれば (一 1)、そして「研究開発費に該当しない」のに「研究開発費に該当する部分」があると看做すのであれば、自社利用のソフトウェアであろうと、受注制作のソフトウェアであろうと、「研究開発費に該当する部分」があると看做すのが、会計基準の分類基準としては整合的（一貫的）な適用ではないか。国民経済計算において、「自社開発ソフトウェアのうち研究開発 (R&D) 部分は中間消費とみなして控除する等の一定の仮定を置いた調整を行っている」[122]のは、尤もな解釈である（なお、国民経済計算のソフトウェアの分類は会計基準の分類とは異なっており、「自社開発ソフトウェア」はソフトウェア企業の自社利用（社内利用）のソフトウェアに限定されている)。従って、「研究開発費に該当する部分」があるか否かという観点から、ソフトウェアの分類を会計基準に明示的な規定により捉えてみる場合と、明示的ではないが整合的な解釈を施した場合とは、次のように異なる。

図表Ｂ−３−4　研究開発的観点からのソフトウェア分類の明示的規定に基づく理解

			研究開発該当部分の有無
研究開発			
研究開発以外	受注制作		
	市場販売目的		
	自社利用	サービス提供	
		購入	
		社内利用	

（出典：筆者作成）

図表Ｂ−３−5　研究開発的観点からのソフトウェア分類の整合的解釈に基づく理解

			研究開発該当部分の有無
研究開発			
研究開発以外	受注制作		
	市場販売目的		
	自社利用	サービス提供	
		購入	
		社内利用	

（出典：筆者作成）

多少の補足説明を行なうと、網掛け部分が「研究開発該当部分」であるが、研究開発と市場販売目的のソフトウェアに関しては範囲も規定に沿ったものとしているが（あくまでイメージ的なものではあるが)、それ以外は該当部分があるということを表示しているだけで、範囲の多寡を意味しない。明示的な規定はないので、確実な範囲画定はできないからである。

なお、本論（第２章）で詳細に論及するが、筆者の見解は次の通りである。そもそも「制作目的」として「研究開発目的」というのは、曖昧な規定である。製品又は製法として具体化される前の段階を研究開発目的と称するのは差し支えないが、制作が具体化されて以降の段階では研究開発目的ではなく、その段階からの制作目的は市場販売目的、受注制作、自社利用のいずれかになるであろ

[122] 木村俊孝、木村早霧(2012)p. 111

う。その中でも、部分的に複数代替案を実装し、選択するために制作する場合に研究開発目的と言っても構わないが、総体的には最早研究開発目的ではなく、市場販売目的等が制作目的だと言うべきである。次に、研究開発以外のソフトウェアにそもそも「研究開発該当部分」を混入させることが間違いであり、研究開発以外の受注制作、市場販売目的、自社利用のソフトウェアには「研究開発該当部分」はない、ということである（論証は該当箇所で行なう）。

（2）組み込み系観点からのソフトウェア分類の点検

第二に、組み込み系ソフトウェアの位置付けという観点から、ソフトウェア会計基準において、ソフトウェアがどのように分類されているか、を点検してみよう。意見書並びに基準（本体）には組み込み系ソフトウェアに関する規定はなく、実務指針にあるだけである。しかも、自社利用の購入の場合だけが取り扱われている（実務指針 17、41）。下記の図表は、それを表示したものである。

図表Ｂ−３−6　組み込み系観点からのソフトウェア分類の明示的規定

研究開発				……
研究以外開発	受注制作			……
	市場販売目的			……
	自社利用	購入		……
			機器組込み	有機的一体の場合は有形固定資産
		自社開発		……

（出典：筆者作成）

この規定が訝しいのは、自社利用で、しかも下位分類基準として取得形態を導入し、購入という取得形態だけに限定して「機器組込み系ソフトウェア」を取り扱っていることである。限定した理由は何も説明されていない。ここだけに「機器組込み系」を挙げているということは、他は全て明示的ではないが、エンタープライズ系だと解せられる。「機器組込み系」は、自社利用で、自社開発することはないのであろうか。そのようなことはない。市場販売目的で「機器組込み系」を制作することはないのであろうか。そのようなことはない（スマホやタブレットの例を挙げるまでもないであろう）。受注制作にしても、研究開発にしても、同様である。組み込み系ソフトウェアを取り扱うならば、次のような取り扱いをしなければならないことは明白であろう。つまり、自社利用の購入に限定するのではなく、エンタープライズ系の分類の全範囲に対応して、組み込み系ソフトウェアの取り扱いを挙示しなければ、取り扱う以上は不徹底となる。たとえ、現行基準では「有機的一体」の場合は有形固定資産とし、また研究開発や受注制作は費用処理とするにしても、である。

図表Ｂ−３−7　組み込み系観点からのソフトウェア分類の改訂案

機器組込み系ソフトウェア				
エンタープライズ系ソフトウェア				
研究開発			……	
研究以外開発	受注制作		……	
	市場販売目的		……	
	自社利用	購入	……	
		自社開発	……	

（出典：筆者作成）

但し、上記のことは現行基準ベースで不徹底なことを是正する場合の改訂案であって、筆者の見解は異なる。今日では「有機的一体」であっても、大半の場合に、ソフトウェア部分の識別は可能である。逆に確かに識別しにくいとすれば、購入する場合ぐらいであろう（必ずというわけではない）。しかも、その場合も今日ではハードウェア主体の有形固定資産として取り扱うのではなく、ソフトウェア主体で無形固定資産として取り扱うのが実態適合的である（開発費の比率等から明らかである）、あるいはハードウェア部分を有形固定資産としソフトウェア部分を無形固定資産として分けて取り扱うことが更に実態適合的である。従って、組み込み系をわざわざ別扱いする必要性に乏しく、エンタープライズ系と分けて取り扱う必要はないと考える。

（3）収益の源泉という観点からのソフトウェア分類の点検

第三に、収益の源泉という観点から、ソフトウェア会計基準におけるソフトウェアの分類を点検してみる。研究開発は、収益の源泉としては未確定（あるいは不確実）であろう。受注制作は、収益の直接的な源泉である。市場販売目的も、受注制作ほど収益額の確度は高くないとしても、収益の直接的な源泉である。自社利用のサービス提供も、収益の直接的な源泉である。自社利用の社内利用は、収益に関しては間接的な源泉とするのが穏当である。明解に費用削減に直結する場合もなくはないが（明白な労働代替の場合）、それはケース・バイ・ケースであり、一般的には間接的な源泉と捉えるのが妥当であろう。このように整理すると、際立つのは、市場販売目的と自社利用のサービス提供の取り扱いの違いである。市場販売目的とサービス提供の違いは、基本的には財貨の販売か、サービスの供与かの違いである。営業品目として確かに違いはあるが、それによって収益を獲得する点では変わりない。そうであるにもかかわらず、現行基準での取り扱いは大きく異なる。資産要件、資産範囲が大きく違うのである。市場販売目的のソフトウェアは、「研究開発費に該当する部分」が広範囲にあると看做され、資産計上はごく限られた範囲でしか行なわれない。それに対し、自社利用のサービス提供は資産要件（将来の収益獲得の確実性）を充足すれば、事実上全範囲が資産計上の対象となる。両者における財貨かサービスかの違いに対して、余りにも異なる取り扱いと言わざるを得ない。

図表B-3-8　収益の源泉という観点からのソフトウェア分類

			収益の源泉（直接的／間接的）	会計処理
研究開発			未確定	費用処理
研究以外開発	受注制作		直接的源泉	費用処理
	市場販売目的		直接的源泉	資産計上（限定的）
	自社利用	サービス提供	直接的源泉	資産計上（要件充足、高蓋然性）
		社内利用	間接的源泉	資産計上（要件充足）

（出典：筆者作成）

収益の直接的源泉／間接的源泉という観点から、ソフトウェアを分類するならば、もっと別様の分類がなされて然るべきではないか。それを図表化すると、次のようになる。

図表B−3−9　収益の源泉という観点からのソフトウェア分類の改訂案

		収益の源泉 （直接的／間接的）	会計処理
研究開発		未確定	費用処理
研究開発以外	受注制作	直接的源泉	費用処理
	市場販売目的又はサービス提供	直接的源泉	資産計上（要件充足）
	自社利用（社内利用）	間接的源泉	資産計上（要件充足）

（出典：筆者作成）

（4）総合的観点からのソフトウェア分類の改訂案

これまで3つの観点から、現行ソフトウェア会計基準におけるソフトウェアの分類を点検してきた。その点検結果を纏めてみる。要約するような形ではなく、総合する形で、改訂案を提示することにしたい。

図表B−3−10　総合的観点からのソフトウェア分類の改訂案

研究開発

（資産性） 収益の源泉	制作目的	取得形態	会計処理
直接的源泉	受注制作	自社開発＊	費用処理（収益総額表示）
		委託（丸投げ）	費用処理（収益純額表示）
	市場販売目的 又は サービス提供	自社開発＊	資産計上（同一要件充足）
		購入＊＊	
間接的源泉	自社利用	自社開発＊	
		購入＊＊	

＊一部委託含む

＊＊パッケージ購入（カスタマイズ含む）又はプライベート・クラウド利用等

（出典：筆者作成）

①研究開発費等に係る会計基準を分離し、研究開発会計基準とソフトウェア会計基準とを各々独立の会計基準とする。

②研究開発費等会計基準におけるソフトウェアの取り扱いは、ソフトウェアの研究開発に当然のことながら限定し、他方ソフトウェア会計基準のソフトウェアは研究開発以外のソフトウェアを取り扱い、且つ研究開発に該当する部分はないものとする。そもそも研究開発以外だが、研究開発に該当する部分があるなどということはないのである。なお、研究開発会計基準においては、ソフトウェアの分類は不要である。新奇性を追求する研究開発では、既存のソフトウェア分類には当て嵌まらないソフトウェアが出現することもありうるからであるし、現行基準では専らプロダクト（製品）としてのソフトウェアが取り扱われているが、研究開発ではプロセス（製法）としてのソフトウェアをも取り扱うのである。これらに関しては、本論第2章で詳細に論及する。

③ソフトウェア会計基準では、ソフトウェアを分類するが、第一分類規準は収益の源泉（直接的／間接的）とする。資産性（将来の経済的便益）の具体的な態様規定である。会計的には、「制作目的」より上位の最重要な規準である。これによって、現行基準では市場販売目的のソフトウェアと自社利用のサービス提供のソフトウェアが切り離され、異なる会計処理が規定されているが、その不整合を解消する。両者は財貨とサービスという形態的な違いはあるが、営業品目として、直接的

な収益の源泉たることでは共通する。更に、今日的にはクラウド・コンピューティングの普及・拡大により、同一のソフトウェアが財貨としての販売とサービス提供の資源とを兼ねる事態が出現しており、区分することの方が却って取り扱い上不都合が生じるのである[123]（詳細は、本論第８章「クラウド・コンピューティング」で論及する）。

④ソフトウェアの第二分類規準は、制作目的でよい。但し、上記で言及した通り、市場販売目的とサービス提供は一括りとし、自社利用をサービス提供と社内利用に細区分せず、自社利用とする。自社利用と言っても、ネットワーク化の進展により、対外的な接続・連繋や共同システム利用といった形態を含めてのことである。その意味で、現行基準の「社内利用」という用語はミスリードとなることが懸念されるので、引き継がないことにする。

⑤ソフトウェアの第三分類規準は、取得形態とする。大きくは、購入と自社開発に分けられるが、まず自社開発の内容には十分な理解が必要である。自社開発といっても、社内の社員だけで開発することの方が珍しい。大なり小なり委託開発を含むのである。しかも、委託開発といっても、契約形態としては委託（請負）・準委任・派遣・出向といった幾つもの形態があり、管理方式も多様であり、それらを包括した意味で委託開発と言っているのである。受注制作に関しては、それでも自社開発と区分して、委託（丸投げ）を別立てとしたのは、企業会計基準委員会(2006)の趣旨を継承し、収益の総額表示と純額表示を区別するためである。また、自社利用に関して、購入と自社開発を区分するのは、パッケージ・ソフトウェアの購入（カスタマイズを含む）又はプライベート・クラウドの利用と、カスタム・メイド（オーダー・メイド）の開発とでは、金額的な会計処理としては違いがないが、内訳情報としては企業のソフトウェアに対するスタンスの開示として有意な違いがあるからである。市場販売目的又はサービス提供に関しても、自社開発以外に購入がありうるのである。

⑥費用処理／資産計上という会計処理としては、受注制作だけが費用処理であり、それ以外は全て資産計上とする。しかも、全て同一の資産要件、資産範囲での会計処理として統一する。敢えて違ったものとする特段の事由はないからである。

⑦そして、これらの会計処理によって行なわれる財務報告は、例えば国民経済計算の「基礎統計」として遺漏のないものとして、且つ多段階の推計という現行の国民経済計算の精度とは比較にならない信頼しうる「基礎統計」たりうるのである。

---

[123] 少し古く、またアメリカの事例であるが、Hagel Ⅲ(2002)でＡＳＰ(Application Service Provider、クラウドの前身と言ってよいだろう)が取り上げられており、その分類の中に、「新しいアプリケーションを開発し、インターネットを活用して潜在的な顧客を開拓する」「ＡＳＰデベロッパー」(邦訳 p. 80)だけでなく、自ら開発せず「リセラー・サービス企業として大手ベンダーの企業アプリケーションを利用し」「新しいインターネット・ベースの流通経路を開発し、中小企業に大手ベンダーのアプリケーションをより低コストで提供・サポートするサービスに特化した」「ＡＳＰリセラー」(邦訳 p. 79)を加えている。日本の現行会計基準では、サービス提供は専ら自社開発した場合だけを想定しているが、それと異なる場合も出現しているのである。

補遺　ソフトウェア分類別のライフサイクルへの関与

ここで補遺として、ソフトウェアの分類別に、該当企業がライフサイクルにどのように関与するかを整理しておくことにする。図表に纏めたので、それに沿って、考察する。

図表B－3-11　ソフトウェア分類別のライフサイクルへの関与

ソフトウェア分類	該当企業	企画	開発	保守	運用	廃棄
市場販売目的	販売企業	○	○	○	×	○
	利用企業	× △１	×	× △２	○	○ ○１
サービス提供	提供企業	○	○	○	○	○
	利用企業	× △３	×	× △４	× △５	×
自社利用	利用企業	○	○	○	○	○

○１：販売企業の廃棄の影響（保守等サービス停止）を受けるが、タイミングは同期するとは限らないし、作業は独自に行なう。
△１：企画検討を行ない、その結果購入を選択する場合のことだが、但し当該ソフトウェアの企画に関与するわけではない。
△２：購入時のカスタマイズないし独自仕様の変更を許容される場合がある。但し、それを行なうのは一般的には販売企業側であり、利用企業が行なえるのは「売り切り制」の場合に限られる。
△３：△１と同様、企画検討を行ない、その結果サービス利用を選択する場合のことだが、但し当該ソフトウェアの企画に関与するわけではない。
△４：△２と同様、サービス利用開始時のカスタマイズないし独自仕様の変更を許容される場合がある。
△５：利用と運用は別のことだが、利用の中に一部運用が含まれる場合がある。

（出典：筆者作成）

まず特徴的なことを挙示する。第１に、ソフトウェアの分類別に該当企業のライフサイクルへの関与がかなり異なることである。第２に、販売・提供企業はライフサイクル全域にほぼ関与するが、市場販売目的のソフトウェアの場合は当該ソフトウェアの運用は行なわない。「使い勝手」の確認や改良のために、自社でも利用し、従って運用を行なうことがあるが、それは目的はどうあれ、主として自社利用のソフトウェアと見做すべきことであって、販売企業として市場販売目的のソフトウェアのライフサイクルへの必当然のあるいは一貫的な関与と見做すことはできず、せいぜい副次的な関与に留まる。それに対し、自社開発したソフトウェアを資源としてサービス提供をする場合、運用も行なうので、収益の直接的源泉である点では同じ市場販売目的のソフトウェアと異なるのである（但し、今日ではクラウド事業等で同一のソフトウェアを両方の営業対象とすることも少なくない）。第３に、利用企業は、自社開発し自社利用する場合には、サービス提供のソフトウェアの提供企業と同様、ライフサイクルの全域に関与するが、それ以外では関与が少ないのである。市場販売目的のソフトウェアを購入する場合には運用と廃棄にしか関与しないし、サービス提供を享受する場合には利用するだけなので、ライフサイクル全域に関与しない。利用と運用は、境界領域はあるが、業務領域や作業の当事主体が異なるので混同してはならない。なお、図表で注記した微妙な

あるいはレアケースとも言うべき事態に関しては、本文では再言しない。

図表B-3-12　受注制作のライフサイクルへの関与

ソフトウェア分類	該当企業	企画	開発	保守	運用	廃棄
受注制作	受注企業	△ △1	○	△ △2	△ △3	△ △4

△１：企画段階から参画する場合があるが、必ずというわけではない。
　　委託企業が企画と開発を別の企業に委託することは少なくないからである。
△２：開発の後、引き続き保守を受託する場合が少なくないが、必ずというわけではない。
　　委託企業は、新規開発は多大な工数が掛かるので委託するが、保守は自社で行なう
　　こともあるし、あるいは保守は別の企業に委託する場合もあるからである。
△３：△２と同様、開発の後、引き続き運用を受託する場合があるが、必ずというわけではない。
　　委託企業は、新規開発は多大な工数が掛かるので委託するが、運用は自社で行なう
　　こともあるし、あるいは運用は一括して別の企業に委託する場合もあるからである。
△４：運用を受託している場合に、その延長上で廃棄作業を任される場合は多い。
　　但し、廃棄の意思決定等は委託企業が行なう。

（出典：筆者作成）

資産認識という点では、受注制作（受託開発）のソフトウェアは専ら費用処理対象なので、対象外のため、前記の図表では除外したが、同じようにどのような関与をしているか、確認しておく。一見してわかるように、受注制作（受注企業）の場合、ライフサイクルへの関与は部分的で、非一貫的なことが最大の特徴である。開発以外の大フェーズに関与することもあるけれども、概ね偶発的ないし条件限定的である。委託企業側の事情に圧倒的に依存しているのである。特に委託企業が大企業の場合、企画はコンサルタント会社等に委託し、開発はそれとは別の大手ソフトウェア企業にコンペ形式で委託し､保守は自社ないし継続的に委託している特定のソフトウェア企業に委託し、運用は一括的に特定の１社に全て委託するといったことを定型的な業務パターンとしていることが比較的多いのである。また、受注企業（受託企業）側にとっては、個々のソフトウェアに関しては部分的な関与だが、複数企業の複数ソフトウェアに関して開発又は保守又は運用等に多数関与するという並列的な態様で事業運営を行なっている。そうした意味で、多数のソフトウェアを包含して総合的にはライフサイクル全般に関与しているが、特定のソフトウェアに関しては部分的な関与が多いと言えるのである。

本論

第１章　ソフトウェア企画

１．ソフトウェア企画の概観

（1）ＪＩＰＤＥＣ「システム管理基準」における企画

（2）ＩＰＡ「共通フレーム」における企画

（3）プロダクト企画の案件対応

（4）ソフトウェア企画（業務）の全体像

２．ソフトウェア企画における各種ドキュメント

（1）ＲＦＰと提案書

（2）企画書

（3）各種標準

（4）各種規程

（5）システム戦略並びに全体構想に関するドキュメント

３．ソフトウェア企画に係る会計処理

（1）現行の会計処理

（2）ソフトウェア企画のドキュメント資産に適合的な会計処理

第１章　ソフトウェア企画

１．ソフトウェア企画の概観

ソフトウェアのライフサイクルに沿った本論の先頭（第１章）として、現行のソフトウェア会計基準では、全く対象外とされているソフトウェアの企画を取り上げる。対象外とはされているが、類推的に全て費用処理の対象であると想定されていることはほぼ間違いないであろう。ソフトウェアの企画には、全く資産性がないと暗黙的に看做されており、会計実務においても、そのように処理されているであろう。しかし、果たしてそれでよいのか、資産性がないと看做すことが妥当か、筆者には疑問である。もちろん、慎重な取り扱いが必要であり、対象範囲を限定しなければならないが、ソフトウェア企画の一部には開発等と同様に資産性を有する事象が存在すると考える。それを明確化するために考察する。

まずは、ソフトウェアの企画とはどのようなことか、概観する。

（1）ＪＩＰＤＥＣ「システム管理基準」における企画

最初に取り上げるのは、ＪＩＰＤＥＣ「システム管理基準」であり、その中で企画がどのように捉えられているか、確認する。

「企画業務」は、中項目として「１．開発計画」、「２．分析」、「３．調達」の３項目で構成され、各々小項目は９項目、８項目、６項目、計 23 項目で構成されている[124]。何よりも特徴的なことは、「全体最適化計画との整合性を考慮」（Ⅱ．１．(2)）したり、「運用及び保守」を関わらせたりはしているが（Ⅱ．１．(6)、２．(1),(3)、３．(1)）、専ら開発に関する企画だけを取り扱っていることである。「情報システムの目的を達成する実現可能な代替案を作成し、検討すること」としながら、それは「開発計画の策定に当たっては」（Ⅱ．１．(9)）という枠内のことであり、「代替案」として購入は一応想定されてはいるが[125]、保守や対処しないということは全く想定されていない。まして、運用に関する企画は、全く想定外である。これでは、企画の全体に対しては「システム管理基準」として部分的と言わざるを得ないものである。

なお、企画において作成するドキュメントとして想定しているのは、「開発計画書」[126]と「ユーザニーズの調査結果」[127]である。

---

[124] 経済産業省商務情報政策局(2005)pp. 6, 107-144

[125] 「パッケージソフトウェアの使用に当たっては、ユーザニーズとの適合性を検討すること」（Ⅱ．２．(8)、pp. 6, 134-135)、とある。

[126] 同書 pp. 109, 112

[127] 「ユーザニーズは文書化し、ユーザ部門の責任者が確認することが必要である」（Ⅱ．２．(4)、p. 129)とあり、他の諸項目と違って、特に「文書化」が明記されている。

（2）ＩＰＡ「共通フレーム」における企画

次に取り上げるのは、ＩＰＡ「共通フレーム」であり、その中で企画がどのように捉えられているか、確認する。

最初の 1994 年版「共通フレーム」では、「企画プロセス」は、「3.1　企画プロセス開始の準備」、「3.2　情報化戦略の立案」、「3.3　情報システム構想の立案」、「3.4　システム化計画の立案」という４つのアクティビティで構成され、各々のアクティビティは 3、9、9、11 タスクで構成されている[128]。特徴的なことは、想定している企画が、「現行情報システム」に対して（3.2.3、3.2.7）[129]、それに替わる新システムの「情報化戦略」、「情報システム構想」、「システム化計画」の立案だということである（筆者のいう「既存システムの全面再構築案件」に相当すると言えようか）。そのように特定し、他のケースは考慮していない。系統的な内容になっているので[130]、応用できるとは言え、やはり１つの特定の想定であることは否めない。

なお、企画において作成するドキュメントとして想定しているのは、「情報化戦略指針書」[131]と「情報システム構想企画書」[132]と「システム化計画書」[133]である。

次の 1998 年版「共通フレーム」では、「企画プロセス」は、「1.3.1　プロセス開始の準備」、「1.3.2 情報戦略の立案」、「1.3.3　情報システム構想の立案」、「1.3.4　システム計画の立案」という４つのアクティビティで構成され、各々のアクティビティは 3、9、9、12 タスクで構成されており[134]、多少の異同はあるが、1994 年版と大差ないものである。但し、「共通フレームの企画プロセスは情報システムの大規模な企画を想定している」と明記し、「計測・制御用システム、組込みシステム、通信システム、小規模なシステムでは、「業務」を「機能など」と読み替えて必要に応じてアクティ

---

128 共通フレーム検討委員会編(1994)pp. 44-49

129 同書 pp. 45, 46

130 「3.2　情報化戦略の立案」は「3.2.1　ビジネス環境の調査・分析」、「3.2.2　現行業務の調査・分析」、「3.2.3　現行情報システムの調査・分析」、「3.2.4　情報技術動向の調査・分析」、「3.2.5 基本戦略の策定」、「3.2.6　新業務イメージの作成」、「3.2.7　情報化対象の選定と投資目標の策定」、「3.2.8　情報化戦略指針書」、「3.2.9　情報化推進体制の確立」、「3.3　情報システム構想の立案」は「3.3.1　対象業務の概要調査」、「3.3.2　対象業務の詳細調査」、「3.3.3　対象業務に係わる情報システムの分析」、「3.3.4　適用情報技術の調査」、「3.3.5　新業務モデルの作成」、「3.3.6　システム基本構想の策定」、「3.3.7　業務改善効果の予測」、「3.3.8　情報化戦略との検証」、「3.3.9　情報システム構想企画書の作成と承認」、「3.4　システム化計画の立案」は「3.4.1　情報システム化構想の確認」、「3.4.2　実現可能性の検討」、「3.4.3　開発スケジュールの大枠作成」、「3.4.4　機種選定方針の策定」、「3.4.5　開発条件の明確化」、「3.4.6　移行運用保守に対する基本方針の明確化」、「3.4.7　開発環境に対する基本方針の明確化」、「3.4.8　教育訓練に対する基本方針の明確化」、「3.4.9　品質に対する基本方針の明確化」、「3.4.10　システム化計画書の作成と承認」、「3.4.11 プロジェクト計画の立案と承認」となっており(pp. 44-49)、後続の「システム管理基準」より遥かに系統的且つ具体的である。

131 同書 p. 46

132 同書 p. 47

133 同書 p. 49

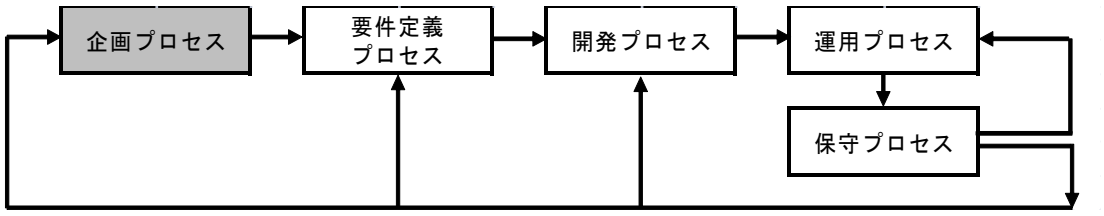
134 SLCP-JCF98 委員会編(1998)pp. 131-138

ビティ・タスクを利用するとよい」[135]とガイダンスしている。1994 年版にはなかった想定の明示は親切ではあるが、企画が開発しか射程に入れていないことは改められていない。

なお、企画において作成するドキュメントとして想定しているのは、「情報戦略」[136]と「システム構想」[137]と「システム計画」[138]である。

３番目の 2007 年版「共通フレーム」では、「企画プロセス」は、「1.4.1　プロセス開始の準備」、「1.4.2　システム化構想の立案」、「1.4.3　システム化計画の立案」という３つのアクティビティで構成され、各々のアクティビティは 4、9、18 タスクで構成されており[139]、1998 年版に比べアクティビティが１つ減ったが(戦略の立案)、タスクは２つ減っただけなので、編成替えはなされたが、実質的なタスク・レベルの内容としては大差ないものである。但し、企画プロセスの説明として、「共通フレームの企画プロセスは、新規事業の立ち上げや現行事業の大幅な見直し等に伴い、システムの新規取得又は現行システムの改良について検討する際のアクティビティ及びタスクを定義している。企画プロセスは要件定義プロセスや開発プロセスに先立ち実施することを想定しているが、事業の見直しを伴わない小規模なシステム改良、目的や用途が明確になっているシステム（計測、制御用システム、組込みシステム、通信システム等）では、本プロセスを省略して、要件定義プロセスや開発プロセスを選択してもよい」[140]、としており、個々のタスク内容は 1998 年版と大差ないが、企画の位置付けに関してはかなり射程が拡がり、且つより明確になっている。それでも自ら限定してしまっていることが惜しまれる。その点を明確にするために、掲示されている「企画プロセスの位置づけ」の図を転載する。

図表１－１-1　2007 年「共通フレーム」における企画プロセスの位置づけ



（出典：ＩＰＡ編(2009)p. 247）

「保守プロセス」からのフィードバック・ループは描かれている（捉えられている）が、「企画プロセス」は（「要件定義プロセス」を含めた[141]）「開発プロセス」へと到る「起点」的な位置付けと

---

135 同書 p. 131
136 同書 p. 133
137 同書 p. 135
138 同書 p. 138
139 ＩＰＡ編(2009)pp. 101-109。なお、2007 年版に関して、第２版の方を使用したが、「第２版における主な改訂内容」として「(1)企画プロセス及び要件定義プロセスの整備」が挙げられており、具体的には「・アクティビティ又はタスク間の記述内容の粒度のバラツキ、タスク同士の結び付きを見直し強化」(P. viii)、となっているからである。第１版と第２版の異同の指摘は、省略する。
140 同書 pp. 101-102
141 「要件定義プロセス」を「開発プロセス」とは分離し、独立的なプロセスとしているのは、当該

なっているだけで、企画が開発だけではなく、保守並びに運用に関する企画を行なうことが捉えられていない（保守に関しては、図示されてはいないが、上掲の引用にあるように本文では触れられているが)。更には、「システム戦略」ということで、総合的な企画があることも的確に位置付けられていない。

なお、企画において作成するドキュメントとして想定しているのは、「企画プロセス実施計画」[142]と「システム化構想」[143]と「システム化計画」[144]である。

最新版の 2013 年版「共通フレーム」では、「企画プロセス」は、「2.1.1　システム化構想の立案プロセス」、「2.1.2　システム化計画の立案プロセス」という２つの小プロセス、その各々は 3、3 アクティビティで構成され、各々のアクティビティは 4、7、2、4、16、2 タスクで構成されており、2007 年版に比べ「プロセス開始の準備」が独立的ではなく、小プロセス個々に設定され、各小プロセスが「プロセス開始の準備」と「立案」と「承認」という同じアクティビティ構成となり[145]、より形式化を強めた編成替えがなされている。実質的なタスクは 7 と 16 の計 23 であり、大筋の内容はほとんど変わっていない[146]。また、「企画プロセス」の位置付けに関しても、2007 年版とほとんど同じ説明がなされており（但し図はなくなっている）[147]、進展・改善はなく、同様の問題を抱えたままである。

なお、企画において作成するドキュメントとして想定しているのは、「(企画) プロセス実施計画」[148]と「システム化構想」[149]と「システム化計画」[150]である。

以上を纏めて一覧化すると、次の図表のようになる。システム管理基準に「調達」という特異なアクティビティがあるのを除けば、大同小異と言ってよい。「戦略」を明示的に独立化させるか、「構想」の中に含めるかといった違いはあるが、それ以外は類同的である。ドキュメントにしても、名称に若干の差異はあるが、実質的にはほぼ同様なものと言える。

プロセスを重要視し、「要件定義」を充実させることが開発を成功に導くものだという意図を体現したものであり、その意図は諒とするが、開発することが既決事項であり、それを前提としたプロセスという意味では、「開発プロセス」の最初の小プロセスと位置付けるのと実質的な違いはそれほどない。特に異議を唱えないが、賛成もしない。何れが効果的かは、経験則的に選別されていくであろう。

142 同書 p. 102
143 同書 p. 104
144 同書 p. 107
145 ＩＰＡ監修(2013)pp. 114-126
146 「共通フレーム 2013 における主な改訂内容」でも企画プロセスに関しては挙げられていないし(P. vi～vii)、付 2-2「共通フレーム 2013、共通フレーム 2007(第 2 版)と ISO/IEC 12207:2008(JIS X 0160:2012)の対比」(pp. 58-60)においても少なくともタスク項目としてはほとんど異同がないことから確認できる。
147 同書 pp. 115, 303
148 同書 pp. 116, 121
149 同書 p. 118
150 同書 p. 125

図表１－１-2　企画アクテイビティ並びにドキュメント一覧

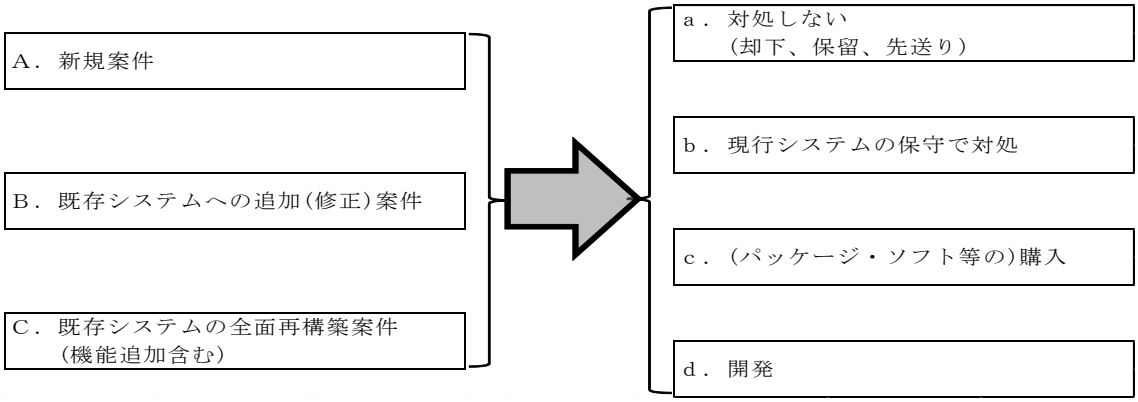
	システム管理基準	共通フレーム			
		1994年版	1998年版	2007年版	2013年版
アクティビティ	開発計画 分析 調達	企画プロセス開始の準備 情報化戦略の立案 情報システム構想の立案 システム化計画の立案	プロセス開始の準備 情報戦略の立案 情報システム構想の立案 システム計画の立案	プロセス開始の準備 システム化構想の立案 システム化計画の立案	システム化構想の立案 システム化計画の立案
ドキュメント	開発計画書 ユーザニーズの調査結果	情報化戦略指南書 情報システム構想企画書 システム化計画書	情報戦略 システム構想 システム計画	企画プロセス実施計画 システム化構想 システム化計画	プロセス実施計画 システム化構想 システム化計画

（出典：『システム管理基準』並びに『共通フレーム』各版から項目抽出、表化筆者）

（3）プロダクト企画の案件対応

これに関しては、ソフトウェア基礎論第２章「ソフトウェア・ライフサイクル」でも触れたが、補足的に説示しておきたい。

図表１－１-3　プロダクト企画の案件対応



（出典：筆者作成）

プロダクト企画の対象は大別すると（ここでは一応開発に即した範囲に限定してのことだが）、A．全くの新規案件か／B．既存システムへの追加（修正）案件か／C．既存システムの全面再構築（機能追加を含む）案件に分けられる。もちろん、「全面再構築」と言っても、既存システムを一部そのまま存続させる等、截然といずれかに分けられるとは限らない場合もあるが、一応の区分としては有用である。そして、検討結果として、ａ．対処しない（却下、保留、先送り）／ｂ．現行システムの保守で対処／ｃ．（パッケージ・ソフト等の）購入／ｄ．開発、という選択肢（代替策）があるが、各案件区分に対してほぼいずれでの対応もあり得る。A．に対して、ｃ．又はｄ．となる可能性が高いかもしれないが、予算の手立てが付かないか、必要性が薄弱ということで、ａ．となることもあれば、構想を縮小するなり、部分的な対処ということでｂ．となることもあろう。B．に対しては、ｂ．となる可能性が最も高いかもしれないが、予想外に変更が大きいので、ｃ．ないしｄ．となるかもしれないし、予算の都合等でａ．となることもあろう。C．に対しては、ｃ．ないしｄ．となる可能性が高いかもしれないが、予算の手立てが付かないか、時期尚早等の理由でａ．となる

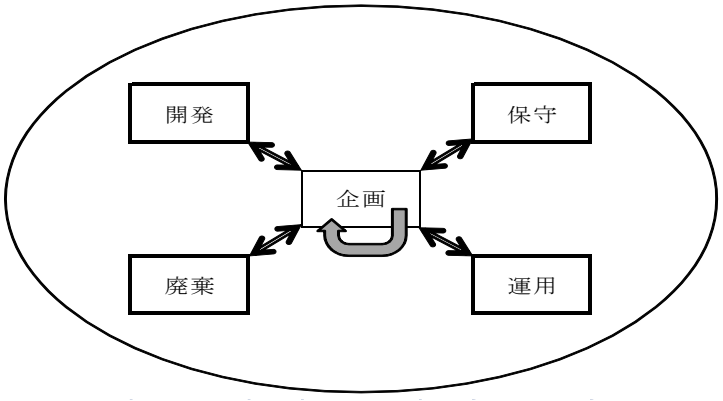
か、あるいは限定的にｂ．で対処することもあり得る。

その検討結果をドキュメントとして纏めたものが企画書であり、あるいはその前段階として作成するのがＲＦＰである。

（4）ソフトウェア企画（業務）の全体像

以上のことを更に敷衍すると、ソフトウェアの企画は、開発に関する企画が例え多数を占めるとしても、運用や保守（そして廃棄）に関する企画もあるし、戦略や全体構想という自己言及的（自己回帰的）な企画もあるのである。「共通フレーム」は戦略に関しては明示的に取り上げている。従って、2007 年版「共通フレーム」で図示されているようなサイクリックな「起点」といった捉え方ではなく（内容と適合していない）、ハブ・アンド・スポーク的な企画・開発・保守・運用・廃棄のいずれに関する企画もあるというように捉えることが実態に適っていると言える。それを図示すると、次のようになる。

図表 1－1-4　ソフトウェア企画（業務）の全体像



（出典：筆者作成）

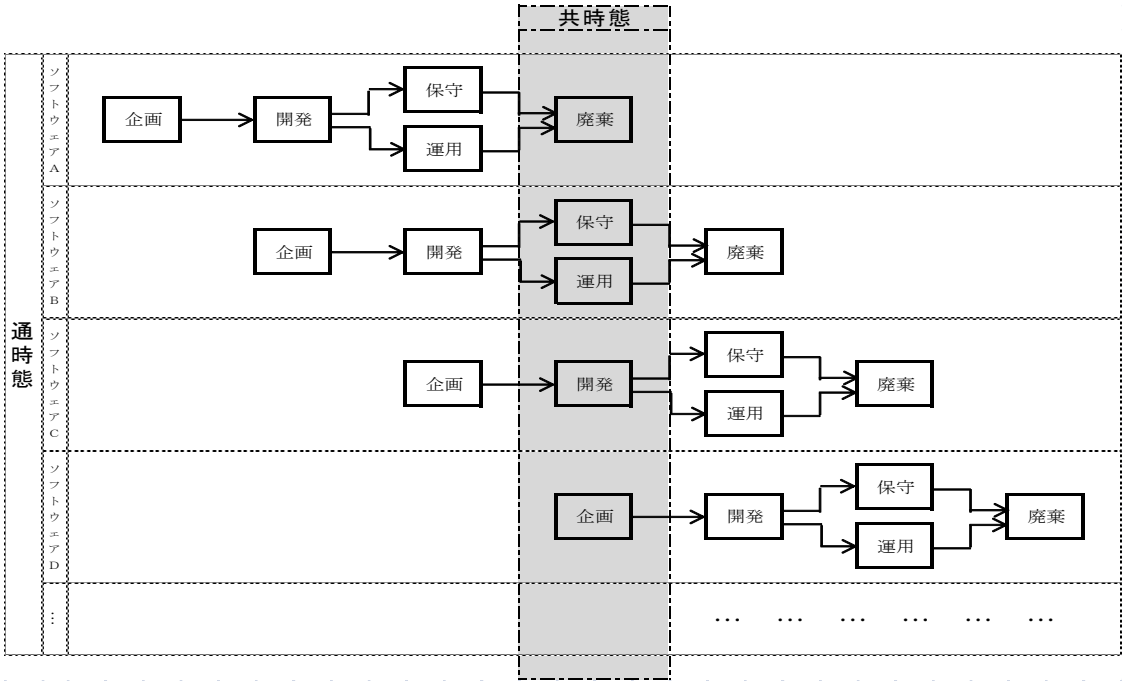
そうすると、これまでソフトウェアのライフサイクルとして、企画に始まり廃棄で終わる捉え方をしてきたが、それを変更する必要があるのだろうか。そうではないのである。

構造言語学[151]における通時態（diachrony）と共時態（synchrony）という概念を援用するとわかりやすいのだが、個々のソフトウェアを通時的に捉えれば、企画から始まり廃棄で終わることは何ら変わりはないのである。そして、複数のソフトウェアは各々異なる時期に出現し、また寿命も異なる場合がある。それらを全体的に時系列的に捉えた場合が通時態である。それに対して、特定の

[151] 構造言語学は、フェルデナン・ド・ソシュール(Ferdinand de Saussure)に始まる言語学の大きな潮流であり、通時態/共時態以外にも、ラング(langue) /パロール(parole)、シニフィアン(signifiant)/シニフィエ(signifié)という独特の概念を設定し、言語理論を構築している。なお、弟子のシャルル・バイイ(Charles Bally)とアルベール・セシュエ(Albert Schehaye)が編集した『一般言語学講義』(Cours de linguistique générale)がソシュールの言語観を不正確にしか伝えていないということで、1950 年代以降新たなソシュール研究がなされ、見直しがなされているが、ここでの援用には影響しないので、立ち入らない。

時点で同時的に複数の異なる位相にあるソフトウェアを捉えた場合が共時態である。そして、共時態において、異なる位相にあるソフトウェアに対して企画が関わることがあり、その場合には開発の前段階としての企画であったり、保守に関する企画であったり、運用に関する企画であったり、廃棄に関する企画であったりするのである。あるいは、個々のソフトウェアではなく、戦略や全体的な構想という企画に関する企画である場合もある。あるいは、開発を予め想定していたが、予算等の都合で大幅な保守で対応するなり、開発をせずにパッケージ・ソフトを購入することにしたり、当面は現行システムのままで新たな開発（購入）又は保守対応をしないことにする場合もある。

図表 1－1－5　ソフトウェア企画の態様（通時態と共時態）



（出典：筆者作成）

更に、（3）ではプロダクト企画の案件対応を捕捉したが、企画にはプロダクト企画だけではなく、プロセス企画もある。各種の標準や規程といった、企画・開発・保守・運用・廃棄の作業の進め方や手順等の規則を検討し、制定するという企画である（標準や規程の中にはプロダクトに関するものもあるが、どのように行なうかというプロセスに関するものも少なくない）。

これらが、ソフトウェア企画（業務）の全体像である。

2．ソフトウェア企画における各種ドキュメント

ソフトウェアの企画を総合的に、又より具体的に捕捉するために、続いて企画においてどのようなドキュメントを作成するか、確認していくことにする。予めその準備として、ＪＩＰＤＥＣの「システム管理基準」やＩＰＡの「共通フレーム」における企画の内容を確認する過程で、その各々が想定しているドキュメントは抽出しておいた。それらが参考になる。

図表１−１−６　ＲＦＰと提案書のサンプル

提案依頼書（ＲＦＰ）

０．はじめに
１．提案依頼の趣旨
２．システムに求める機能
　２－１．業務要求
　　(1)業務基本要求
　　(2)業務機能要求
　　(3)業務フロー要求
　　(4)画面要求
　　(5)帳票要求
　２－２．技術要求
　　(1)ＩＴインフラ要求
　　(2)システム能力要求
　　(3)障害対策要求
　　(4)セキュリティ要求
　２－３．運用要求（広義）
　　(1)運用要求（狭義）
　　(2)保守要求
　　(3)教育・研修要求
　　(4)移行要求
　　(5)アウトソース要求
３．予算
４．プロジェクト期間
５．特記事項

（出典：永井昭弘(2005)pp. 130-143、一部改変）

提案書

１．提案の趣旨
２．提案対象領域
３．システム構成
　(1)システム概念図
　(2)ネットワーク構成図
　(3)ハードウェア構成図
　(4)ソフトウェア構成図
４．プロジェクトの内容
　(1)プロジェクトの作業
　(2)貴社と当社の役割（作業別）
　(3)成果物（作業別）
５．運用体制
　(1)運用支援サービス
　(2)研修サービス
６．プロジェクト体制
７．スケジュール
８．お見積もり
９．特記事項

（出典：永井昭弘(2005)pp. 144-151、一部改変）

提案依頼書（ＲＦＰ）

１．会社の概要
　(1)会社の概要
　(2)組織
　(3)グループ会社の状況
２．システム化の概要
　(1)システム化の背景
　(2)システム化の目的
　(3)システム化の方針
　(4)解決したい課題
　(5)狙いとする効果
　(6)システムの利用者
　(7)社内のＩＴ化の状況
　(8)予算
３．提案の依頼事項
　(1)システム化対象範囲
　(2)現行業務の現状と改善要望
　(3)システム構成要件
　(4)ネットワーク構成要件
　(5)品質・性能要件
　(6)セキュリティ要件
　(7)運用要件
　(8)保守要件
４．開発計画
　(1)スケジュール
　(2)納品条件
　(3)プロジェクト管理方針
　(4)開発推進体制
　(5)役割分担
　(6)開発場所
　(7)そのほか
５．テスト・移行・教育
　(1)テスト
　(2)移行
　(3)教育
６．契約事項
　(1)発注形態
　(2)検収
　(3)支払条件
　(4)保証年数（瑕疵担保責任期間）
　(5)機密事項
　(6)著作権など
　(7)そのほか
７．提案書への記載事項
８．提案手続およびスケジュール
　(1)提案書の提出
　(2)提案のプレゼンテーション
　(3)貴社製品のデモンストレーションとＱ＆Ａ
　(4)プロジェクトの発足
　(5)問い合わせ
　(6)担当
　(7)そのほか

（出典：広川敬祐編著(2011)pp. 204-206、一部改変）

（1）ＲＦＰと提案書

真っ先に挙げられるのは、プロダクトに関わり、且つ開発に関わるＲＦＰ(Request For Proposal：

提案依頼書）である。「共通フレーム」等には挙示されていないが、ソフトウェア実務では欠かせないものである。ＲＦＰは、多くの業界・業種で使われており、大枠は共通しているが、それぞれの分野特有の情報が凝縮されており、ソフトウェア業界のＲＦＰも同様である。ユーザー企業等が、複数のソフトウェア企業を候補とし入札形式で業者を選定する場合に発行して、業者にシステム提案を求めるものである。業務（取引）の手順は次の通りである。（ＲＦＩ（Request For Information：情報提供依頼書）[152]→回答→）ＲＦＰ→提案書→業者選定→契約→開発（又は購入）。業者選定に到るまでには、質疑応答やプレゼンテーションないしデモあるいは双方の視察等、様々なやりとりがなされる場合があるが、中心的なものは文書としてのＲＦＰ並びに提案書である。通常は各々数ページから数十ページ、多ければ数百ページ（付録資料を含め）くらいのコンパクトなもので、それ故に必要十分な情報が凝縮されているものである。何よりもソフトウェア業界で広範囲に使われており、これに基づいて開発が行なわれるエッセンスとも言うべき情報であることに意義がある[153]。

ＲＦＰ並びに提案書から、社内稟議を経てのことだが、即座に開発に進む場合があるが、その社内稟議のために業者からの提案書に簡単な頭書を付加する程度の場合もあれば、提案書をベースに依頼企業側で企画書を編纂する場合もある。あるいは、依頼企業と提案企業が共同で、提案書をベースにしてのことだが、再度より実現性の高い内容に洗練させて企画書を作成し直す場合もある。

ＲＦＰと提案書のサンプルは、次のようなものである。大差はないが、やや簡略なものと、より詳細なものとの２種類を掲示する。

依頼された業者側が作成し、提出する提案書は、基本的には、ＲＦＰの項目に沿って、どのような内容で対応するかを記述したものである。契約的な諸事項や諸条件（例えば新たな要件が発生した場合の取り扱い、規模が増大した場合の取り扱い等）を含め、一般的にはＲＦＰの数倍程度のボリュームのものとなる。

これらを見てわかることは、１.（１）、（２）で概観した企画のアクティビティないしタスクと全般的によく対応していることである。ＲＦＰの場合、未だ提案を依頼する段階であり、また内容の精細度は様々ではあるが、それ相当の調査・分析・検討をしなければ、作成できないのであり、企画の作業を行なっているのである。その段階で既に業者側が参画する場合もあり得る。提案書に関しても、作業的にはほぼ同様なことを行なう。提案書がＲＦＰと異なるのは、比較的短期間に作成しなければならないことである。従って、主としてＲＦＰに依拠し、且つ企画業務の蓄積並びに類

---

[152] ＲＦＩ(Request For Information：情報提供依頼書)は準備段階で技術情報等が不足している場合に発行するものだが、必ず発行するとは限らないのでカッコで括った(Bud Porter(2002)邦訳 p.6 等)。

[153] 官公庁では随意契約はごく限られた範囲での適用となっており、指名入札ないし一般競争入札が圧倒的に多く、民間ではそこまで多くはないが、比較的規模の大きい開発案件では入札形式が一般的と言える。ＲＦＰ作成にはそれ相応のシステムに熟達した組織能力を必要とするので、内容的に問題含みで開発時に紛糾要因となったり、自社だけでは作成できず、ＩＴコンサルタントに依頼ないし協力を仰がねばならなかったり等の問題があるが(Bud Porter(2002)、永井(2005)、永井(2011)、広川編著(2011)参照)、論旨に大きな影響を及ぼすことではないので、立ち入らない。

似案件の実績等がないと、対応は容易ではないのである。

（2）企画書

企画書は、内容的には提案書と大凡同様なものである。異なるのは、提案書が依頼された業者側が作成するものであるのに対して、企画書は当事者である企業が作成するということである。その際、自社だけで出来ない場合には、業者にコンサルテーションを委託し（準委任、派遣等を含む）、主として業者側が作成することもあり得る。いずれにせよ、企画書を作成するということは、社内稟議を経てのことではあるが、開発等を行なうことを決定するということである。これまでの説明は大凡開発に沿ったものであったかもしれないが、対象案件としては比較的規模の大きい保守や運用に関することもある。当面何もしないという結論を出す場合にも、企画書として纏めることが全くないわけではないが、その場合には途中で作成した各種資料に見送りとする頭書程度を付加して、企画書という形にしない場合の方が多いであろう。

なお、案件が開発の場合、企画書とは別に独立的なより詳細な開発計画書を作成する場合もあれば、企画書に掲示したサンプルにあるように開発計画の内容も盛り込む場合とがある。

（3）各種標準

各種標準は、主としてプロダクトに関する様々な標準を定め、それをドキュメントの形で纏めたものである。

代表的なものは、プログラム標準又はコーディング規約である。制定の目的は、開発プロジェクトでは多数の開発要員が参加するので、個人差や属人性を極力排し、可能な限り均質的なプログラムを作成すること、わかりやすいものにすることでバグの作りこみを防止し、また保守性（保守のしやすさ）を担保すること等である。そのため、記述の仕方、コメントの付け方、使用する命令語の制限、規模の制限等々を定めるのである。その場合、例えば規模に関して、大凡の目安とするか、非常に厳格に僅かな行数超過でも、１つにした方が纏まりがよくても、分割するといった適用とするかは様々である。

他に、命名規則（各種資源に関する命名の仕方を定めたもの）、ドキュメント標準（各工程等でどのような形式と内容のドキュメントを作成するかを定めたもの）、テスト標準（単体・結合・総合テスト別に、テスト・ケースの設定、テストの入出力、テスト方法等を定めたもの）等がある。

これらは、例えば開発に関する各種標準で言えば、実際の適用はもちろんのこと、制定・作成も開発部門が行なうこともあるが（その方が多いかもしれないが）、性格的には企画の作業というべきである。また、実際の適用に際して、特に大規模開発プロジェクトでは、全社標準（社内標準）をベースに、一部アレンジし、独自に設定することもしばしばある。

（4）各種規程

各種規程は、主としてプロセスに関する様々な規則を定め、それをドキュメントの形で纏めたものである。

代表的なものは、プロジェクト管理規程である。制定の目的は、開発プロジェクトはアドホックに編成されるが、開発諸条件やプロジェクト・マネージャーの個人的能力等に極力左右されずに、可能な限り等質的なプロジェクト運営を行ない、プロジェクトの失敗を防止・回避することである。そのため、開発方式、開発方法、進捗管理等の進め方等々を定めるのである。今日では、プロジェクト管理ツールを採用することが多いから、それの利用方法等を含めることも多い。

他に、品質管理規程（品質管理の方法、品質目標値、出荷判定基準等を定めたもの）、見積り規約（見積り方法、難易度等の係数設定、アローワンスの設定、変動事象に対する前提あるいは追加条件設定等を定めたもの）等がある。

これらは、各種標準と同様に、例えば開発に関する各種規程で言えば、実際の適用はもちろんのこと、制定・作成も開発部門が行なうこともあるが（その方が多いかもしれないが）、性格的には企画の作業というべきである。また、実際の適用に際して、特に大規模開発プロジェクトでは、全社規程（社内規程）をベースに、一部アレンジし、独自に設定することもしばしばある。

（5）システム戦略並びに全体構想に関するドキュメント

システム戦略等は、特に大規模な開発案件等の場合には、個別案件においても検討され策定されるが、そして1.（1）、（2）で概観した中ではそうした性格のものが取り上げられていたが、確かにそうしたことの方が実際には多いかもしれないが、個別案件に即したものではなく、既存システム並びに今後構築していくであろうシステムを含めて、総合的なシステム戦略や全体構想を策定することもある。中長期的な経営計画の一環として位置付けられる場合もあれば、システムに関して単独で策定される場合もあり得る。

自社だけでは行ない難く、経営あるいはＩＴコンサルタント（業者）と共同で行なう場合が多いであろう。これに基づいて、開発・保守・運用・廃棄の各フェーズにおいて順次具体化していく規範ないし手引きとなる。

3．ソフトウェア企画に係る会計処理

1．並びに2．の考察を踏まえて、ソフトウェアに係る会計処理を取り上げる。まずは現行の会計処理がどのようになっているかを取り扱うが、その際現行のソフトウェア会計基準は独立的に企画に関する規定を設けておらず、それに準拠するだけでは実際の処理は行なえないので、会計実務の慣行的な処理を行なっていると解せられる。そうした会計実務に関しても、可能な限り包括的に取り上げることにする。また、現行の基準では認められていないが、企画の実態からすれば、資産計上することが適合的な事象があると筆者は考えているので、それを続けて取り上げることにする。

（1）現行の会計処理

まず一覧的な表を掲示し、それに沿って説明を行なうことにする。

図表１−１−7　ソフトウェア企画に係る現行の会計処理

<table>
<tr><th rowspan="2">企業分類</th><th colspan="2" rowspan="2">業務態様</th><th rowspan="2">収益実現有無</th><th colspan="2" rowspan="2">費用発生有無</th><th colspan="2">会計処理</th></tr>
<tr><th>収益</th><th>費用</th></tr>
<tr><td rowspan="3">ユーザ企業</td><td colspan="2">社内企画</td><td rowspan="3">実現せず</td><td colspan="2">社内費用発生</td><td></td><td>販管費</td></tr>
<tr><td colspan="2">企画コンサルテーション委託</td><td colspan="2">委託費用発生</td><td></td><td>外部委託費</td></tr>
<tr><td colspan="2">提案依頼</td><td colspan="2">無償なので発生せず</td><td></td><td>なし</td></tr>
<tr><td rowspan="6">ソフトウェア企業</td><td rowspan="4">企画の受託</td><td rowspan="2">企画コンサルテーション受託</td><td rowspan="2">有償なので実現</td><td>再委託あり</td><td>社内費用発生
委託費用発生</td><td rowspan="2">情報サービス売上</td><td>売上原価
外部委託費</td></tr>
<tr><td>再委託なし</td><td>社内費用発生
委託費用発生せず</td><td>売上原価</td></tr>
<tr><td rowspan="2">提案</td><td rowspan="2">無償なので実現せず</td><td>再委託あり</td><td>社内費用発生
委託費用発生</td><td rowspan="4"></td><td>販管費
外部委託費</td></tr>
<tr><td>再委託なし</td><td>社内費用発生
委託費用発生せず</td><td>販管費</td></tr>
<tr><td colspan="2" rowspan="2">自社製品又はサービスの企画</td><td rowspan="2">企画段階では実現せず</td><td>再委託あり</td><td>社内費用発生
委託費用発生</td><td>販管費
外部委託費</td></tr>
<tr><td>再委託なし</td><td>社内費用発生
委託費用発生せず</td><td>販管費</td></tr>
</table>

（出典：筆者作成）

ユーザ企業とソフトウェア企業ではソフトウェアに関して対称的となることが多いので、分けて捉えることにするが、業務態様の社内企画～提案依頼はソフトウェア企業の社内システムに関しても該当する。企業分類でソフトウェア企業とした方の業務態様は対顧客の事業活動（営業活動）に関するものだけとしている。なお、用語等は異なるが、大枠のソフトウェア分類はソフトウェア基礎論第３章で提示したものと同じである。

②業務態様としては、ユーザ企業に関しては社内企画と企画コンサルテーションの委託と提案依頼に分け、ソフトウェア企業に関しては大きく企画の受託と自社製品又はサービスの企画に分け、企画の受託は更に企画コンサルテーションの受託と提案に分けた。社内企画は社内だけで企画業務を行なうことであり、企画コンサルテーションの委託はそれと同等なことを能力・必要知識がない等の理由で外部委託することであるが（契約形態としては準委任、派遣等を含む）、両者を合体した形で取り組むことも少なくない。提案依頼はＲＦＰを発行して（ＲＦＰの作成は社内企画の方に該当する）、依頼業者から提案書が提示されることまでの内容である。ソフトウェア企業の企画の受託において、企画コンサルテーションの受託はユーザ企業の企画コンサルテーションの委託に対応するものであり、提案はユーザ企業の提案依頼に対応するのである。企画コンサルテーションの受託と提案の違いは、有償か無償かである。自社製品又はサービスの企画は、パッケージ・ソフトを製作し販売するとか、それを利用したＡＳＰ事業等を行なうとか、一般的なアウトソーシングといった事業に関する企画である。

③企画業務から直接的な収益が実現するかということでは、企画コンサルテーションの受託しか収益は実現しない。

④費用の発生有無は、各々の業務態様に関して当該企業で費用が発生するかどうかを捉えるものである。従って、提案依頼の場合、依頼された業者側は提案書作成の費用が発生するが、依頼元であるユーザ企業では費用は発生しないとする。ソフトウェア企業の場合に関して、再委託の有無で細区分しているのは、企画業務であっても、そうする実態が少なくないことと、費用科目が異なることになることを明確にするためである。

⑤会計処理は、収益が実現する場合の収益科目と、費用処理に関しては各々どのような費用科目で処理するかを挙示することにしている。収益が実現する場合は売上原価、実現しない場合は販管費で計上し、外部委託（再委託）している場合にはその分は外部委託費としている。

これらが、ソフトウェアの企画に係る現行の会計処理と捉えて、まず間違いないと思う。科目に関しては、多少の異同はあるであろうが、実質的には大同小異であろう。

（2）ソフトウェア企画のドキュメント資産に適合的な会計処理

現行の会計処理に対して、ソフトウェア会計基準におけるソフトウェア定義の関連文書（ドキュメント）を開発関連文書と限定せず、それを資産と見做すならば、異なった会計処理が適合的であると筆者は考える。現行の会計処理の場合と同様に、一覧的な表を掲示し、それに沿って説明を行なうことにする。

①表形式の大枠は現行の会計処理と同じであるが（従って、同様の説明は繰り返さない）、企画の採否と企画対象という２項目を追加している。企画の採否というのは、例えば新規案件の企画に関して調査・分析・検討を行なったが、結論的には見送りとした場合とか、社内稟議で棄却された場合とかで、開発等を行なわない場合は「採用せず」とし、何らの形で実施されることになった場合は「採用」とするものである。何故この（区分）項目が必要かと言うと、例え同じような工数・費用を投じ、同じような企画書が作成されても、具体的な後続の業務（開発等）に繋がらず、具体的なソフトウェアに結び付かないのでは、ソフトウェア定義のドキュメントに該当しないと見做すのが穏当と考えるからである。

②もう１つの追加項目である企画対象は､企画～廃棄のいずれに関する企画かという区分である。当該企画が複数の対象を包含したものであることもあり得るが、その場合は複数対象各々の会計処理を併せて行なうことになる。対象として、開発・保守・運用は同じ取り扱いとなるが、企画と廃棄は異なる取り扱いとなる。企画は、上記の「採用せず」とは異なり、採用はされるのだが、やはり直接的に、具体的なソフトウェアとは結び付かないので、ソフトウェア定義のドキュメントに該当しないと見做すのが穏当と考えるからである。廃棄に関しては、ドキュメントを作成し、一部の資源に関しては引き続き保存し、全く消却するわけではない場合もあるとは言え、最早通常の利用は行なわないので、全てを含めて除却処理とするのが穏当と考えるからである。企画業務として行

ない、ドキュメントを作成しても、それをソフトウェア資産と見做すことは抵抗があろう。こうしたことを区分するために、企画対象という（区分）項目を設定している。

図表 1-1-8　ソフトウェア企画に適合的な会計処理

<table>
<tr><th rowspan="2">企業分類</th><th colspan="2" rowspan="2">業務態様</th><th rowspan="2">収益実現有無</th><th colspan="2" rowspan="2">費用発生有無</th><th rowspan="2">企画採否</th><th rowspan="2">企画対象</th><th colspan="2">会計処理</th></tr>
<tr><th>収益</th><th>費用</th></tr>
<tr><td rowspan="13">ユーザ企業</td><td colspan="2" rowspan="6">社内企画</td><td rowspan="12">実現せず</td><td colspan="2" rowspan="6">社内費用発生</td><td rowspan="5">採用</td><td>企画</td><td rowspan="6"></td><td>販管費</td></tr>
<tr><td>開発</td><td>資産計上</td></tr>
<tr><td>保守</td><td>資産計上</td></tr>
<tr><td>運用</td><td>資産計上</td></tr>
<tr><td>廃棄</td><td>除却費</td></tr>
<tr><td>採用せず</td><td>全て</td><td>販管費</td></tr>
<tr><td colspan="2" rowspan="6">企画コンサルテーション委託</td><td colspan="2" rowspan="6">委託費用発生</td><td rowspan="5">採用</td><td>企画</td><td rowspan="6"></td><td>外部委託費</td></tr>
<tr><td>開発</td><td>資産計上</td></tr>
<tr><td>保守</td><td>資産計上</td></tr>
<tr><td>運用</td><td>資産計上</td></tr>
<tr><td>廃棄</td><td>除却費</td></tr>
<tr><td>採用せず</td><td>全て</td><td>外部委託費</td></tr>
<tr><td colspan="2">提案依頼</td><td colspan="2">無償なので発生せず</td><td>採否両方</td><td>全て</td><td></td><td>なし</td></tr>
<tr><td rowspan="17">ソフトウェア企業</td><td rowspan="4">企画の受託</td><td rowspan="2">企画コンサルテーション受託</td><td rowspan="2">有償なので実現</td><td>再委託あり</td><td>社内費用発生
委託費用発生</td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2">情報サービス売上</td><td>売上原価
外部委託費</td></tr>
<tr><td>再委託なし</td><td>社内費用発生
委託費用発生せず</td><td>売上原価</td></tr>
<tr><td rowspan="2">提案</td><td rowspan="2">無償なので実現せず</td><td>再委託あり</td><td>社内費用発生
委託費用発生</td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td>販管費
外部委託費</td></tr>
<tr><td>再委託なし</td><td>社内費用発生
委託費用発生せず</td><td>販管費</td></tr>
<tr><td colspan="2" rowspan="7">自社製品又はサービスの企画</td><td rowspan="7">企画段階では実現せず</td><td rowspan="3">再委託あり</td><td rowspan="3">社内費用発生
委託費用発生</td><td rowspan="4">採用</td><td>企画</td><td rowspan="7"></td><td>販管費
外部委託費</td></tr>
<tr><td>開発</td><td>資産計上</td></tr>
<tr><td>保守</td><td>資産計上</td></tr>
<tr><td>運用</td><td>資産計上</td></tr>
<tr><td>廃棄</td><td>除却費</td></tr>
<tr><td rowspan="2">再委託なし</td><td rowspan="2">社内費用発生
委託費用発生せず</td><td>採用せず</td><td>全て</td><td>販管費
外部委託費</td></tr>
<tr><td rowspan="5">採用</td><td>企画</td><td>販管費</td></tr>
<tr><td>開発</td><td>資産計上</td></tr>
<tr><td>保守</td><td>資産計上</td></tr>
<tr><td>運用</td><td>資産計上</td></tr>
<tr><td>廃棄</td><td>除却費</td></tr>
<tr><td>採用せず</td><td>全て</td><td>販管費</td></tr>
</table>

（出典：筆者作成）

③会計処理としては、企画が採用され、企画対象が開発・保守・運用のいずれかであれば、企画業務を遂行し、企画書等のドキュメントを作成したならば、それはソフトウェアとしてのドキュメントに相当し、それに投じた費用相当額をソフトウェア資産として、資産計上することが適合的であると考える。企画対象が企画の場合は、例え企画としては採用されても、それを資産計上することは拡大解釈に過ぎ、不適切であろう。廃棄の場合は、前述した通り、除却費として処理することが穏当であろう。

これまでソフトウェアの企画に関しては、会計的にはほぼ等し並みに費用処理がなされてきたと

みて間違いないであろう。会計実務でも特に疑義が呈せられてきたわけではないであろう。しかし、ソフトウェア会計基準の論理構制からすれば、不整合的である。ソフトウェアの定義として、「関連文書」を含めているのであり、ソフトウェアの構成要素としてプログラムと併せてドキュメントを含めている。それならば何故、企画におけるドキュメントはソフトウェアと見做されてこなかったのであろうか。ソフトウェア会計基準のソフトウェアの定義における「関連文書」の規定が実務指針でも簡単な例示に留まり、対象範囲が不明確だからである。尚且つ、ソフトウェア会計基準にはソフトウェア・ライフサイクルというソフトウェアを全域的に捕捉する明確な視座（基軸）が欠落しているため、開発を主要な範囲とする狭い捕捉しかしていないからである。ソフトウェアの定義を整備し[154]、ソフトウェア・ライフサイクルという視座（基軸）を据えれば、企画に関しても相応の位置付けがなされることになる。そうすれば、開発において作成される設計書等がソフトウェアたるドキュメントと見做されているのと同等に、企画において作成されるドキュメントが取り扱われることが整合的な取り扱いと言うべきであることは言を俟たない。開発において、特に設計フェーズでは企画書が依拠する最重要のドキュメントであることは紛れもないことである。従って、企画におけるドキュメントという成果物を産出するに到る過程で投入した作業に要する費用をソフトウェア資産として計上することが、適合的な会計処理である。

但し、慎重を期する限定的な取り扱いは必要である。既述したように、企画で作成する全てのドキュメントをソフトウェア資産と看做すことは拡大解釈に過ぎ、資産概念を曖昧にしかねない。ソフトウェア戦略や全体構想といった、具体的なソフトウェアとの繋がりが不明確なものは、ソフトウェア資産と看做すのは適切ではない。性格は異なるが、対象とするソフトウェアを廃棄することに関するドキュメントも、新たにソフトウェア資産とすることは適切ではない。こうした限定は欠かせないが、その範囲内にあるドキュメントはソフトウェア資産と看做し、資産計上とする会計処理が適合的である、と筆者は考えている。これまでの費用処理からすれば、容易には受入れられないかもしれないが、ソフトウェアをライフサイクル全域で捕捉する会計基準構制を採るならば、合理的であり、且つソフトウェアの実態に適合的な会計処理なのである。

なお、企画と連動する開発等との関連について補足すると、企画が独立的な大フェーズであることから、企画の完了時点で資産計上することが適宜的であると考える。その上で、開発等が着手予定凍結ないし途中で中止した場合には、資産の未償却残高があれば、その全額を除却処理することが後続の対応として妥当であると考える。企画完了時点の決定に反する新たな事象・事態の発生であり、遡及的な処置は不要だからである。

---

[154] 企画の一定の範囲を会計処理において資産計上する根拠である、ソフトウェアとしてのドキュメントに関しては、ソフトウェアの定義における重要な要素であるが、ソフトウェアの精細な事項に関しては、既にソフトウェア基礎論第１章「ソフトウェアの定義」で詳論している。

本論

第２章　ソフトウェアの研究開発と非研究開発

１．予備的考察

（1）アメリカ基準における不当な判断規準の策定

（2）日本の現行のソフトウェア会計基準の問題点

（3）研究開発類推適用への数少ない疑義の呈示

２．ソフトウェア・プロダクトの位相

（1）ソフトウェア・プロダクトのライフステージ座標

（2）ソフトウェア・プロダクトのライフステージ測位

３．ソフトウェア・プロセスの様相

（1）ソフトウェア開発のプロセス・イノベーション

（2）ソフトウェア開発と研究開発のプロセスの形態的相違

（3）ソフトウェア開発と研究開発のプロセスの定量的差異

第２章　ソフトウェアの研究開発と非研究開発

ソフトウェアの研究開発と非研究開発を主題的に考察する前に、研究開発に関わる比較的近年の重要な動向を一瞥しておきたい。国民経済計算において、2008SNA[155]では、「研究開発（R&D）を資本形成として扱う（現在は中間消費）」[156]ことになったのである。2008SNA への移行は、オーストラリアは 2009 年に移行済、カナダは 2012 年に一部移行済（研究開発の資本化は移行済）であり[157]、日本は「基本計画等を踏まえ、次回基準改定（平成 28（2016）年目途）までの移行に向け、その具体的な課題の整理、推計方法の検討を行っている」[158]、とのことである。

「2008SNA マニュアル」の勧告概要は、次の通りである。

「研究開発（R&D）は、知識ストックを増進させ、知識ストックを活用して新たな応用が生まれるようにするために、体系的に執り行われる創造的作業である。R&D は、付随的な活動ではなく、可能な場合には、それについて別個の事業所が区分されるべきである」、

「R&D の産出は、購入されたもの（アウトソースされたもの）であれば市場価格で評価し、自己勘定で行われたものであれば生産費用総額に、生産に使用された固定資本のコストを表す適当なマークアップを加えて[159]評価される。政府や大学、非営利の研究機関等で行われた R&D は非市場産出であり、使用された資本の収益を除いて、生産費用総額で評価される」、

「R&D への支出は、総固定資本形成として扱われる。ただし、当該活動がその所有者に何ら経済的利益をもたらさないことが明らかである場合は中間消費として扱う」、

「資産分類では「生産資産」の「固定資産」の「知的財産生産物」の内訳「研究開発」に計上される」、

「R&D への支出を総固定資本形成に含めることにより、特許実体は資産として表れなくなる。特許契約は、使用ライセンスの一形態で、サービスの支払または、資産の取得に対する支払として扱われる」[160]、

とのことである。少し補足的な説明を加えると、「R&D の成果は経済成長の重要な源泉であるにもかかわらず、SNA ではこれまで資本形成とされていなかった。資本形成に変更する場合、R&D 支出のすべてを資本形成とするべきか、あるいは R&D を実施した主体に便益をもたらすものなど一部に限るべきかが問題となる。これについては、将来に向かって便益をもたらす R&D の全額を資産に含めるべきだが、完了時点で全く経済的便益が認められないものは除くべきとされる。R&D を資本形成と

---

155 SNA(System of National Accounts：国民勘定体系)は、国民経済計算の国際基準である。
156 内閣府経済社会総合研究所国民経済計算部(2013)p. 1
157 同資料 pp. 2-3
158 同資料 p. 3
159 固定資本収益(純)を上乗せすることである(国民経済計算次回基準改定に関する研究会(2014)p. 1)。
160 同資料 p. 1(下線：原文)。

すれば、市場生産者のそれについては中間消費から振り替わることになるので、GDP が増加する。政府など非市場生産者の R&D については、最終消費支出から資本形成への振替なので、GDP の大きさには影響がない」[161]。「また、93SNA では特許を非生産無形資産とする」ことにしていたが、「R&D 資産を導入すれば特許を資産として別掲する必要がなくなり、R&D 資産の一部とすればよいことになる」[162]のである。

そして、「次回基準改定における対応の考え方（案）」としては、「2008SNA 勧告に沿って、R&D をより広範かつ明示的に捕捉する。R&D の産出額は、全て経済的利益をもたらすものと整理し、資産分類「知的財産生産物」の内訳「研究開発」として、資本化の対象とする」[163]、とのことである。「その際、JSNA[164]で対象とする R&D の範囲として、ＯＥＣＤ「知的財産生産物の資本計測に関するハンドブック（以下、IPP ハンドブック）」[165]を踏まえ、諸外国の事例と同様、フラカティ・マニュアル（以下、ＦＭ）[166]に準拠した科学技術調査（我が国の場合、SRD[167]）の対象範囲と同様とする。具体的には、産業の研究機関に加え、企業内研究開発や、政府サービス生産者及び対家計民間非営利サービス生産者に属する研究機関及び大学等の研究開発を対象とする」[168]、とのことである。

国民経済計算においては、国際的に研究開発が資産計上され、総固定資本形成として計上されることになり、日本の移行はまだではあるが（そして移行後、実際に推計され、年報の数値が発表されるのは更に数年後だが)、ほぼ既定路線のようである。ＩＦＲＳでは、研究と開発を切断し、開発は要件を充足すれば資産計上することになっているが、研究に関しては費用処理することになっており、会計基準の取り扱いとは懸隔が甚だしい取り扱いである。しかし、国際連合、欧州共同体委員会、ＩＭＦ（国際通貨基金)、ＯＥＣＤ（経済協力開発機構)、世界銀行が揃って採択した 2008SNA の規範力は強力である。国際会計基準等は、どう対応するのか。また、日本の会計分野ではどうコミットするのか。それにしても、気になるのは「基礎統計」として専ら SRD が俎上に乗っているが、財務報告における「研究開発費」は何故「基礎統計」の対象にならないのであろうか。

ソフトウェア開発は、研究開発ではない。正確を期した言い方をするならば、ソフトウェア分野でも当然に研究開発はあるが、あくまで一部であり、大半のソフトウェア開発は研究開発ではないのである。これが、本章の主題であり、本章を挙げて論証していくことである。

---

161 中村(2010)p.119(傍点：引用者)。
162 同書 p.120
163 国民経済計算次回基準改定に関する研究会(2014)p.3(下線：原文)。
164 日本版 SNA のこと。
165 ここに注番号 6 が付され、「Handbook on deriving capital measures of Intellectual property products, OECD 2010」と注記されている。
166 ここに注番号 7 が付され、「Frascati Manual, Proposed standard practice for surveys on research and experimental development, OECD 2002：研究開発に係る統計調査マニュアル」と注記されている。
167 Survey of Research and Development(科学技術研究調査)の略称。
168 同資料 p.3

ところが、不可解と言うしかないが、ソフトウェアに係る会計基準は真逆の認識[169]に立脚しており、ソフトウェア開発の大半を研究開発と看做しているのである[170]。日本基準は特に顕著で、そもそも研究開発に係る会計基準の中にソフトウェアに係る基準を含めていることに表れている(以下、「ソフトウェア会計基準」又は文脈で判然とする場合には単に「会計基準」「基準」とする)。しかし、それは日本基準に留まらず、アメリカ基準がそもそもの発端である[171]。

ソフトウェア並びにソフトウェア開発の実態を少しでも知っていれば判然としたことであるにも関わらず、会計関係者の認識は乖離が甚だしい。インタンジブルズ研究で著名なバルーク・レブ（Baruch Lev）さえ、ソフトウェア資産の認識において、開発原価の範囲問題ではアメリカ基準に異議を唱えているが[172]、ソフトウェア開発（の大半）が研究開発であるという誤認は共有し、疑義を呈していない[173]。我が国でも影響力を有していることからも残念である。また、日本におけるソフトウェア会計の代表的な研究者である櫻井通晴は、比較的最近の論稿でも「基準」３（２）に関して「逆に言えば、ソフトウェア開発にも研究開発に該当しない開発費があることを正式に認めた」[174]という見解を表明しているが、それ以上に本格的な議論を深めていない。なお、櫻井に関しては後ほど改めて取り上げる。

本章の目的は、ソフトウェア会計基準の全面刷新に向けて、そのセントラル・ドグマ[175]を解体し、新たな中核ないし基底的認識を構築することである。そのために、ソフトウェアにおける研究開発と、ソフトウェア開発という研究開発ではない一般的な開発（以下、非研究開発と概ね略記）を明確に区別可能とする判断規準を提案することである。

その意義は、これまでの混濁した財務情報によるミスリードを是正することである。現行のソフトウェア会計基準に基づいて処理された財務情報は、取り立てて新奇性のないソフトウェアを研究開発の成果と看做すことにより、ソフトウェアに関して「紛らわしい」財務情報となっている。他

169「認識」には会計特有の概念並びに用法があるが、本章ではより広く一般的な意味で「認識」という用語を概ね用いていると了解されたい。但し、会計的な脈絡では会計の概念として使用している。

170 ソフトウェア会計基準では、ソフトウェアの非研究開発にも研究開発に該当する部分がある ⇒ *ソフトウェア開発≒研究開発*、と看做している。それに対して筆者が論証することは、予め端的に言い表せば、ソフトウェアの非研究開発には研究開発に該当する部分などない ⇒ *ソフトウェアの研究開発≠ソフトウェアの非研究開発*、というごく当たり前のことである。なお、本章では、会計基準におけるソフトウェアに関する基本的な捉え方を問題にしているので、「市場販売目的」等の下位分類以前のソフトウェアを総体として取り上げる。

171 ソフトウェア開発の大半を研究開発と看做していることは、IAS38 無形資産も同類であり、大同小異である。

172 Lev(2001)邦訳 p. 106

173 Lev(2001)邦訳 pp. 106, 118

174 櫻井(2012)p. 112

175 セントラル・ドグマとは、体系的な教義において、中核であり且つ全体を貫く教義のことである(典型例はカトリックの三位一体説であり、周知の通り、プロテスタントにとって教義的には最大の争点であった)。ソフトウェア会計基準においては、ソフトウェア開発の大半を誤って研究開発と看做す規定が、それに相当する。

方で、新奇的な研究開発ではない活動を研究開発と看做すことにより、研究開発費が「実態以上に」計上された財務情報となっている。研究開発費は、ミクロ的には研究開発費売上高比率という経営指標により企業の成長性を判断する重要な情報となっており、マクロ的には研究開発費ＧＤＰ比率という経済指標により一国経済の成長性を判断する重要な情報となっている[176]。ところが、その研究開発費に研究開発ではない「模倣的」な活動の費用が混在しているとすれば、「嵩あげ」された数値により、判断はミスリードされ続けてきたことになる。これは速やかに是正されなければならない。本章は、そのための明解な判断規準を提示するものである。それにより、ソフトウェアにおける研究開発と非研究開発を適正に識別した情報が企業活動の信頼し得る情報となり、その開示がステイクホルダーへの適切な情報開示となることを目的としている。更に敷衍すれば、正確な情報こそが経済への貢献（良導）になる、と考える。

本章の構成概要は、次の通りである。まず第１に、予備的考察として、アメリカ基準における不当な判断規準の策定がなされた経緯とその内容をコンパクトに捉え直し、それを踏まえて現行のソフトウェア会計基準の問題点を挙げ、更に会計基準に対して呈示された数少ない疑義を吟味する。第２に、研究開発と非研究開発を区別するために、ソフトウェア・プロダクトの位相を捉える枠組みを提示する。まず、ソフトウェア・プロダクトのライフステージ座標を掲示する。次に、それを適用することで、ソフトウェア・プロダクトのライフステージの測位が可能になることを明らかにする。第３に、同様の目的で、ソフトウェア・プロセスの様相を考察する。まず開発プロセスそのものが研究開発である場合のプロセス・イノベーションを挙示する。次に、プロセスそのものは研究開発とは言えないが、プロダクトが新奇的か否かで、研究開発と非研究開発のプロセスが形態的に相違のあることを明らかにする。更に、それがプロセスの定量的差異となって顕現することをも明らかにする。こうして、プロダクトとプロセスの両面から総合的に、研究開発と非研究開発の区別を明らかにすることにより、筆者の主題的な主張の論拠を具体的に証示する。

１．予備的考察

（1）アメリカ基準における不当な判断規準の策定

アメリカの市場販売用ソフトウェアに係るソフトウェア会計基準等は、年代順に挙示すれば次の通りである[177]。

---

[176] 国民経済計算において、2008SNA では研究開発を全て資産計上し、総固定資本形成に計上することになっていることは、既述の通りである。

[177] 中村(2003)p.4。なお、各原名はカッコ内の通りである。内国歳入庁(Internal Revenue Service)、財務会計基準審議会(Financial Accounting Standards Board)、証券取引委員会(Securities Exchange Committee)、アメリカ公認会計士協会(American Institute of Certificated Public Accountants)、歳入手続(Revenue Procedure)、財務会計基準書(Statement of Financial Accounting Standards)、解釈書(Financial Interpretation)、適用指針(Technical Bulletin)、財務報告通牒(Financial Reporting Release)、問題提起書(Issue Paper)、公開草案(Exposure Draft)。

・1969 年　内国歳入庁（IRS）　内国歳入庁歳入手続 69-21（Rev. Proc. 69-21）

・1974 年　財務会計基準審議会（FASB）　財務会計基準書第 2 号（SFAS No.2）

・1975 年　財務会計基準審議会（FASB）　FASB 解釈書第 6 号（FIN No.6）

・1979 年　財務会計基準審議会（FASB）　FASB 適用指針第 79-2 号（FTB No.79-2）

・1983 年　証券取引委員会（SEC）　財務報告通牒第 12 号（FRR No.12）

・1984 年　アメリカ公認会計士協会（AICPA）　問題提起書

・1984 年　財務会計基準審議会（FASB）　公開草案

・1985 年　財務会計基準審議会（FASB）　財務会計基準書第 86 号（SFAS No.86）

最初は、「ソフトウェア開発費は、多くの面で 1954 年制定の内国歳入法第 174 条に該当する試験研究費と非常によく似ている」[178]という歳入手続 69-21 の規定から始まっている。しかし、どのように「多くの面で」「非常によく似ている」のか、説明はなされていない。その取り立てて根拠のない類推適用が、ソフトウェア会計基準の今日に到るも解消されていないセントラル・ドグマの発端である。続く SFAS No.2 では、「コンピューター・ソフトウェアは、多種多様な目的のために開発される。それゆえ、個々のケースについては、ソフトウェアが開発される活動の性質を第 8-10 節の指針に照らして判断し、当該ソフトウェア原価がそれに該当するか否かを決定すべきである。たとえば、販売のために新奇で高度なコンピューター・ソフトウェアの能力を開発する活動は、本基準書でいうところの研究開発に該当する」[179]（傍点：引用者）、と規定している。第 8-10 節の指針は新奇性に関する指針であり、この限りでは明解な規定のはずであった。新奇性を有するものは研究開発であり、新奇性を有しないものは研究開発ではない、というように。「ただし、SAFS No.2 は、ソフトウェア原価の中には研究開発に該当する活動と該当しない活動があるとも指摘」[180]することにより、再び錯綜・混迷の道に入り込んでしまったと言える。

それ以降は、周知の通り、新奇性の指針は後退し、「研究開発に該当する」か否かをソフトウェア開発の工程で区分することになり、その根拠として「技術的実現可能性（technological feasibility）[181]」が枢要の位置を占めるようになる[182]。しかしこれは、問題の誤った解法である。研究開発の判断規準は、あくまでも製品又は製法の新奇性である。このことは、ソフトウェアであろうと、それ

---

[178] 同上 p.5

[179] 同上 pp.5-6

[180] 同上 p.6

[181] SFAS No.86 Research and Development Cost of computer Software 4.における「技術的実現可能性」の確定の証拠は a.又は b.である。a.詳細プログラム設計を含む製作の場合、（1）製品制作のための技術的な利用可能性、（2）詳細プログラム設計の完了及び詳細プログラム設計と製品設計との首尾一貫性、（3）詳細プログラム設計の開発上の不確実性の解消。b. 詳細プログラム設計を含まない製作の場合、（1）製品設計及び作業モデルの完了、（2）作業モデルの完了及び作業モデルと製品設計との首尾一貫性、である。

[182] 問題提起書並びに公開草案では販売可能性と財務可能性も挙げられていたが、SFAS No.86 では削除された。なお、歴史的な考察としては丁寧なトレースが必要であるが、本章は問題の剔抉を主眼としているので、簡略に留める。同上 pp.10-45 以外にも、西澤(1991)、櫻井(1993)等参照。

以外のものであろうと、何ら変わらない。製品又は製法に新奇性のないソフトウェア開発は、研究開発ではない。ところが、ソフトウェアだけは何故か、新奇性のない、非研究開発にも「研究開発に該当する」部分（要素）があるという不合理な捉え方がなされた。会計基準設定の関係者にとって、それほどソフトウェアは不可解なものだったのであろうか。そのため、開発工程や「技術的実現可能性」といった「判断規準」が連鎖的に導出されたのである。しかしこれも、非適合的な論理展開である。言うまでもなく、研究開発では技術的実現可能性は重要な関心事であるが、それはあくまで研究開発の成否の判断規準である[183]。技術的実現可能性が確保されたからといって、研究開発がそれ以降研究開発でなくなるわけではない。研究開発は、当該研究開発が完了するまで、研究開発のままである。そうであるにも関わらず、ソフトウェア開発だけは何故、技術的実現可能性の確保というファクターによって、それまでの研究開発が研究開発でなくなるのか。成否によって基本的な属性が変わることはあり得ない。ソフトウェアに関しても、研究開発であれば、技術的実現可能性が確保される前も、確保された後も、何ら変わらず研究開発である。研究開発でなければ、始めから終わりまで研究開発ではない、というのが真っ当な論理的判断である。何れの場合も、技術的実現可能性は開発の成否の判断規準ではあるが、研究開発か否かという基本的な属性の判断規準ではない。開発工程による区分はその派生系であるが、研究開発であれば何れの工程で技術的実現可能性が確保されるかは一律ではないし（それだけ不確定的、不確実的であるのが研究開発というものであろう）、研究開発でなければ余程杜撰な開発計画でない限り、予め技術的実現可能性は確保されているのである[184]。そもそもプロダクト（製品）を不問に付して、工程の区分というプロセス（製法）だけを決定的な規準とすることが、片面的である[185]。こうした問題のコンタミネーション（contamination）と、資産あるいは費用の何れとすべきかという会計処理の議論が混然となって錯綜し、出来上がったのがSFAS No.86である[186]。

（2）日本の現行のソフトウェア会計基準の問題点

日本のソフトウェア会計基準は、市場販売目的のソフトウェアに関して、一言で言えば、アメリカ基準を模倣し、更に誤謬を増幅させたと言える[187]。模倣した点は、ソフトウェア開発を研究開発

---

[183] 成否の判断規準として、重要であるが、あくまで規準の１つである。技術的には実現可能でも、投資効果等で成功の見込みなしと見做し、中止あるいは製品化の断念等は当然にあるからである。

[184] 一般的なソフトウェア開発でもプロジェクトの失敗は少なくないが、それは別の様々な要因による（本論第６章参照）。なお、研究開発でない一般的なソフトウェア開発においても、技術的実現可能性に疑念がある場合は、開発の早期にフィージビリティ・スタディ（必要ならば実装を含めて）を行なうものである。なお、ソフトウェアに半可通なバルーク・レブが見当違いにも、パッケージ・ソフトウェアに関するプロジェクト内のテスト完了後に行なうαテストやβテストに「技術的実現可能性」を結び付けていることは誤りである。

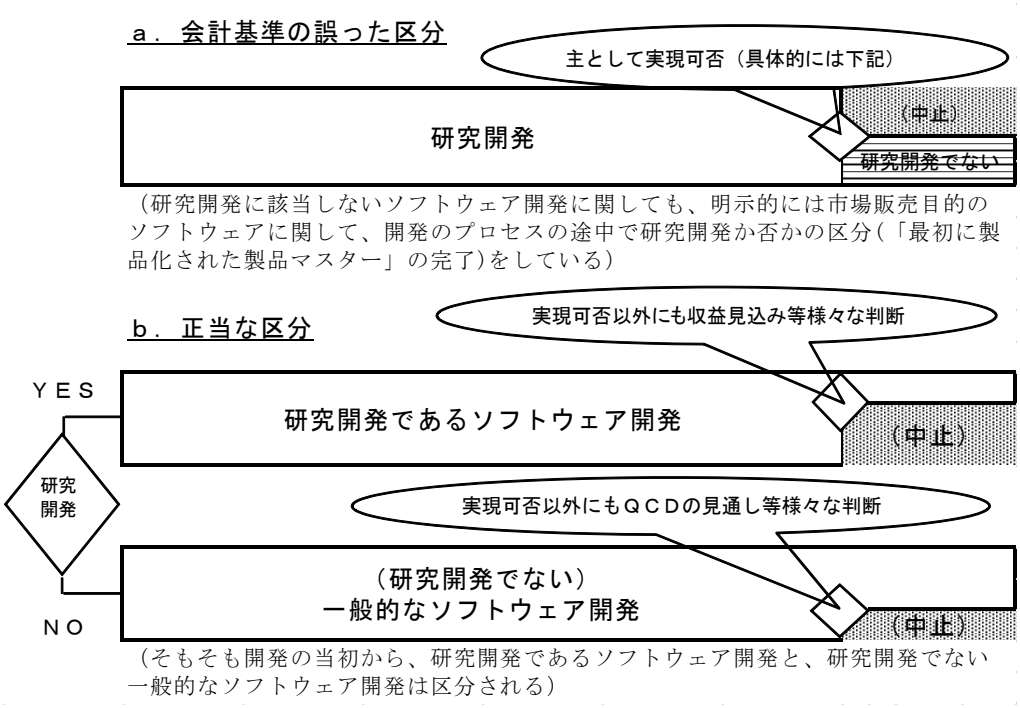
[185] SFAS No.86の「技術的実現可能性」の要件には、一部プロダクトに関する規定も含まれてはいるが、形式的な要件に過ぎないので、取り扱っていないに等しい。

[186] ＩＢＭ社の影響力によるバイアスといった刻印が顕著だが、政治力学の問題なので割愛する。

[187] 櫻井(1999)は、「移植」(p.4)という言い方をしているが、カスタマイズもしており、且つそれが

と不当に拡大解釈し、研究開発であるものはもとより、非研究開発にも研究開発的要素があるということ、その有無を判断するのに「技術的実現可能性」が確保されたか否かを規準とし、開発工程で区分するということである。誤謬を増幅した点は、開発工程における「技術的実現可能性」の判断時点をアメリカ基準より下流に引き下げ、研究開発を「最初に製品化された製品マスターの完成」までとしたこと、研究開発の判断規準に「販売の意思」をも含めたことである。これによって、研究開発ではない範囲はごく僅かに狭められた。

図表１−２−１　ソフトウェア開発の区分



（出典：筆者作成）

西澤脩は、会計基準をほぼ忠実に祖述している。「同じソフトウェアでも、研究開発目的のソフトウェア（専ら当該研究開発プロジェクトに使用され、他の研究開発プロジェクトに転用できないもの）は、ライフサイクルの全過程が研究開発とされる。問題は、非研究開発目的のソフトウェアである。この種のソフトウェアについては、開発過程までが研究開発とされ、生産以後は研究開発の対象外となる」[188]。市場販売目的のソフトウェアに関しては、「最初に製品化された製品マスター」の完成「時点までの制作活動は研究開発と考えられるため、ここまでに発生した費用は研究開発費

改悪の方向になっている。

[188] 西澤(2003)p. 31(傍点：引用者)。なお、同ページの図表 2-1「ソフトウェアのライフサイクルと研究開発の範囲」によると、ソフトウェアのライフサイクルを大きく「制作過程」と「付随過程」に区分し、次に「制作過程」を「計画」、「設計」、「開発」、「生産」に区分しているので、「生産以後」とは、「生産」(マスター制作とマスター複製)と「付随過程」(「アフターサービス」(顧客支援、保全、機能強化)と「廃棄」(除去・評価)を意味する。

として処理する」[189]、としている。市場販売目的のソフトウェアに限らず、自社利用のソフトウェアに関しても誤って同様の記述をしている[190]。総じて西澤は、非研究開発目的のソフトウェアであるにも関わらず、「開発過程までが研究開発とされ」る、あるいは製品マスター完成時点までの「制作活動は研究開発と考えられる」などと、自明の如く断定するだけで、理由等の説明を何ら行っていない。何故、研究開発目的ではないにも関わらず、ソフトウェア開発のある範囲が「研究開発」と看做されるのか。一般的なソフトウェア開発は、開発工程によって「研究開発活動」と「商業生産活動」の区分などがあるわけもなく、全てが「商業生産活動」である。受託開発はそれに基づいて成り立っている。市場販売目的の開発でも同様である。研究開発を目的としていないのに、「実態」としてそうだと看做すのであれば、それ相当の根拠ないし理由付けがあって然るべきではないのか。しかし、何も説明せず、会計基準をただ鵜呑みにしているだけである。それは西澤に限ったことではなく、管見の限り、ほとんど自明の理であるかの如くに、不問に付せられたまま、それを当然の前提として資産認識される原価の範囲問題等が論じられている。このように、アメリカ基準の誤謬を模倣したことが、第１の問題である。

第２に、「著しい改良」に関する会計処理の非合理性は、上記の研究開発と看做す誤りの派生系である。「著しくない改良」が資産の増加となるのに、それよりも更に資産性の増す「著しい改良」が大半は資産計上されず、費用処理されるという、著しく合理性を欠く規定とならざるを得なかったのは、「著しい改良」を研究開発と誤って看做した結果である。「著しい改良」に関する実務指針の例示に関しては、何れも研究開発でないことが明らかである。精細にケース分けすれば、採用する機能や技術のライフステージ如何によっては、研究開発であり得るものもあるが、その立ち入った考察は後ほど行なう。

第３に、市場販売目的と自社利用の会計処理に大きな相違があることである。市場販売目的のソフトウェア開発の大半をリスクの高い研究開発と看做し、他方では自社利用のソフトウェア開発にはリスクを想定しておらず、それほど研究開発的要素が多くあると看做していないため、資産認識の範囲が著しく異なっている。しかも資産認識の判断規準として、市場販売目的では「技術的実現可能性」としているのに対して、自社利用では「将来の収益獲得又は費用削減の確実性」とし、「開発」と「利用」という全く異なる側面に照準を当てた規準となっている。そもそもの予断として、市場販売目的のソフトウェアは「新奇性」があり且つ技術的に高度な開発であるという「思い込み」が伏在しているが、実態は必ずしもそうとは言えない。利用以前に、開発においても自社利用ソフトウェアの方が、新奇性があり技術的にも高度なものは少なくない[191]。そしてまた、開発と利用と

---

[189] 同書 p. 41（傍点：引用者）。

[190] 同書 p. 38。そして、市場販売目的と自社利用の会計処理について費用と資産の範囲を同じように図示しているが、それは実務指針を誤って解釈したものである。正しくは自社利用の場合には資産要件を充足していれば、開発の全範囲を資産計上とするのである。

[191] 別の例証を挙げる。Jones（1995）において、ジョーンズ自らが所属する SPR（Software Productivity Research）社の調査結果として、６つのソフトウェア分野に関するソフトウェアの成

は全く別次元であるが、自社利用のソフトウェアの利用による効果の「確実性」が利用におけるリスクの想定だとすれば、事前に将来のリスクを「確実に」回避できることを要件とするということは、リスクがないと看做しているに等しい。しかし、利用におけるリスクもまた存在するのである。

誤謬を増幅した点の１つとして、「技術的実現可能性」の判断時点を後方化させたのは、製造業一般における研究開発並びに研究開発後の製造と無理に対応付けようと意図したからである。ソフトウェア開発には言うまでもなく研究開発と、研究開発過程のない非研究開発がある。研究開発である場合には製造業一般と対応付けても一応間違いではないが、研究開発でない場合にはそもそも対応付けは成り立たない。製造業においても研究開発がなく、全てが製造である業態はあり、強いて言えば、その製造に相当する。但し、定型性の該当範囲が異なり、そうであるが故にソフトウェアでは「製造」と言わず、開発と言うのである[192]。しかも研究開発である場合、研究開発後の製造は製造業一般の製造と比べれば、複製の容易性等からごく軽微な工程でしかないことは無形のソフトウェアの特性によるもので、無理に同等ないし近似的な対応付けをする必要などないのである。

２つに、研究開発の判断規準として、技術的実現可能性に留まらず、市場におけるソフトウェア製品の「販売の意思が明らかにされること」までをも含めている。この点がアメリカ基準との大きな違いであるが[193]、開発におけるリスクと販売におけるリスクとを混同している。それ以外にも問題は少なからずあるが、本章の主題との兼ね合いで、以上に留める。

（3）　研究開発類推適用への数少ない疑義の呈示

ソフトウェア会計基準設定前後に相当数の関連論文等が発表されているが、ソフトウェア開発を研究開発と看做すことに疑義を呈している先行研究は思いのほか少ない。

その数少ない研究の１つが、会計基準設定前に発表された、天明茂（1996）である。「汎用ソフトウェア」（会計基準の分類では「市場販売目的」のソフトウェアに相当する）に関して、次のように

功と失敗を、６つの評価基準によりランク付けしている。ソフトウェア分野は、①システムソフトウェア(OS 等)、②軍需ソフトウェア、③情報システムソフトウェア(企業の業務システム)、④外部委託ソフトウェア(③と同様のソフトウェアをソフトウェア企業が受託開発したもの)、⑤市販ソフトウェア、⑥ユーザ開発ソフトウェア(例えば表計算ソフトにマクロを組み込んだシステムのようなもの)、である。評価基準は、①スケジュール、②コスト、③品質、④利用性、⑤支援、⑥保守、である。この調査結果に拠ると、日本及びアメリカの会計基準における市場販売目的のソフトウェアに相当する①システムソフトウェアと、⑤市販ソフトウェアの成功が、総合的に他の分野に比べて遥かに優れた結果となっている。これは、本章の論旨を支持する内容であり、会計基準において自社利用に比して、市場販売目的に関する技術的実現可能性が困難で、リスクが多く、研究開発的性格が濃厚と看做していることが実態にそぐわないことの一つの例証と言える。

[192] ソフトウェア開発の中の１つの工程として、プログラム製造があるが、製造業一般における製造(全般)とは意味並びに対象範囲が異なる。

[193] SFAS No.86 では資産要件から除外されたが、公開草案では「市場可能性」と「経営者の支持」も要件とされていた。このうち「経営者の支持」が、日本基準の要件の「販売の意思」に相当するが、日本基準はごく限定して不完全な要件のみを継承した、という意味において誤謬を増幅させている。

述べている。「よほど新規技術を伴う研究開発要素の高いものは別として、通常のソフトウェア開発においては基本設計以降に残された不確実性は、開発コストの増加、開発期間の延長、効率低下などソフトウェアの経済性に影響しても、ソフトウェアの技術的な成否に直接結びつかないのがほとんど(15)と言われる。こうした状況に鑑み、自社のソフトウェアの特性から、技術的実施可能性が確立される工程がほぼ決まっているような企業にあっては、当該特定工程以降の原価を資産計上の対象とすることも実務上の簡便法として認められよう」[194]。更に続いて、次のように述べている。「なお、資産計上の対象とされない費用についてFASが期間費用としていることには異論はないが、そのすべてを研究開発費とすることには疑問である。私見では研究開発要素の強いものは除いて、通常のソフトウェア開発に関わるものは売上原価として当該期間の売上高に対応させるべきものと考える。したがって製品マスターの開発費は研究要素の強い商品化決定前の研究開発費、通常のソフトウェア開発費のうち商品化決定前の期間原価、そして同じく商品化決定後の資産計上すべき原価に３分類されることになる」[195]。これらにおいて、「よほど新規技術を伴う研究開発要素の高いものは別として」、あるいは「研究開発要素の強いものは除いて」、「通常のソフトウェア開発」は研究開発と看做すべきではない、という見解が述べられている。更に精緻化されれば有力な見解たり得たであろうが、「研究開発要素」の詳細な内容に踏み込んだ進展はそれ以降にもみられなかった。なお、工程での区分という点ではアメリカ基準に囚われている、と言える。

もう１つは、会計基準設定後に発表された、櫻井通晴（1999）である[196]。「グローバル・スタンダードに一歩近づけた」[197]こと、また「統一的な判断基準を与えたものとして」「賛意を表する」[198]としながらも、基本的な構成等に関する疑問を呈している。まず、そもそも「ソフトウェアの会計基準を“研究開発費等”とする名称のなかで取り扱ったことが妥当であったかについては疑問が残る」[199]とし、続く「私見」でも次のように述べている。「第一に、「基準」では、研究開発費等に係る基準とされている。しかし、その実質的な内容がソフトウェアの会計基準であることは誰の目にも明らかである。つまり、本文から明らかなように、研究開発費に関する基準としては、“研究開発費は費用処理”と規定したにすぎなく、実質的にはソフトウェア会計基準である。であるとすれば、な

---

194 天明(1996)p. 92。なお、同稿は汎用ソフトウェアの資産性と共に、(長期)棚卸資産としての計上を妥当とする主張を行っているが、今日的には科目としては無形資産計上が通説となっており、筆者も同意見であり、棚卸資産論議には最早歴史的「意義」しかなく、棚卸資産であるコピー製造した製品と、マスターは別個のものであるので、立ち入らない。また、文中の注(15)は田宮治雄(1989)を参照している注記への指示である。

195 同上 p. 92

196 櫻井は、同時期に同主題の論文を他にも発表しているが、基準を最も明確に批判しているのは、当論文である。

197 櫻井(1999)p. 4。なお、「グローバル・スタンダードはアメリカのそれであり」と断り書きをし、「日本企業とは企業環境が異なるところへの会計基準の移植は、ソフトウェア・ディベロッパーやベンチャー・ビジネスに対して若干の問題を提起した」(p. 4)と続けている。

198 同上 p. 6

199 同上 p. 6

ぜ正直にソフトウェア会計基準としなかったか。明確な弁明が必要であったように思われる」[200]。しかし、設定関係者からの「弁明」は、管見の限り全くなされていない。

「第二に」挙げている理由は、「ソフトウェアを研究開発費の枠組みのなかでとりあげるのはあまりに時代に遅れているように思われる」とし、それはアメリカで「一九八五年の会計基準（ＦＡＳＢ第八六号）が研究開発費とソフトウェア開発費とを切り離して論じていることからもあきらかである」[201]としている。

「第三に」挙げている理由は、次の通りである。「日本企業でソフトウェアを研究開発費のなかで取り上げることは、日本企業のソフトウェア開発の実態からすると、OS を開発している特定のコンピュータ・メーカーを除けば、現実の感覚と遊離しているといわざるをえない。現在のソフトウェア会計基準の制定に当たっては、少なくとも三種類のソフトウェア・ディベロッパーのソフトウェア開発環境を考慮すべきであった」[202]。「コンピュータ・メーカーが開発する OS はたしかに研究開発的要素が大である。しかし、ユーザー企業がソフトハウスに開発を依頼するのは、大方が研究開発的要素の少ない委託開発である。ソフトハウスの多くは研究開発的要素が少ないアプリケーション・ソフトウェアを受託している（後略）」[203]。「このように見ると、「意見書」はコンピュータ・メーカーの開発する OS を想定して制定されているようにすら思われる。その反面、日本における大多数のソフトウェアを開発しているソフトハウスの実態や意見は「意見書」にほとんど反映されていない」[204]、と断じている。業界事情に踏み込んで、傾聴に値する見解を述べているが、肝心なところでは支持できない。１つは、企業（ディベロッパー）単位の捉え方の粒度が粗いことである。コンピュータ・メーカーも、研究開発だけではなく、受託開発等もしている。また、「研究開発的要素

---

[200] 同上 p. 6。なお、後続の箇所でも、「研究開発費の基準ではなく、「ソフトウェア会計基準」と題した基準を制定してほしかった」(p. 6)、と繰り返している。

[201] 同上 p. 6。但し、櫻井が根拠とする歴史認識は支持できない。「ソフトウェアを研究開発費のなかで公式にとりあげたのは、アメリカでは IRS(内国歳入庁)が一九六九年、FASB(財務会計基準審議会)は一九七四年のことである。当時はまだコンピュータ・ソフトウェアに関する知識も浅くソフトウェアの開発が困難を極めていたから、研究開発費会計の枠組みのなかでとりあげられたのは当然であった」とし(傍点：引用者)、「しかし、一九八〇年代になると、ソフトウェアの開発は研究開発というよりは一種の生産活動であると解されるようになってきた」(p. 6)という極端な年代的対比をしているが、その間に劇的な変化があったわけではない。従って、ソフトウェア開発という事象自体は、一般的な開発であれば、1969 年ないし 1974 年においても研究開発ではなかった。また、「ソフトウェア資産性をめぐる問題は以前ほどの重要性を持たなくなっている」と述べ、その理由として「情報技術の環境は、旧来のレガシー・システムから現在ではクライアント／サーバ・システム(Client/Server System;C/S システム)に移行し」、それは「従来の汎用コンピュータによるレガシー・システムのように大幅なソフトウェア開発費は必要でなくなった」(p. 4)としているが、明白に誤りである。Ｃ／Ｓ系は当初ローカルな比較的小規模なシステムに採用されたから、開発費が少なかっただけで、次第に基幹系システムにも普及拡大し、開発が大規模化するにつれて、開発費も増大したのである。些かも資産性が「重要性を持たなくなっ」たわけではない。

[202] 同上 p. 6

[203] 同上 p. 6

[204] 同上 p. 6

が少ないアプリケーション・ソフトウェア」というが、アプリケーション・ソフトウェアでも研究開発はある。従って、あくまで個々のソフトウェア単位に、研究開発か非研究開発かを捉えなければならない。もう１つは、「研究開発的要素が少ないアプリケーション・ソフトウェア」というように、「少ない」ながらも「研究開発的要素」が含まれていると述べていることである。その「研究開発的要素」の多少とは具体的にどのような事態であり、どのように分析あるいは測定し得るのかを明確にし、「研究開発的要素が少ない」のではなく、非研究開発には研究開発的要素が含まれていないことを突き詰めなければ、会計基準の中核的な誤謬を打破し得ないのである。それ以降の研究を含めて、櫻井はそこまで進展させていない。

２．ソフトウェア・プロダクトの位相

（1）ソフトウェア・プロダクトのライフステージ座標

研究開発は、新奇的な製品又は製法を追求する活動である。ソフトウェアに関しても、何ら変わらない。そこでまず、ソフトウェア・プロダクト（製品）の新奇性をどのように捉えるか、を明らかにする。それによって、個々のソフトウェア開発が研究開発であるか、それ以外の一般的なソフトウェア開発であるかを明確に判断することができるからである。天明茂並びに櫻井通晴が、疑義を呈しながらも、具体化に到っていないことへのアプローチを試みるのである。

筆者は、ソフトウェア・プロダクトの新奇性を捉えるために、ジェフリー・ムーア（Geoffrey Moore）が提唱するイノベーションの「ライフサイクル」論を援用する。それは、「新たなテクノロジーに基づく製品が市場に受け入れられていくプロセスを、製品ライフサイクルの進行にともなって顧客層がどのように変遷するかという観点からとらえたもの」[205]である。「製品ライフサイクルの進行」に重点をシフトすれば、ソフトウェアが新奇的であるかどうかを捉える座標とし得る。その際、ソフトウェアのライフサイクル（開発～保守・運用～廃棄）と紛らわしいので、「ライフステージ」と呼ぶことにする。また、「顧客層」の区分・画期は、①革新派（Innovators）、②早期適応派（Early Adopters）、③前期多数派（Early Majority）、④後期多数派（Late Majority）、⑤無関心派（Laggards））であるが、ソフトウェアのライフステージの画期に相応しく改変する。①創生期、②普及初期、③普及拡大期、④コモディティ期、⑤レガシー期、とする。①創生期とは新奇的なプロダクトが誕生するステージであり、②普及初期とは先進的な利用が少数ながら開始されるステージであり、③普及拡大期とは利用が広範囲に広がっていくステージであり、④コモディティ期とは利用が飽和状態となり当たり前に使われるが差別化の余地は少なくなるステージであり、⑤レガシー期とは旧式化・陳腐化し衰退傾向にあり機会があれば別のプロダクトにリプレースされるステージである。

なお、ムーアを援用するのは、イノベーション研究において３ないし４段階に区分する一般的な

[205] Moore（1991）邦訳 p. 14

「ライフサイクル」論に比べ、ソフトウェアの変遷・推移を捉えるのに最も適合的なものだからである。１つは、創生期と普及初期を区分することは、研究開発か否かを識別する上で決定的に重要である。２つには、普及初期と普及拡大期を区分することは、キャッチアップが先進的か否か、というソフトウェア業界の主要ターゲットを識別する上で有意義である。３つには、コモディティ期とレガシー期を区分することは、技術革新の激しいソフトウェアに関して、意外に思われるかもしれないが、単純に新旧交代がなされるのではなく、コモディティ期は比較的長く続き、またレガシーであっても長く延命する技術等もあり、それらを適切に識別するのに相応しいのである。

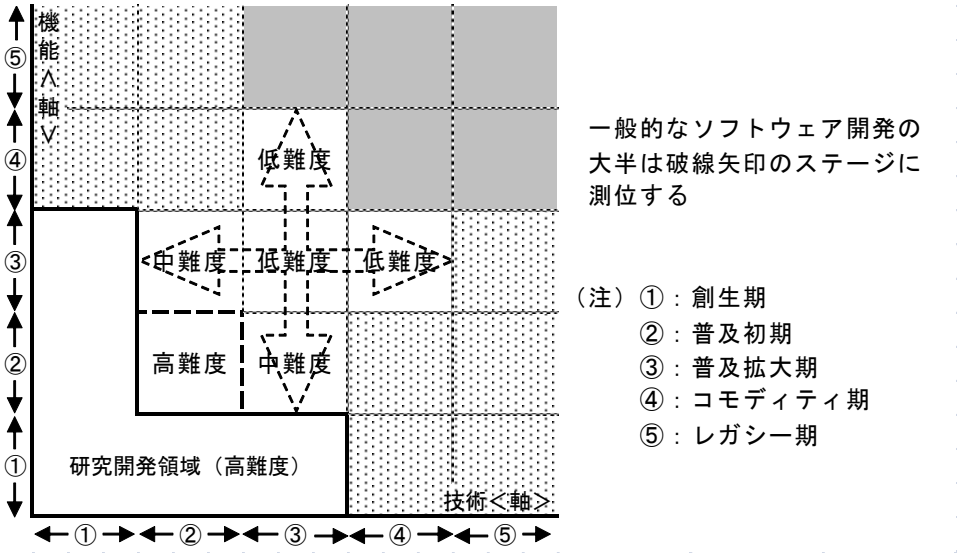
ソフトウェア・プロダクトの新奇性を捉えるためには、もう１つの分析的な枠組みが必要である、と考える。それは、ソフトウェアの機能と技術（内容と実現手段）という区分である。プロダクトとして出来上がったものは一体的であるが、分析的には区分可能であり、且つ区分しなければ新奇性が不分明になるおそれがあるからである。何故ならば、機能的には新奇的だが、技術的には広範に普及したもので些かも新奇的でないというプロダクトがあり、またその逆もあるからである。また、基本設計等において、ある規模以上の開発の場合、機能設計と方式設計はアクティビティとして分けられることも多く、実務的にも容易に区別することができるのである。会計基準における「技術的実現可能性」は、両者を識別せず、一体として取り扱っている。ソフトウェアを分析的に適切に捉えるには、両者を識別すべきことを強調しておきたい。なお、機能と技術が分解可能なことは、よく行われる異なったプラットフォーム（ハードウェア又はOS等）への移植（migration）を考えれば、理解しやすいであろう。同一の機能を異なった技術で実現することだからである。

これらにより、機能に関してライフステージを設定し、技術に関しても同様にライフステージを設定して、２次元の座標とすれば、開発対象であるソフトウェアの位相を特定することができる。それを図表１-２-２ に図示した。そして、機能並びに技術が総合的にみて新奇的か否か、従って当該ソフトウェア開発が研究開発であるか、それとも非研究開発であるかを、プロダクトの観点から明確に判断することができるのである[206]。

---

[206] これに関してもう１つ参考としたアイディア源を紹介する。反復型開発方式の代表的な提唱者の一人であるウォーカー・ロイス(Royce,Walker)著『ソフトウェアプロジェクト管理―21世紀に向けた統一アプローチ(Object Technology Series)』（日本ラショナルソフトウェア監訳、ピアソン・エデュケーション刊行、1998年原著初版、1999年邦訳初版、2001年邦訳新装版）における「プロセスフレームワーク」である。ロイスは、ソフトウェア開発に如何にプロセスフレームワークを適合化させるかという観点から、種々の考察を行っている。その中で、「プロセスの多様性を表わす２つの主要な次元」として、「技術的複雑さ」と「管理的複雑さ」という２つの次元を設定し、それぞれの複雑さを高低で表わしている。ソフトウェア開発のプロセス間の違いを、プロジェクトの規模等の６つのパラメータで捉えている。ロイスの言う「技術」とは、機能と技術を包含した、広義の技術を意味する。ロイスの観点はプロセスであるが、プロダクトに重点を置き、分析的に捉える場合には、機能と狭義の技術に分解することが適切である。主要なアイディア源ではあるが、そのまま採用せず、独自の座標を設定することとした。

図表１−２−２　ソフトウェアのライフステージ座標



（出典：筆者作成）

（２）ソフトウェア・プロダクトのライフステージ測位

ソフトウェア・プロダクトのライフステージに関して、注意を要することがある。１つは、全てのソフトウェア・プロダクトが、全てのライフステージを経過するわけではない、ということである。ある機能又は技術は、出現しても、余り普及もせずに消えていく（相当数はそうであろう）、あるいは大した期間を経ずに急速に普及拡大し更にコモディティ化してしまうものもある（例えばクライアント／サーバ型システム等）、あるいはレガシーと言われながらも、長らく利用され続けているものもある（メインフレーム系等）。もう１つは、ソフトウェア・ライフサイクル（開発～保守・運用～廃棄）とは異なり、ライフステージはプロダクト自らが変化して変わるわけではなく、主としてソフトウェア（業界）の動向によって受動的にステージが変化することである。

さて、ライフステージの測位は、果たしてどの程度の客観性を有するか。１つに、『日経コンピュータ』等々の業界誌やインターネットの市場・技術動向情報等により確認できる。２つに、各業界における他社動向等は競争上相当に意識されキャッチアップがなされる傾向が強いので、各企業の採用状況から確認できる。３つに、開発実務に即した最も確かな裏付けとして、開発プロジェクトを編成する場合、採用する機能並びに技術の普及度によって、要員調達の難易度が大きく左右されるのである。従って、開発の難易度も左右される。このように、業界ないし市場、各企業、当事者（プロジェクト）の３つの視点から動向を捉えることによって、ライフステージの測位は十分に客観性を有すると言える。

図表１−２−3　機能と技術のライフステージの組み合わせ

		技術				
		①創生期	②普及初期	③普及拡大期	④コモディティ期	⑤レガシー期
機能	①創生期	稀なケース	チャレンジ（リスキー）	チャレンジ（リスキー）	ビジネス的に考えにくい	ビジネス的に考えにくい
	②普及初期	チャレンジ（リスキー）	チャレンジ	○	ビジネス的に考えにくい	ビジネス的に考えにくい
	③普及拡大期	チャレンジ（リスキー）	○	○	安全第一で多少ありうる	ビジネス的に考えにくい
	④コモディティ期	ビジネス的に考えにくい	ビジネス的に考えにくい	安全第一で多少ありうる	開発はなく、保守の領域	開発はなく、保守の領域
	⑤レガシー期	ビジネス的に考えにくい	ビジネス的に考えにくい	開発はなく、保守の領域	開発はなく、保守の領域	開発はなく、保守の領域

（出典：筆者作成）

測位に関して、図表１−２−3 に沿って説明を行なう。第１に、機能又は技術が①創生期の場合が新奇性のあるプロダクトであり、その開発が研究開発である。片方が①創生期の場合、他方が④コモディティ期ないし⑤レガシー期にある場合を除外しているのは、販売可能性の観点からビジネスとしては考えにくいからである。機能又は技術が②普及初期の場合も同様である。機能が③普及拡大期にある場合に採用技術を⑤レガシー期とすることも同様である。第２に、機能が④コモディティ期ないし⑤レガシー期にある場合、（新規）開発は考えにくいが、利用されている限り、保守していくことはある。但し、④コモディティの機能を③普及拡大期の技術で開発すること並びにその逆は、低リスクなので多少はあり得るかもしれない。第３に、機能又は技術が②普及初期ないし③普及拡大期の場合が（新規）開発の圧倒的多数と考えてよいであろう。開発の難易度は前掲の図表１−２−1 に示した通りである。第１の場合以外は、何れも研究開発でないことは確実である。

以上によって、ステージの境界上に位置する等で判断が微妙なケースが若干あり得るとしても、個々のプロダクトを具体的に測位することは容易且つ高い蓋然性で可能である。次に取り上げるプロセスと総合的に捉えるならば、完全に判然とする。

３．ソフトウェア・プロセスの様相

（1）ソフトウェア開発のプロセス・イノベーション

研究開発は、新奇的な製品又は製法を追求する活動である。ソフトウェアに関しても、何ら変わらない。そこで、ソフトウェア・プロダクト（製品）に続いて、次にソフトウェア・プロセスの新奇性をどのように捉えるか、を明らかにする。それにより、個々のソフトウェア開発が研究開発であるか、非研究開発であるかを十全に明確に判断することができるからである。その際、２通りのアプローチが必要である。１つは、プロセス自体が新奇性を有しているか否か、ということである。もう１つは、プロセス自体に新奇性はないが、プロダクトが新奇的であるか否かで、それに適応的なプロセスが異なり、それによりプロセスとして研究開発か否かを区別する、ということである。

ここではまず、プロセス自体が新奇性を有する場合を概観する。ソフトウェア開発のプロセス・イノベーションを包括的に捉えるには、大別すると、①開発方式、②開発技法、③開発ツール、④プロジェクト管理という事項に分けて捉えることが適切である[207]。

①開発方式は、ウォーターフォール型、反復型（総称）としてインクリメンタル、プロトタイピング、スパイラル、アジャイル型等がある[208]。

②開発技法は、モジュール化、共通化、標準化、構造化、データ駆動開発、オブジェクト指向、モデル駆動開発、テスト駆動開発、リファクタリング、アスペクト指向、コンポーネント開発、アーキテクチャ駆動開発、ドメイン駆動開発等がある[209]。

③開発ツールは、他の①②④と異なり、膨大な数がある。古くはコンパイラ（アセンブラを含む）、テキスト・エディタ、デバッガに始まり、1970 年代の CASE ツールや今日の Visual Studio のような総合的なもの、更にはフレームワークと呼ばれる開発ツールでありつつ、開発するソフトウェアにコンポーネントとして組み込まれるものまで多数ある。それらを利用することがプロセスの範疇に属することであり、ツール自体はプロダクトの範疇のものである。そして、開発ツールは当該プロジェクトで自ら新奇性のあるツールを考案・作成し、且つ利用する場合を除き、既存のものを利用する限り、プロセスとしての新奇性はなく、研究開発には該当しない。

④プロジェクト管理は、ＰＭＢＯＫ（Project Management Body of Knowledge：プロジェクトマネジメント知識体系）を始めとして様々あるが、固有名が付いたものは余りない。

筆者の知る範囲で挙示したので、漏れはあるかもしれないが、特徴的なことは③開発ツールを除き、それ以外はソフトウェアの歴史 60 年ほど[210]を通しても、ごく限られた数のものが考案され、実用化されただけだということである。これらを考案し、実用化する取り組みは研究開発と言ってよいであろう。従って、プロセス（製法）自体が研究開発と言えるのはほんの数える程のものである。それ以外は全て、プロセスに新奇性はなく、プロセスとしては研究開発の性格を有していないのである。

（2）ソフトウェア開発と研究開発のプロセスの形態的相違

次に、もう１つのアプローチを行なう。プロセス自体に新奇性はないが、プロダクトが新奇的で

---

[207] この分類は、特定の文献等に依拠したものではなく、ソフトウェアに関する多くの文献並びに業界関係者から得た情報を元に、筆者が独自に抽出・整理したものである。

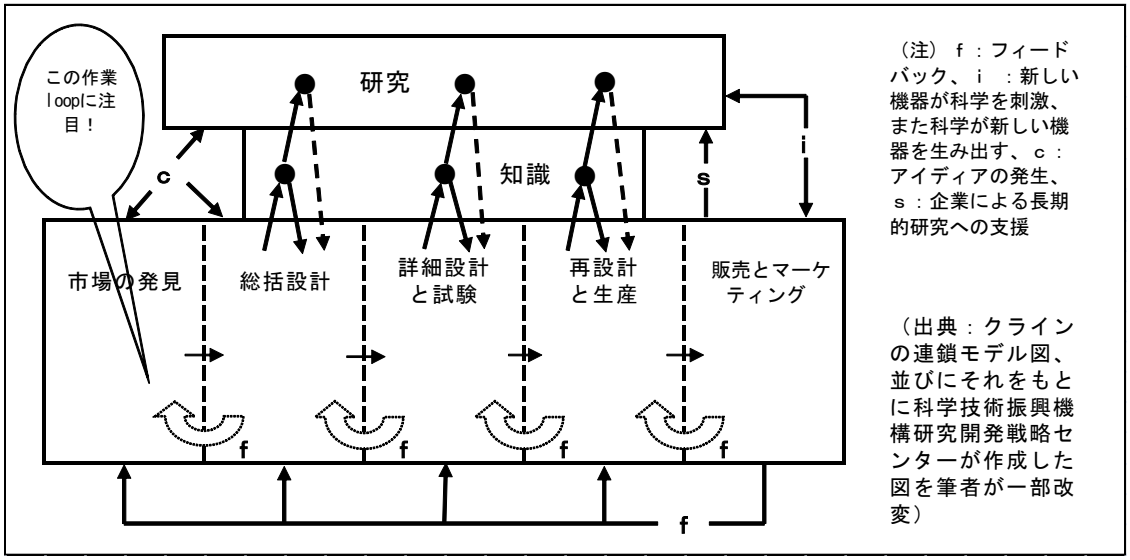
[208] Pressman(2005)邦訳 pp. 35-55。但し、分類並びに呼称を簡略化のために、また筆者の分類である②開発技法を混在させているので、一部改変した。

[209] 開発技法の概説書は、筆者の探索の範囲では適当なものは見当たらなかった。個々の技法に関する解説書は夥しくあり、そのほんの一部を筆者は閲読したに過ぎないが、それらから抽出した。

[210] コンピュータの創生は早くて 1939 年、ないし 1940 年代だが(Lattanze(2009)邦訳 p. 27)、ソフトウェアが独立的な意義を有するようになったのは 1950 年代に入ってからで、１つのメルクマールはプログラミング言語として、機械語より自然言語に多少近いアセンブリ言語が考案されたことであり、それから凡そ 60 年が経過している。

あるか否かで、それに適応的なプロセスが異なり、それによってプロセスとして研究開発か否かを区別する、ということである。プロダクトのライフステージ測位と併せて、ソフトウェア開発が研究開発であるか否かを確実に捉えることができることになる。

図表１－２-4　研究開発プロセスの模式図

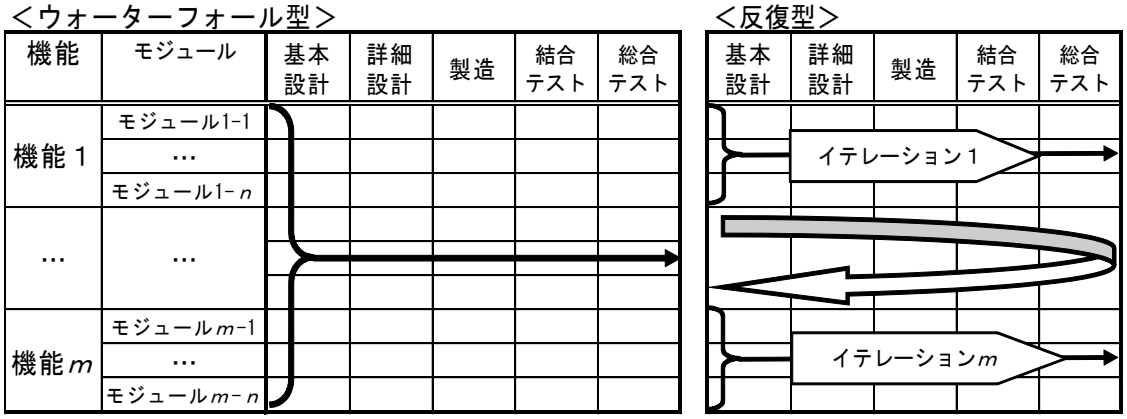


端的に標語的に対比すれば､研究開発のプロセスは反復的(iterative)ないしループ的であるが、非研究開発は直線的（linear）である、ということである。その意味するところは、以下の通りであり、各々を図表１－２-4、１－２-5 に図示した。研究開発は、実績のない、成果が不確実なものを探求するのであるから、プロセスにおいて不可避的に試行錯誤を繰り返さざるを得ない。どの程度の試行錯誤の繰り返しを許容するかは、計画やマネジメントによるとしても、それを許容しない研究開発はあり得ない。研究開発マネジメントにより管理は強化される傾向にあり、短期開発並びにコスト削減のミッションも強まっているとは言え、試行錯誤の繰り返しは研究開発プロセスに構造的に組み込まれている。それに対して、非研究開発では、まず開発計画において試行錯誤の繰り返しが組み込まれることはない。レビューやテストというアクティビティやフェーズで誤りが摘出され､「戻り作業」が発生することはあるが、それによる進捗の遅延は許容されない。試行錯誤は実態的にはしばしば発生するが、あくまでネガティブに捉えられ、遅延の回復を強要される。最終的にプロジェクトの失敗（ＱＣＤ(品質(Quality)、コスト(Cost)、納期又は開発期間(Delivery)の大幅な超過又は遅延・延長、プロジェクトの中止・中断)も少なくないが、研究開発のように成果が出ないことが計画に織り込み済みで許容されることはなく、成果が出ることが当然且つ通常のこととされている。

ここで起こり得る疑問を挙げ、それに応える形で事態を一層明確にしたい。１つは、一般的なソ

フトウェア開発を直線的と捉えるのに対して、それはウォーターフォール型[211]の開発のことであって、反復型[212]もあり、必ずしもそうとは言えないのではないか、ということである。これは「反復」の意味・内容が異なるのであり、研究開発のような同一事項に関する試行錯誤の繰り返しを意味するわけではなく、異なる対象に対してアクティビティないしタスクを反復的に遂行するということである。簡単に模式的に図示すると、図表１－２-5 のようになる。反復型はアジャイル型を想定している。

図表１－２-5　一般的なソフトウェア開発プロセスの模式図



（出典：筆者作成）

もう１つは、研究開発のプロセスに関して、筆者のように反復的ないしループ的と捉えるのは一般的ではなく、「従来のモデル」は「第二次世界大戦以降、西欧で一般的に考えられているイノベーションの形態」の「リニアモデル」（linear model）であり[213]、それに対して「連鎖モデル」（chain-linked model）という「改良モデル」を提唱している、クライン（Stephen J. Kline）の有力な学説があるのではないか[214]、という疑問である。これは、同じプロセスと言っても、プロセス全体を俯瞰的に捉えたモデルに関する議論であり、筆者が問題にしているプロセスの内在的な作業レベルのこととは次元の異なる議論である[215]。同じ次元では、藤本隆宏が、研究開発における設

---

[211] ウォーターフォール型は、その名の通り、全般的に上流工程から順次下流工程に作業を進めていく開発方式であり、現在でも最も多く採用されている方式である。ポイントは、各工程で確実な成果物を作成し、後戻りしないことである。

[212] 反復型は、総称として括ったが、細かくは様々な流儀や名称があるが、大枠としては大同小異である。プロトタイプの作成から始めて、インクリメント(Increment：追加)並びにイテレーション(Iteration：反復)を繰り返して、段々に開発対象と範囲を拡大していく進め方である。個々のインクリメント並びにイテレーションのサイクルはウォーターフォール型と同様だが、大きな違いは、フル機能でなくとも、個々のサイクルの終了時にリリースまで可能なことである。

[213] Kline(1990)邦訳 p.16。なお、通商産業省編(1992)も、「技術革新(ここでは、新製品の開発、新製法の導入等を指す)の過程は、従来、基礎研究から派生した新たな科学の知見が技術の芽となり、応用研究、開発を経て商業化へとつながる一本の直線的道筋(リニアモデル)でしばしば説明されてきた」(pp.11-12)、としている。

[214] Kline(1990)邦訳 p.16。なお、研究開発マネジメントの世代論議に関しては原(2009)p.7 等参照。

[215] クラインの「連鎖モデル」でも、各フェーズ内並びにフェーズ間の「フィードバック・ループ」は設定されており、同一次元では捉え方は変わらない。

計に関して、「２段階設計プロセス」モデルを提唱していることが参考になる。「公理系設計論をベースに、設計行為を、不確実性下で製品機能・製品構造の連立方程式を解く、２段階設計プロセスによって近似する。第１段階は、構造・機能の因果知識が不完全な中での暫定解の導出、第２段階は、その暫定設計解から、最適設計解へと漸近する試行錯誤のプロセスである」[216]（傍点：引用者）。独特の用語を使っているが、演繹的なアプローチにおいて、個別具体的な諸条件に適合させていく試行錯誤のプロセスを定式化したものであり、筆者の捉え方を支持するものと解して差し支えないであろう[217]。

（３）ソフトウェア開発と研究開発のプロセスの定量的差異

ソフトウェアにおける研究開発と非研究開発のプロセスにおける形態的相違、すなわち試行錯誤の繰り返しの構造的な組込みの有無は、ほぼ必然的に定量的な差異を生じさせる。それは、生産性の著しい差異である[218]。従って、当該開発の生産性が如何なる水準にあるかによって、研究開発であるか否かは判然とする。但し残念ながら、生産性に関する公表された信頼し得る統計的な情報はない。特に研究開発の生産性は企業機密に属することであり、具体的な数値を取り上げることはできない。そこで本稿では、モデル的な論証に留めざるを得ない。

簡単な例を挙げる。開発フェーズは、設計・製造（試作）・テストの３フェーズとし、工程比率は各 30%・40%・30%とする。開発規模は 120KLOC とする。

①一般的なソフトウェア開発の場合

・設計工数：45 人月、製造工数：60 人月、テスト工数：45 人月、合計 150 人月、とする。

・生産性は、0.8KLOC/人月（＝120KLOC／150 人月）、となる。

・開発単価を 100 万円/人月とすると、直接開発費は 15,000 万円(=100 万円×150 人月)である。

②研究開発の場合

・設計工数：１回 45 人月を３回やり直すとすると、合計 135 人月、試作工数：１回 60 人月を３回やり直すとすると、合計 180 人月、テスト工数：１回 45 人月を２回やり直すとすると、合計 90 人月、となる。工数総合計は 405 人月となる[219]。

・生産性は、約 0.2963KLOC/人月（≒120KLOC／405 人月）、となる。

---

[216] 藤本(2011)p. 275。

[217] 設計ループに関しては、Suh, Nam P.(1990)邦訳 pp. 24, 39 参照。また、n次試作に関しては岡本(2005)p. 19 参照。

[218] 生産性(productivity)は、経済学の概念では、労働生産性と全要素生産性があるが(藤田・長岡(2011)p. 4 参照)、一般的にソフトウェア開発で取扱う生産性(KLOC/人月又はFP/人月等)は労働生産性のことである。他方、研究開発で取扱う生産性は主としては全要素生産性である(事業成果(OP)/R&D 投資(IP)、浦川卓也(2010)p. 159 参照)。但し、差異を比較考察する観点から、本稿では研究開発に関しても、ソフトウェア開発で取扱う労働生産性を個別プロジェクトに関して問題とする。

[219] 実務レベルでは、フェーズ単位というより、アクティビティないしタスク単位のより小さな作業単位でのやり直しの方が多いであろうが、モデル的に単純化した設定とする。計算の精粗や複雑さが異なるだけで、実質的なことは変わらない。

・開発単価を 100 万円/人月とすると、直接開発費は 40,500 万円(=100 万円×405 人月)である。

その差異が、歴然としていることがわかるであろう。例示では、研究開発の生産性は、一般的な開発対比 0.3704%と著しく低く、研究開発の直接開発費は、一般的な開発対比 2.7 倍と多額となる。公表された統計的数値はないが、例えば同一企業でソフトウェアを研究開発する場合と非研究開発する場合があれば、計画値並びに実績値は明確に差異があるはずである。

なお、一般的な開発でもプロジェクトの失敗によって大幅に低い生産性となる場合があり得るが、それは筆者が本論第６章で提示する仕損会計の処理を施すべきことであり、本章の考察に影響するものではない。また、もう１つ例外的な事態がある。研究開発で生産性がそれほど低くない場合である。天才的な、あるいは卓越した発想と能力を有する者が、常軌を逸した「全身全霊を賭した」とも言うべき献身により、作業に没頭して、ほとんど文字通り「不眠不休」で完成に辿り着くような場合であり、マイクロソフト、グーグル等の事例がある。しかし、そのような事例はあくまで「異例」のことであり、範例とはなり得ないと言うべきである。

ソフトウェア会計基準において、およそ研究開発ではないソフトウェア開発まで不当に拡大して研究開発と看做す規定が、アメリカ基準を発端とし、日本基準でもそれを再検討することもなく踏襲された。ドグマティックに研究開発と看做すために、研究開発の本来の判断規準である製品又は製法の新奇性が軽視され、替わって「技術的実現可能性」の確保前後を工程で区分する判断規準が窮余の一策で策定されたが、それらが何れも根拠のない不合理なものであることを明らかにした。日本基準設定前後に多くの論者が多くの論文等を発表したが、非研究開発を研究開発であると看做している規定に疑義を呈するものは僅かであった。その数少ない論考も、状況認識並びに発想としては概ね首肯できるものではあったが、更に進展させて、具体的にソフトウェア開発が研究開発である場合とそうでない場合を明確に区別する判断規準を提示するまでには到らなかった。それは、今日でも変わっていない[220]。セントラル・ドグマは今も護持されている。

そこで筆者は、セントラル・ドグマを解体し、適正な基礎を構築するために、プロダクトとプロセスの両面から判断規準を明確化した。プロダクトに関しては、機能と技術のライフステージ座標を設定し、個々のソフトウェアを測位可能とした。プロセスに関しては、まず新奇的なプロセスの事例を挙示し、続いて大半のプロセスはそれ自体新奇的ではないが、プロダクトの新奇性に適応的な研究開発とそれ以外の開発のプロセスが形態的並びに定量的に如何なる相違があるかを明確にした。これらにより、十全に研究開発と非研究開発との区別を確実なものとした。

この考察の帰結は明解である。研究開発である場合は、研究開発費会計の対象として会計処理す

[220] 企業会計基準委員会は、2010 年時点では現行基準を「維持」しつつ IAS38「無形資産」会計を重ね合わせた「二重基準」というコンバージェンスの方向を検討していたが(第 197 回企業会計基準委員会(2010 年 3 月 11 日)審議事項(５)－１(https://www.asb.or.jp/asb/asb_j/minutes/20100311/20100311_index.jsp))、2012 年時点では現行基準を「維持」し、変更しない方向を鮮明にした。

ることでよい。それに対し、非研究開発の場合は、研究開発費会計とは別個の独立的なソフトウェア会計基準を設定し、その会計処理対象とすることである。そして、基準において資産要件を設定し、それを充足する場合には資産計上とする。なお、資産要件は必要十分な規定、明示的には、(1)開発計画の策定(承認含む)、(2)開発予算の確保(執行含む)、(3)プロジェクト管理の態勢化(実施含む)とするべきである[221]、と考える。そもそも企業が効用もないソフトウェアの開発に少なくない投資をするなどと考えることが前提的に不自然である。また、一般的なソフトウェア開発にとって技術的実現可能性などは些少なことである。資産性の毀損に関しては、筆者が本論第６章で提示する「仕損会計」（当初）と、減損会計（それ以降）で対処すれば十分である。

本章は、全面的に刷新されるべきソフトウェア会計の中核的な考察である。それに基づいた会計基準自体の体系的、包括的な試案を策定することは、今後の課題である。全面刷新されるべきソフトウェア会計基準は、ソフトウェアのライフサイクル（開発～保守・運用～廃棄）に沿った会計基準としなければならない。開発におけるソフトウェア再利用、開発ツールの利用、プロジェクトの失敗による仕損、保守、運用、廃棄を包含し、且つ複合的な実態を的確に捉えたものにしなければならない。また、市場販売目的と自社利用目的で大きく異なる規準ではなく、収益の源泉としての直間性に基づいた、統一的な規準としなければならない。陳腐化した、あるいは事実誤認の例示も刷新しなければならない。そもそも「日進月歩」は誇張だとしても、技術革新の激しいソフトウェアの会計基準が、設定から 10 数年経過しているにも関わらず、僅かな改訂しか成されていないこと自体が怠慢の謗りを免れない。例えば、オープンソースやクラウド・コンピューティングが会計的に惹起している問題に対処する必要性を、筆者は痛感し研究を進めている。それらを完成した暁には全面刷新案を提案したいと考えており、本章の内容はその核心部分を構成することになることは言うまでもない。それ以外は、後続の本論各章で引き続き考察を行なう。

最後に、繰り返しになるが、ソフトウェア開発を研究開発と看做すセントラル・ドグマから解放され、ソフトウェア会計が正常化されるために、本章が一石を投じることを念願している。

---

[221] IAS38 の資産要件は一見包括的で周到なようで、研究「開発」と重ね合わせているので、ソフトウェア開発に関しては必ずしも適切な要件とは言い得ないものであり、見直しが必要である、と考えている。但し、本章は研究開発と非研究開発の識別が主題なので、詳細には立ち入らないが、(a)技術的実行可能性は後続の本文で言及するように必須要件としないが、それ以外の 5 要件（(b)使用又は売却の意図、(c)使用又は売却の能力、(d)有用性の立証、(e)資源の利用可能性、(f)支出の測定能力（第 57 項））は筆者の挙示する 3 要件に全て吸収され、且つ必ずしも具体的な捕捉が容易ではないという難点を有する。それに比して、筆者の 3 要件は遥かに具体的であり、証憑の取得も監査可能性も十分に備えている。

本論

第３章　ソフトウェア開発

１．ソフトウェア開発の概観

（1）ＩＰＡ「共通フレーム 2013」における開発

（2）機能と技術の立体的構造化

２．ソフトウェア資産の測定

（1）ソフトウェア測定の意義

（2）ソフトウェア原価計算

（2-1）原価計算の目的と対象

（2-2）ソフトウェア原価計算に関する先行研究のレビュー

（2-3）原価計算の方法

３．ソフトウェア資産の開示

（1）開示情報としてのソフトウェア測定項目

（1-1）ソフトウェアの規模

（1-2）ソフトウェア開発の生産性

（2）ソフトウェア資産の開示事例

（2-1）開示事例のケース設定

（2-2）開示事例の具体的数値

（2-3）比較評価

図表１-３-1　ソフトウェア＝機能と技術の立体的構造化モデル

第３章　ソフトウェア開発

ソフトウェア企画に続き、ソフトウェア・ライフサイクルの大フェーズとしては、２番目に当たるソフトウェア開発を取り上げる。前章である本論第２章で、ソフトウェアにおける研究開発と非研究開発を明示的に区別し、非研究開発には現行のソフトウェア会計基準とは異なり、研究開発要素はないものと論証したので、それを踏まえ、本章ではソフトウェア開発の大半を占める非研究開発としてのソフトウェア開発を取り扱う。その会計的な含意は、開発が明らかな失敗等でなかった限り、出来上がったソフトウェアは（受託開発を除き）資産となるということである。しかも、現行のソフトウェア会計基準のように、「市場販売目的のソフトウェア」と「自社利用のソフトウェア」とで資産要件や資産範囲が異なるといったことにはならず、資産要件や資産範囲は同一とすることが妥当である。また、開発の失敗に関しては、後続の第６章「ソフトウェア仕損」で取り扱うので、本章は「正常」な開発を想定した取り扱いとする。

本章は、まずソフトウェア開発を概観し、続いてソフトウェアを機能と技術が立体的に構造化されたものとして捉えることを明確にする。それらを踏まえて、ソフトウェアを全般的に資産として取り扱い、特に資産の測定を重点的に取り上げ、ソフトウェアに特有の原価計算を詳細に具体化し、資産額を確実なものとする計算方法を提示する。次いで、開示に関して、財務報告の比較可能性並びに信頼性を担保するために、資産額の基礎算定項目である規模・工数・生産性を財務諸表に注記すべきであることを提言する。

１．ソフトウェア開発の概観

ソフトウェア開発をまず概観するが、第１章「ソフトウェア企画」や後続の第 10 章「ソフトウェア廃棄」とは異なったアプローチを行なう。何故ならば、企画や廃棄はどのようなことを行なうのか、そのアクティビティやタスクは一般的には余り知られておらず、またソフトウェアに関する技術書等でも主題的に取り上げられることが少ないため、限られた文献等からではあるが、かなり立ち入って予備的に理解をしておかなければならないのである。そうしなければ、それを前提ないし背景とする会計的な考察を行ないにくいからである。それに対し、ソフトウェア開発は作業の詳細な内容はともかく、具体的なソフトウェアを制作するという点で遥かに理解しやすいものである。また、基礎論第１章～第３章で開発を様々に取り上げており、改めて取り上げる必要性はそれほどないのではないかとも思えるからである。そうした意味で、後続の考察に必要な範囲で、極めてコンパクトに内容を絞り込み、且つ多少ではあるが内容的に深化した形で取り上げることにする。

（1）ＩＰＡ「共通フレーム 2013」における開発

まずＩＰＡの「共通フレーム 2013」において、ソフトウェア開発がどのように取り扱われている

かを取り上げる。ソフトウェア企画等での取り上げ方と異なり、最新版である 2013 年版だけを取り上げる（取り上げる趣旨からすれば、それで十分だからである）。「2.3　システム開発プロセス」は、

「2.3.1　システム開発プロセス開始の準備プロセス」〔1(5)〕

「2.3.2　システム要件定義プロセス」〔2(1,2)〕

「2.3.3　システム方式設計プロセス」〔2(3,2)〕

「2.3.4　実装プロセス」→「2.4　ソフトウェア実装プロセス」〔9 小プロセス：1(5),1(3),1(7),1(8),1(5),1(6),1(6),1(2),1(3)〕

「2.3.5　システム結合プロセス」〔2(3,3)〕

「2.3.6　システム適格性確認テストプロセス」〔1(7)〕

「2.3.7　システム導入プロセス」〔1(2)〕

「2.3.8　システム受入れ支援プロセス」〔1(3)〕

という２つの小プロセス、その各々は〔　〕内のアクティビティ数と、その各々のアクティビティは(　)内のタスク数で構成されている[222]。「2.3.4　実装プロセス」は、特定の条件下では[223]、「2.4 ソフトウェア実装プロセス」に置き換わる、とされる。

「共通フレーム 2013」における開発のプロセス＞アクティビティ＞タスクの設定は、筆者からみれば、「方式設計」に偏重したものであると言わなければならない。例え「システム開発プロセス」全体の前に「2.2　要件定義プロセス」を設定し（これは「システム開発プロセス」内の小プロセスである「2.3.3　システム要件定義プロセス」とは別の「システム開発プロセス」と同格のプロセスである）、それを十全に遂行したとしても、設計とは明らかに粒度が異なるのであり、設計を代行（あるいは代替）するようなものではない。従って、方式設計と少なくとも同等に、機能設計を行なわなければならないはずである。ところが、機能に関しては、「システム開発プロセス」内の「2.3.3 要件定義プロセス」から一挙に「2.4　ソフトウェア実装プロセス」内の「2.4.2　ソフトウェア要件定義プロセス」並びに「2.4.4　ソフトウェア詳細設計プロセス」に進んでしまうようなアクティビティないしタスクの設定となっている（「2.4　ソフトウェア実装プロセス」内でも、方式設計はまた「2.4.3　ソフトウェア方式設計プロセス」として設定されている）。一般的な開発実務における設計プロセスでは、特段に技術要件が高難度の場合を除けば、機能設計に重点が置かれている。もう少し丁寧に言えば、組み込み系では機能と技術はより「密結合」[224]的であり、同時並行的な作業の必要性が高いが、エンタープライズ系では機能と技術は遥かに「疎結合」的なので、組み込み

---

[222] ＩＰＡ監修(2013)pp. 138-151, 152-168

[223] 注記で、「この共通フレームの利用者は、システム、ソフトウェア製品又はより大きいシステムのソフトウェア要素を扱うことを意図する。ソフトウェア実装プロセス(2.4)は、ISO/IEC 15288〔引用者注：システムライフサイクルプロセスの標準〕の実装プロセスの適合例で、ソフトウェア製品又はソフトウェアサービスを実装するという特定のニーズに特化したものである。ソフトウェア実装プロセスは、この共通フレームの実装プロセスに置き換わる」（同書 p. 144）、とされる。

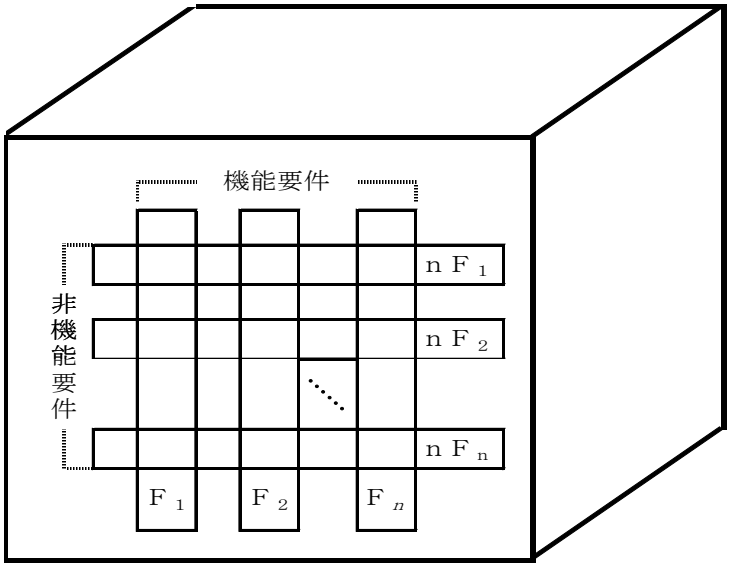
[224] 密結合や疎結合は、技術用語としての一般的な用法は、プログラム同士等の関係に関するものであり、それとは異なる意味であるが、簡にして要な表現なので、転義的に使用する。

系に比べれば、比較的分離して作業が進められる。しかも、技術的なことは概ね機能横断的であり、機能がほぼ個別に設計しなければならないのに比べれば、遥かに作業量が少なくて済む。従って、設計プロセスにおいて機能設計が占める割合が遥かに多いのが一般的である。作業量だけではなく、人間系との関わりがより強いので、質的にも難航する場合が多いのである。確かに方式設計を軽視しがちな風潮があり、それを是正したいという意図はわかるが、反動で逆に機能設計を軽視したプロセス＞アクティビティ＞タスクの設定となってしまっている。これでは標準として不適切である。

（2）機能と技術の立体的構造化

一方で、開発見積り等で使われるＦＰ法は、その名の通り機能に著しく偏重している。ＦＰ法に関しては、後続のソフトウェア資産の測定でより立ち入って取り上げるので、簡単に触れるに留めるが、測定対象を「機能要件」に限定している。それで総体としての規模の見積り等になり得るのか、十分に議論を尽くしたとは言えないまま、開発実務では相当広汎に使われるようになっている。かくして、片や「共通フレーム 2013」における技術偏重、片やＦＰ法における機能偏重、といったアンバランスは、開発実務に混乱を招くか、あるいは適当にチョイスして使うかといったことになり、標準の形成・確立にとって好ましい事態ではない。それに対して、筆者は既に前章の第２章で、機能と技術を区別しつつ総合的に捉えることを行なっている。ここでは、それをもう１段深化させる形で、改めて簡潔に取り上げることにする。

図表Ｂ−２−1　ソフトウェア＝機能と技術の立体的構造化モデル



（出典：筆者作成）

まず、用語に関する説明を行なっておく。技術は、ＩＴという場合のように、広義には機能を包含した意味で使われているが、狭義には機能と区別し、機能を実装する手段という意味である。機能は、広義には（技術との対比で）非機能要件を包含した意味であるが（「非機能」要件を機能と見做すのは、語彙的には矛盾のようにみえるかもしれないが、内容的な理解に基づいてのことである）、

狭義には非機能要件を含まず、何をするかに限定した意味である。

機能（要件）は一応タテ割に区別して図示したが、疎・密の程度は様々であれ、色々な関連があることは言うまでもない。非機能要件は、概ね機能横断的に関係しているし、非機能要件間で関連している場合もある（場合によっては、トレードオフの関係になることもある）。そうした広義の機能を実現するのが技術である。あくまで喩えだが、個々の機能や技術を点―線―面―立体へと構造化していくこと、あるいは当初の漠然としたイメージを次第に具体的で明確な形へと造形し整形していくことが、開発のプロセスであると言ったらよいであろうか。要件定義（要求定義）は、ドキュメント化する場合に概して箇条書きが多用されるが、それはまだ点ないし線でしかなく、機能設計によって具体的な仕様に落とし込み、それを方式設計（技術的設計）によって立体的に構造的な形あるものとし、更に進んでソフトウェア（プログラムと各種環境定義類の構造体）として（広義の技術として）実装化するのである[225]。

そして、非研究開発であるから、機能並びに技術は新奇的ではないが、個々の機能並びに技術の組み合わせ、更にそれを実装化していく進め方等は、精細にみれば、個々のソフトウェアにおいて皆異なっていると言える。そうであるにも関わらず、如何にして統一的ないし共通的に捉え、測定していくか、これがソフトウェアの測定、ひいては会計的な測定の主要な問題となるのである。

２．ソフトウェア資産の測定

本節では、現行会計制度におけるソフトウェア資産の取得原価の測定について取り上げる。「研究開発費等に係る会計基準の設定に関する意見書」では、「製品マスターは、（中略）適正な原価計算により取得原価を明確化できる」（三３(3)）と記載されているが、それ以上の具体的な規定はなく、実際には取得原価の測定方法は会計実務に委ねられている。しかし、後述するように伝統的な原価計算によって十分にソフトウェア成果物に係る原価を捉え得るとは言い難い。そこで、まず後続の考察に不可欠であるソフトウェア測定とは何かを概観する。次いで、ソフトウェア原価計算を取り上げ、原価計算の目的と対象と、これに関する先行研究を点検する。更に、原価計算の方法としてソフトウェア測定を援用したソフトウェア原価計算を筆者としては提言する。その上で、具体的には、工数と人月単価を用いて直接人件費を算出するための各測定項目について詳論する。開発実務

[225] なお、筆者は非機能要件を広義の機能と見做す位置付け・性格付けをして議論を行なっているが、ソフトウェア実務でコンセンサスが得られているわけではない。というより、要件定義並びに設計において、適切に俎上に乗り、周到に検討されるとは限らず、潜在化してしまい、開発の後半に入り、実装したものが開発に参画しているユーザーに見えるようになり、デモやテスト等で操作できるようになった段階で問題として顕在化し、少なくない戻り作業が発生する場合がしばしばある。しかも、ユーザーにとって非機能要件は機能ではなく、「技術要件」であり、非明示的でも開発する側が察知して対応すべきことと看做し、「争点」となることが少なくない（費用負担を含めて）。そうした事態を回避するには、予め要件定義並びに設計の課題として適切に位置付け、検討を怠らないようにするしかない。

では大凡実際に行なわれていることを再確認するという意味合いもあるが、必ずしも厳格に実施されているとは言いかねるし、問題を抱えてもいるので、改めて纏まった形で提示する。

なお、これ以降に取り上げる諸項目は、筆者が**ソフトウェアの基礎情報**として財務報告ないし統合報告において情報開示すべきであると考えているものである。本章では測定的な裏付け等を含めて取り扱うが、開示の仕方等に関しては機会を改めて別途取り上げることにしたいと考えている。

いずれにせよ、確かな測定に基づく資産計上によってこそ、ソフトウェア資産に関する財務情報の比較可能性並びに信頼性が担保されることを再確認しておきたい。

（1）ソフトウェア測定の意義

「ソフトウェア測定」(Software Measurement）とは、一般的には、ソフトウェアの成果物やアクティビティ（作業）等を定量的に測定することである[226]。国際標準として、1998 年に ISO/IEC14143「ソフトウェアの測定 - 機能規模測定」が定められたが、「機能要件」の規模測定に関するものであり、「技術要件」や「品質要件」は除外された限定的なもので[227]、規模総体の測定を取り扱ってはいない。また、工数やコスト等の測定項目を包含し、あるいは保守をも含めた広義のソフトウェア開発全般の測定に関する標準は未だに定められていない[228]。まして、ソフトウェア・ライフサイクル全般に関しては尚更である。ソフトウェアの広範囲な普及と対称的に、ソフトウェアに関する標準の未発達は憂慮すべきことであり、跛行的事態の１つの現れである。ともあれ、ソフトウェア測定は、ソフトウェアの技術面に関わる多くのことが対象となっているが、本章ではソフトウェア資産の原価に関わることに限定して取り上げる。

ソフトウェア測定の目的は、ISO/IEC15939:2007(JIS X 0141:2009)「システム及びソフトウェア技術ー測定プロセス」に拠れば、

①「ソフトウェアプロセス及びソフトウェア製品の管理及び改善を支援する」、

②「ソフトウェアライフサイクル活動を管理し、プロジェクト計画の実行可能性を評価し、プロジェクト活動がそれらの計画に準拠しているかどうかを監視する」、

③「ソフトウェア製品の品質及び組織のソフトウェアプロセス能力を評価するためのかぎとなるものでもある」[229]

---

[226] ＪＵＡＳの推奨する、ソフトウェア･メトリクス(Software Metrics)という類似の分野･取組みがあるが、それは「開発、保守、運用の評価尺度」を「ソフトウェアメトリクスと呼ぶ」とし（ＪＵＡＳ(2010)p.4)、これに関する調査は開発・保守・運用の全般的事項に及んでおり、参考になる。

[227] 情報処理推進機構(2006)p.82

[228] ＣＭＭＩ(Capability Maturity Model Integration：能力成熟度モデル統合)の成熟度レベル２のプロセス領域である「測定と分析」(Measurement and Analysis)やISO/IEC15939：2007(JIS X 0141：2009)「システム及びソフトウェア技術―測定プロセス」は、測定の「プロセス」に関する標準であり、測定項目や測定方法に関する具体的な規定は、２、３の例示程度である。

[229] 日本工業標準調査会審議(2009)p.41。なお、この目的の記載はISO/IEC標準のIntroductionにはあるが、その日本版のJIS規格の序文では削除されており転載されている解説からの引用である。

とされる。

ソフトウェア測定の対象プロセスはライフサイクルの全体に及ぶが、焦点となるのは、①企画フェーズないし開発フェーズの開始時点の見積り、②開発フェーズの終了時点の実績測定、③保守フェーズの開始時点の見積りと終了時点の実績測定、④運用フェーズの利用状況の測定、である。①の見積りは、開発計画の策定や委託開発における契約金額の算出等のため、精度や方法は様々であれ、ほぼ必ず行なっていると言える（③の見積りは①に準ずる）。それに対し、②（③の実績測定は②に準ずる）と④は余り行なわれていないのが実状と言える[230]。つまり、ソフトウェア測定で最も多く実施されているのは開発見積りであり、それを取り扱っている文献等は少なくないが、ソフトウェア原価計算との関連では、考察対象とするのは主として開発の実績測定である。原価として使用するからであるが、理由はそれだけではない。開発実務において、見積りはそれをしなければ、いわば業務が始まらないという意味で必要に迫られて行なっているが、実績測定は何故か軽視されている。しかし、第１に、見積りは多くの仮定を積み重ねて行なうものであり、決して確度は高くないし、しかもそれ自体で当否の検証はできず、実績との突合によるしかないのである。第２に、それには実績測定から見積りへのフィードバック・ループを構築し、継続的に作動させなければならないが、その「起点」はあくまで実績測定でなければならない（作業の時間的な順序としては終点かもしれないが）。第３に、従って、まずは測定が高確度である実績を「土台」としたソフトウェア測定が確立されるべきなのである。そこで、高確度な測定という視座で、開発実績の測定諸項目を精査したいと思う。

（2）ソフトウェア原価計算

（2-1）原価計算の目的と対象

ソフトウェア原価計算を行なう目的は、ソフトウェア資産の測定であり、原価計算基準第一章一（一）に挙げられる、財務報告に役立つ真実の原価の集計にある。ここでは、価格計算や原価管理等の内部管理的な目的は取り上げず、財務会計上の原価計算に焦点を当てる。拠って、仮に第三者との取引のために開発したソフトウェアであっても、あくまで測定対象は開発原価である。原価計算の対象は、自社開発ソフトウェアとする（委託開発であっても、受託企業は同様な計算を行なっているのであり、ただ利益を上乗せした売価の形態と取っているだけで、実質的な内容はそれほど変わらない）。更に、人件費が開発原価の大半を占めるので、直接人件費の測定に限定する。その際に考慮すべきことは、ソフトウェア開発の特性として、プロジェクトの進め方や開発技術者の能力

[230] 実績測定について、ジョーンズは「2008 年時点では多くのソフトウェア組織は測定をほとんどしていない」（Jones（2008）邦訳 p. xv）と指摘している。日本工業標準調査会審議（2009）でも解説で、「ソフトウェアの測定がソフトウェア開発にとって無視できない重要なものであるにもかかわらず、国内においてはこれまで必ずしも十分行われているとは言い難かった」（p. 41）と述べられている。

等により、作業内容及び作業量が異なり、人件費は変動することである。又、知的労働であるので、原価計算における一般的な労働時間の区分は適さないことである。更には、成果物が無形なので、成果物と作業との対応関係が捉えにくいことを承知しておかなければならない。

そこで、これらの特性に対処するため、ソフトウェア測定を援用するのである。原価計算に用いるのは、開発の業容に即した時間尺度である工数という尺度で測定した作業時間と、工数に対応する単価であり、それにより原価を測定する。更に、原価額のみではソフトウェアの特性が鏡映されているとは言い難いので、情報開示において、必要な範囲でソフトウェア測定項目を援用する。規模により無形のソフトウェア成果物を捉え、その規模と工数による作業時間との対応付けに評価尺度として生産性という指標を用いる。生産性は、ソフトウェアとその開発とを可視化するための有力な指標なのである。なお、ソフトウェア開発に特徴的でないことは、原価計算基準に則った原価計算に準じ、簡単に触れるに留める。

（2-2）ソフトウェア原価計算に関する先行研究のレビュー

ソフトウェア原価計算に関する先行研究は、主に５種類挙げられる。それらは、①櫻井通晴(1992)及び櫻井通晴(2006)、②太田昭和監査法人(1992) 、③Wellman,Frank(1992)、④ＪＩＳＡ(2003)、⑤井手吉成佳(2010)である。なお、①は櫻井通晴が著者ないし編著者なので併せて１種と見做す。③は訳者が解説している通り、原題は「ソフトウェア原価計算」(*Software Cost Accounting*)だが、開発見積りや原価管理が主たる内容で、原価計算の専著と見做すのは余り適切ではない。同様に、①と②も過半は原価管理等が占め、原価計算は１部分でしかない。しかも①の前著と②及び③は、ソフトウェア会計基準設定以前のものである。会計基準の設定が直ちに原価計算に影響するわけではないにせよ、それ以降に専著がなく 20 年近く経過していることは、ソフトウェア会計の研究の低調さを象徴しているように思える。これ以後も基準解説の実務書以外では、財務会計の調査・研究はソフトウェアの業界団体であるＪＩＳＡがあくまで実務的な立場で断続的に行ない、貢献している程度である。それ故に、④は原価計算の専著ではないが、その中に「原価計算規程」並びに「原価計算マニュアル」が含まれており、会計基準を踏まえて出された、ソフトウェア原価計算に関する纏まった唯一の単行書と言える。これが外見的な梗概である。

内容を詳細に論及するのは、上記のうち①と⑤とする。理由は、①は長らく日本のソフトウェア会計を専門とし、実務界とも密に連携する中で独自に研究を進め、多大な貢献をしてきた言わば第一人者であり、その著書は同分野の必読文献とも言える程大きな影響を及ぼしているからである。⑤は、会計基準設定以降のものであり、櫻井の研究成果に全面的に依拠しながらも、それを「敷衍」して新たなソフトウェア原価計算を試みた、数少ない研究の１つだからである。但し、両者とも原価計算の目的は内部管理であり、本章が目的とする財務報告に係る資産測定とは異なるが、ソフトウェア測定を取り上げているので、原価計算とソフトウェア測定の関係如何を主に取り上げる。

まず、櫻井のソフトウェア原価計算は、①の前著の状況認識として、大幅な生産性向上を実現し

てきたハードウェアに対し、ソフトウェアはごく僅かな生産性向上に留まっているという当時において周知の認識に立脚し、それを向上させる有力な方策として提唱したものである。その提唱に当たっては、「工業生産物の原価計算とは基本的に異なる立場から論ずることも可能である。しかしながら、本書ではそのような立場をとらなかった」[231]と明言した上で、基本的にはハードウェア原価計算と同じ発想で原価計算を実施すべきであるとする立場を貫いている。それは、ハードウェアとは異なるソフトウェアに即した捉え方があることを承知の上で、敢えて当時の状況との兼合いにより、伝統的な原価計算すら十分に確立されていないソフトウェア分野への原価計算の定着・浸透を企図した、ソフトウェア原価計算の「市民権」を得るための過渡的な方策だったと言える。ところが、その立場を敢えて選択した断り書きが無視され、表面的に後続の研究者等に踏襲されてしまっていると思えてならない。その後、2001 年に櫻井通晴(2006)の初版が刊行されたが、そこでは前著を絶版とし、改めて加筆修正したソフトウェア原価計算を取り上げている。新版でのソフトウェア原価計算の内容は、「ソフトウェアの種類、開発形態、技術レベル、および組織によって異なるのである」[232]とした上で、各々に適した処理方法を取り上げている。ハードウェアと同様の原価計算が適用されるのは、受注ソフトウェアの開発に対してであり、専ら個別原価計算が適しているとする。内容的には前著と大凡の主旨に変わりはない。だが、原価計算目的の１つに製品の価格決定があるが、そのソフトウェアの価格決定に関する箇所で、既述の「意図」についての発展を観ることができる。主に受注ソフトウェアの価格決定は、日本では従来より、ソフトウェアの行数を基礎としたコスト・プラス方式が採用されてきた。しかし、この方式による限り、開発受託側の原価低減や独創性の高いソフトウェアの開発の意欲を失わせ、効率的経営の阻害要因であるとして、この価格設定に疑問を持ち、原価管理等の観点からも「目標原価」を設定する必要性を説いている。そこで、その算定方法としてソフトウェアの開発原価の見積り手法を取り上げ、ＦＰ法が最も有望視されるが、「工数の見積もりの方法であって、ソフトウェアの価値を見積もる方法ではない」[233]としている。

この考察で支持できる点は、開発実務で広く用いられている原価の見積り手法を丁寧に取り上げ、それに基づいて価値測定を考察している点である。また、これと関連して実態を把握した上で広く内部管理的側面の問題を解消・改善していこうとする点である。これらは、開発実務に即した会計研究として貴重である。他方、問題点は、実績を含めたソフトウェア測定の取り扱いとしては、体系的とは言い兼ねる点である。また、櫻井も指摘しているように、ソフトウェアの見積り手法は、価値測定の手法ではなく、ソフトウェアの測定手法であるので、即自的に価値を示すものではない。見積りや実績測定においてソフトウェア測定で定量化されたソフトウェアを基に、何らかの価値を付与するのである。従って、ソフトウェア測定と価値測定を如何に連携させるかという視座が不可欠であり、それに関する、より具体的で緻密な考察があって然るべきである。更に、櫻井は原価や

---

231 櫻井(1992)p. iv
232 櫻井(2006)p. 38
233 同書 p. 71

価格の算定方法をソフトウェアの利用目的により異なる捉え方をしているが、あくまで統一的に捉えるべきである。それを可能とするのが、ソフトウェア測定なのである。ソフトウェア測定は如何なるソフトウェアに対しても基底的であり、如何なる管理目的に対しても汎用的だからである[234]。

次に、井手吉のソフトウェア原価計算は、櫻井の研究に全面的に依拠しており、同様の方向性の下にソフトウェアの価格決定を目的とし、そのための目標原価の算定にソフトウェアの定量化手法の１つであるＦＰを専一的に採用している。つまり、ソフトウェアへの標準原価計算の適用を主張し、その規定値の設定にＦＰという機能規模尺度を用いるべきだと提案しているのである。この考案で支持できる点は、櫻井の研究に促されて原価計算の種々の手法に対してＦＰの具体的な適用を試みている点である。

しかし、問題点も少なくない。第１に、伝統的な実際個別原価計算では、ソフトウェアの価格決定を目的とした管理において的確に機能することができないとし、その善処策として標準原価計算の適用の意義を主張しているが、開発実務においても目標原価の算定と近似的な開発原価の見積りは行われており、新たな考案と言える程の意義は乏しい。更に問題なのは、実務における現況を調査・分析した上で適用可能性を検討するべきであるが、それはなされていない。開発実務における原価見積りで最も問題なのは、実績測定値が次期開発の見積りへフィードバックされていないことである。そういったサイクル及び態勢を整備することが先決であり、それに如何なる目標原価の算定方法ないし見積り手法を用いるかは二次的な問題である。そして、サイクルの「起点」はあくまで実績値である。井手吉自身も指摘しているように「ＦＰ法を導入することによるソフトウェア原価計算手法で算定された原価情報に加えて、従来の個別原価計算による原価情報も利用することで、原価情報の改善ができる」[235]のであり、目標原価による管理上の有効性の向上を真に実現するためには、両者を含めた態勢整備が不可欠なのである。

第２に、ＦＰに疑問が呈されていないことである。ＦＰが最も標準化された計測手法であることを強調した上で、原価標準の設定に関してＦＰ値とＦＰによる規模単価を提案している。しかし、ＦＰには様々な効用があることは確かだが、あくまでソフトウェアの機能要件の規模計測であり、非機能要件は計測対象から除外されているので、ソフトウェアの規模を総体的に捉えたことにはならない。ソフトウェアの規模尺度としては完備性に欠けており、他の要件をも原価標準に加味するか、又は不備を考慮したＦＰの標準値の設定の考案が欠かせないのだが、それに対する考慮が欠落している。それ故、ソフトウェア製品の比較可能性について、機能によるソフトウェア成果物の比較が有効であると説いているが、不完全な比較可能性でしかないのである[236]。

以上のように、これらの先行研究には問題が多く、依拠することが難しいので、独自に考察を進

---

[234] 櫻井の考察の核心部分に焦点を絞り、個々の論点に対する批判的コメントは割愛するが、筆者の論文全体がそれに応えていると考える。

[235] 井手吉(2010)p. 168

[236] この他にも多様にＦＰの適用を探っているが、主題からは外れるのでコメントは割愛する。

めていく必要がある。その前に、筆者の考察の意義並びに先行研究との相違点を明らかにしておく。測定の目的は大枠としては現行制度下におけるソフトウェア会計への寄与を企図しており、測定の対象はソフトウェア資産の取得原価である。先行研究とは目的並びにアプローチが異なるが、価値測定という点では共通性はある。だが、ソフトウェアの価値を捉える議論に関しては、前提的に整備すべき段取りがあり、それを踏まえて行なうべきだと筆者は考えている。それ故、現時点では、ソフトウェアの資産価値を取得原価で表す現行制度上で、価値測定の精緻化を目指すことに、本章の有する意義があると考えている。また、ソフトウェア測定に関しては、既述の通り、如何なるソフトウェアを測定する場合においても基底的であり、且つ高精度・高確度で測定可能なのは実績であるので、先ずは実績測定を実務において確立することが、何よりも重要であると考えている。

（2-3）原価計算の方法

原価計算の方法は、一般的な原価計算の流れに従い、費目別計算→部門別計算→プロジェクト別計算という流れとなる。ソフトウェア開発では、いずれの場合にも特定のプロジェクトを編成することが基本となるので、該当プロジェクト毎の個別原価計算を採用することが合理的である。ここでは、プロジェクト別計算における直接人件費の測定を対象とする。具体的には、ソフトウェアの開発に要する作業時間から工数を算定し、これに技術者の作業単価を乗じて算出する。一見、計算構造としては伝統的な原価計算と変わらないが、原価算出の要素である時間と単価の捉え方がソフトウェアあるいはソフトウェア業界に特有のものである。以下では、それぞれについて詳論する。

①ソフトウェア開発の工数

最初に、工数を取り上げる。開発実績の測定では、工数は直接的に測定可能である。工数の尺度は、人年、人月、人日、人時である。人月は１人の技術者等が１ヶ月稼働したことを示し、人日は時間単位が１日、人時は１時間と、粒度は異なるが測定方法は同様である。一般的には人月を使うことが多いが、小規模開発や保守では人日ないし人時を使うこともある（大規模開発の場合の概数として人年も）。注意を要するのは、測定の目的により異なる算定をする必要があることである。まず、時間に関して原価計算では「労働時間の範囲」として、勤務時間＞就業時間＞実働時間＞直接業務時間、といった区分をしている[237]が、ソフトウェア労働には殆ど適さない区分である。また、そのような区分による差をネグレジブルと看做さざるを得ない程コンピュータ創成期以来、連綿として長時間労働が常態と化している。更に、ソフトウェア労働は知的労働であり、休憩時間や手待ち時間を区別しにくく、そもそも「実働」しているのかを測定することは難しいのである。唯一、開発プロジェクト外の会議出席等の時間は区別できるので、直接業務時間と間接業務時間の実効的な区別はできる、といった程度である。

[237] 太田昭和監査法人(1992)pp. 63-64

それに対し、作業に関する「実態ベース」と「コスト・ベース」の工数の捉え方の区別が必要であり、伝統的な原価計算にはないものである。「実態ベース」は実作業時間に基づくが、「コスト・ベース」は労働に対する支払対価に基づいて工数を算出するものである。時間管理対象外の管理職に関しては、例えば所定労働時間が 168H で、１ヶ月 200H 働いた場合に、「実態ベース」の工数は 1.19 人月であるが、「コスト・ベース」では残業代が出ないため、固定で 1 人月である。時間管理対象の社員に関しては、１ヶ月 200H 働いた場合に、「実態ベース」でも「コスト・ベース」でも 1.19 人月の工数となる[238]。開発の生産性を導出するためには、工数を「実態ベース」で捉える必要があり、コストを導出するためには、工数を「コスト・ベース」で捉える必要がある。人件費の導出には、単純化して、諸手当等がなく時給ベースで、残業の割増率を一律 40%とすれば、1.26 人月となる[239]（但し、社員に関しては給与計算によるので、説明のための例示である）。社外要員で契約により時間精算する場合に、150～180H は固定、それを超過あるいは未満の場合は増減させる時間精算で、増減の割増率は 20%という契約を締結している場合（前記に挙げた条件枠連動の場合）は、「コスト・ベース」では 1.13 人月となる[240]。何れの測定値も、目的・用途に応じて必要となり、これが「実態ベース」と「コスト・ベース」である[241]。コストを算出するには「コスト・ベース」を使用し、生産性を産出するには「実態ベース」を使用する。更に、生産性等を精細な粒度で導出するためには、総工数だけでなく、粒度別の内訳情報が必要となる（フェーズ＞アクティビティ＞タスク・ベース[242]）。測定の粒度に応じた「日報」等の作業報告を資料として測定することになる[243]。

②ソフトウェア開発のコスト（費用）

次に、コストを取り上げるが、ここでは要員別コストを算出する人月単価を中心に取り上げる。人月単価は、ソフトウェア企業や委託するユーザ企業の開発関係者には周知のことであるが、部外者が具体的な金額はもとより、設定内容まで知ることは企業機密に属することであり、実はかなり

---

238 工数(人月、実態ベース)＝月間総労働時間／月間所定労働時間＝200/168≒1.19。先に挙げた管理職の工数算定でも「実態ベース」は同様である。

239 工数(人月,コスト･ベース)＝{月間所定労働時間＋(月間総労働時間－月間所定労働時間)×割増率}／月間所定労働時間＝｛168＋(200-168)×1.4｝／168≒1.26。

240 工数(人月,コスト･ベース,固定枠上限超過の場合)＝{月間固定枠上限労働時間＋(月間総労働時間－月間固定枠上限労働時間)×割増率}／月間固定枠上限労働時間＝｛180＋(200-180)×1.2｝／180≒1.13。

241 実務でも截然と使い分けられているとは限らないし、２つの工数の識別、必要性を明示している研究等は、管見の限り、見当たらない。

242 これらは開発の作業区分を示し、フェーズ＞アクティビティ＞タスクの順に作業粒度は細かくなり、タスク・ベースが最小の作業単位となる。

243 櫻井(1992)は先ず費目から取り扱い、その基礎データたる日報や作業報告書は取り上げていない。太田昭和監査法人(1992)は、フォーマット例が載っている（pp.70-72）が、例示の限り、作業粒度はタスクまで精細となっておらず粗い。まして機能＞プログラムが特定される記載欄がないので、個々の機能＞プログラム単位の生産性は測定することができない(他の例も同様である)。

難しい。言うまでもなく、公表数値といったものはない[244]。

そこでまず、ソフトウェア企業にとっての受託開発における、第三者との取引に用いる売価としての人月単価にアプローチする。これには概ねソフトウェア業界の相場と言えるものがあり、二重の階層がある。１つは、スキル別（職種別）の階層で、プログラマー、ＳＥ、プロジェクト・マネージャー、システム・アナリスト等で各単価が設定されている。もう１つは、企業格付け的な格差で、最上位に「５社体制」（日本ＩＢＭ・日立・富士通・ＮＥＣのメーカー４社と、旧電々公社の後身として別格の企業であるＮＴＴデータ）、次に大手Ｓｉｅｒ、それ以下に何段階かの「下請」企業群があり、それぞれの階層で利益や管理費を確保して再委託するので、下方に行く程安くなる。何れも、ごく大まかに10～20万円ないしそれ以上の価格差がある。それが各コスト構造に反映する。ユーザー企業を対象とするＪＵＡＳの調査結果は、「費用」として加重平均単価 124.6 万円/人月（2008 年度調査は 116 万円/人月）が示されているが[245]、これは委託企業にとっての「費用」であり、受託企業にとっては売価としての人月単価である。同調査では、個別プロジェクトにおける「実績外注比率」（＝実績外注コスト／実績総費用）は、平均値が 72.9%（2008 年度調査は 74.1%）であり、実績外注比率が 100%（丸投げ）の開発プロジェクトが 21.6%（2008 年度調査は 22.1%）となっている[246]。つまりは、個別プロジェクトにおける凡そ４分の３程度はソフトウェア企業への委託である。そして、平均単価額を上下して、上記で触れた二重の価格階層が拡がっているとみて大過ないと言える。それ以上具体的且つ精細な数値は、公表資料に基づく限り示すことができない。

次に、自社で開発する場合の原価としての人月単価に関して、参考として、情報処理サービス業の企業を対象とする情報処理推進機構の『第 29 回　情報処理産業経営実態調査報告書（2007 年度調査実施）』を利用する。ＪＵＡＳ調査と対象年度・企業等は異なるが、当該年度で調査を終了しているため、それ以降は追跡できない。調査結果は、対売上高比の営業利益が 4.2%で、売上原価・販売及び一般管理費が 95.8%となっている[247]。売上の大半がソフトウェア開発の受注で、且つ販管費等の間接費も原価の人月単価に含めているとすれば、ＪＵＡＳの平均単価に対し、119.4 万円/人月（2008 年度調査値に対しては 111.1 万円/人月）が原価の人月単価となる[248]。同調査における対象企業数 723 社のうち、受注ソフトウェア開発を主たる業務とするのが 528 社（73%）で、この 528 社の企業の売上が対象企業全体の売上の、大企業（52 社）では 78.8%、中小企業（476 社）では 59.9%を占めている[249]。これは実績値であるが、予定単価の設定方法となると、企業で様々であり得るが、直接費の範囲で設定する場合、事業部等の範囲で間接費を含めて設定する場合、全社的な間接費を

---

244 下記の情報は、筆者が業界関係者からのヒアリングにより得た情報である。
245 ＪＵＡＳ(2010)p. 91
246 同書 p. 112
247 ＩＰＡ(2008)p. 10
248 119.4 万円/人月≒124.6 万円/人月×95.8%（2008 年度調査は 111.1 万円/人月≒116 万円/人月×95.8%）。
249 同書 p. 5

含めて設定する場合等が想定できる。

このように設定した人月単価から、「実態ベース」と「コスト・ベース」の工数に基づいてコストの金額が算出される。「実態ベース」の工数と人月単価からは、実働を反映したコストが算出されるが（参考値）、取得原価として求められるのは、「コスト・ベース」の工数と人月単価から算出される支払額（受託企業にとっての売上）である。そして、個々の要員毎に算出した毎月の金額をプロジェクト要員全員分集計し、更にプロジェクト期間の全期間分を集計すれば、開発コストを算出することができる。

なお、社員の場合には給与ベースの人件費で算定すると一見異なるように見えるかもしれないが、プロジェクトが社員と社外要員で編成されることが少なくない実態を鑑みれば、統一的な測定を行なう意味から、給与ベースから人月単価を算出し、それ以外は全く同様とすればよい。

３．ソフトウェア資産の開示

ソフトウェア資産の開示（の拡充）では、（1）で開示情報として不可欠であるソフトウェア測定項目の規模、生産性について詳論する。（2）で具体的な開示事例を提示し、ソフトウェアの資産額及び測定項目を用いて比較可能性並びに信頼性を検証する。

これまで、ソフトウェアの資産額の測定について、ソフトウェア測定を援用した考察を行なってきた。しかし、ソフトウェア資産を把握するには、単に金額の多寡だけで十分だとは到底言い難い。そこで、財務報告にとり真に有用なソフトウェア資産の情報とするには、ソフトウェア測定の諸項目を補足して加える、注記としての開示情報の拡充が必要不可欠だと考える。先には原価計算に必要な範囲で、工数及びコスト（人月単価に基づく集計値）のみを取り上げたが、ソフトウェア資産を把握するには、それらと併せて他のソフトウェア測定項目をも示す必要があり、積極的な開示を求めたい。以下では、規模及び生産性という測定項目を取り上げるが、これらは先の測定項目と一連のものであり、セットで示してこそ意義があることを強調したい。そして、これらの測定項目の開示が情報価値を高める上で如何に有意義なものであるかを、具体的な数値事例を用いて説明する。

（1）開示情報としてのソフトウェア測定項目

（1-1）ソフトウェアの規模

規模は、ソフトウェア測定の基幹部分であり、これを的確に捉えない限り、それ以降の導出は「空中楼閣」になってしまう程の重要性がある。開発実績は成果物により測定するため、直接的に測定可能である。成果物を大別すると、ソフトウェアのリソース類と、ドキュメント類がある。リソース類は、やはり大別すると、プログラムと、各種環境定義類（画面・帳票定義、ＤＢ定義、トラン

ザクション定義、ジョブ定義等）である。ドキュメント類は、設計書、テスト仕様書（テスト結果含む）、管理資料、その他である[250]。リソース類並びにドキュメント類は、ソフトウェア開発の成果物としては全てが包含されるが、規模を測定するためには主資料・補助資料の適切な選定が必要である。ドキュメント類は、その種類と量の測定結果をソフトウェア規模に換算する場合には相当変動的で、規模測定の主資料としては余り適切ではない。それ故、リソース類を主資料として規模を測定することが適切であると考える。しかも、プログラムとそれ以外のリソース類も対象とするのが妥当である[251]。それ以外のリソース類は、各工程を通じて不即不離に関わり、ソフトウェアとしての成果物であるから、包含して然るべきであるが、プログラムとは性質が異なることから、別個に各種環境定義類毎の規模を測定して、区分集計した上で総合計を集計する。リソース類の規模の測定尺度は、大別すると、ＬＯＣとＦＰである。

①ＬＯＣ

ＬＯＣ（Lines Of Codes：コード行数）は、プログラムの行数（ライン数、ステップ数とも言う）のことである[252]。まず、プログラムの行には、５種類ほど性質の異なる行がある。ケーパーズ・ジョーンズに拠れば、1.「実行文」（計算等をする処理実行行）、2.「データ定義」（処理に使用するデータを規定する行）、3.「コメント行」（処理内容等を自然言語で説明する行）、4.「空白行」（セクション等を視覚的に分ける行）、5.「デッドコード」（処理実行しないが、履歴として残す行）である[253]。そこで、測定対象とする行の規準が必要になる。ジョーンズは「ＬＯＣ尺度には、(1)論理行ベース、(2)物理行ベース、の２つがある」[254]としているが、両者を区別するのは、言語の特性やプログラミング規約等により、論理行での１行が物理行では複数行となったり、あるいはその逆もあり、行数が変わり得るからである。一般的には、ソフトウェアを継続的に開発・保守する企業では、プログラム標準又はプログラミング規約を制定しており、それと親和的・整合的な測定規約を制定し運用することが望ましい。また、同一の機能・処理であっても言語の種類により行数が異なるので、行数と併せて言語の種類を明示するか、言語毎の規模の測定としなければならない[255]。

②ＦＰ

---

[250] マニュアル類は開発当事者が作成する場合も少なくないが、運用のドキュメントなので含めない。
[251] このことに関しては、ソフトウェア基礎論第１章「ソフトウェアの定義」で詳論しておいたので、ここでは再言しない。
[252] ここで言うプログラムは、厳密には、ソース・プログラムのことである。
[253] Jones(2008) 邦訳 pp. 59-60。用語・説明はジョーンズの訳文を参照したが一部改変した。
[254] 同書 p. 276
[255] 言語間の規模の比較可能性に関する善処策としては、プログラミング言語同士の「言語補正係数」(Laird(2006)邦訳 pp. 49-52, 54-55) がある(例えば、COBOL の 73 行と、C 言語の 148 行と、VB の 50 行と、Java の 60 行が同等といった)。これを使って何れかの言語ベースに換算した規模であれば、統計的近似として比較可能である。

ＦＰは、1970年代末にＩＢＭのアラン・オールブレヒト（Allan Albrecht）が考案した規模測定概念から発展したものであり[256]、その直系とも言えるＩＦＰＵＧのＦＰ法が「標準」と言えるので[257]、それを主として取り上げる。ＦＰ法は自ら宣揚するように、最大の利点は「利用者視点」である[258]。それに対してＬＯＣは「開発者視点」と言えよう。ＦＰ法は、「基本的には論理設計に基づき、利用者に供される機能を定量化することによりソフトウェアを測定するもの」であり、その目的は「利用者が要求し受けとる機能を測定する」ことと、「開発で用いられる技術には関係なく、ソフトウェアの開発および保守について測定することである」[259]。ＦＰの「計測手順」は、1.「計測種別の決定」→2.「計測範囲およびアプリケーション境界の設定」→3.「データファンクションの計測」、4.「トランザクションファンクションの計測」→5.「未調整ファンクションの決定」→6.「調整係数の決定」→7.「調整済ファンクションポイントの計算」である[260]。1.と2.は前処理ないし準備段階と言え、3.～7.が本格的な計測であるが、これも大別すると、3.と4.と5.が第１段階、6.と7.が第２段階と言える。ＦＰの原初形態又は発想は極めてシンプルで、「ソフトウェアアプリケーションの入力、出力、照会、論理ファイル、インターフェースの５つの外的視点に基づくものである」[261]。それに「複雑度」等を加味して実態に即した規模を計測する。

ＦＰは、理念的にはプログラミング言語には依存しないという利点がある。しかし、項目数等は実数を直接計測するが、その項目数の僅かな違いにより、複雑度等のレベル低・中・高が変わる。更にその複雑度等のレベルに対して設定されているポイント数が離散的に変わる。そして、項目数の集計以後は、マトリクス表による「指数」化と更に調整係数による「指数」化が行われ、二重の「指数」となる。特に調整係数は、実績であっても解釈の余地があり、確度はそれ程高くない。また、機能要件のみが計測対象であり、総合的に規模を計測するには、技術要件並びに品質要件に関する補訂が不可欠となる。

③ＬＯＣとＦＰの利用の仕方

ＬＯＣとＦＰの利用の仕方を纏めるが、その前に別個に取り上げてきたので、各々を比較検討したい。まず、開発の実績測定に限定すれば、ＬＯＣは殆どが「実数」の測定であるのに対し、ＦＰは数段階の調整によって変換した「指数」であるから、ＬＯＣの確度が高い。次に、ＦＰ法は、理

256 ＩＦＰＵＧ(2005)p. iii

257 ＪＵＡＳ(2010)p. 39では、ＦＰ計測手法の調査結果を載せているが(回答192件)、ＩＦＰＵＧが109件（57%）で「６割近くを占めている」。それ以外の手法でも、「自社独自の基準でＦＰ値を計測している例も４分の１にのぼっている」とコメントしている。

258 ＩＦＰＵＧ(2005)p. 2-2

259 同書p. 2-2

260 同書p. 2-3。なお、引用に際しては、図示を文章化し、1.等の項番付けを筆者が施した。また、用語に関して、筆者は本論文全体では主として「測定」を用いるが、ＦＰに関わる箇所では典拠としているＩＦＰＵＧの「計測」を尊重し、混用しているが同義と理解されたい。

261 Jones(2008) 邦訳p. 51

念的にはプログラミング言語には依存しないが、機能要件のみで、非機能要件は計測対象外なので、それらの考慮を独自に行なわなければならない。この点、ＬＯＣはプログラム並びに各種環境定義類にそれらが体化されていると言える。その限りではＬＯＣが優位である。ＦＰ法が普及しても尚、ソフトウェア開発の実務でＬＯＣが使われているのは、技術者にとっては「実感」の持てるものであり、実績に関しては直接的に高確度で「実数」を測定できるからである。ＦＰには様々な効用があり、それを承知していてもＬＯＣを捨て切れない所以である。しかし、実績測定だけでなく見積りまで視野に入れれば、ＦＰに優位性はある。特に、見積りにおいてＬＯＣだけではプログラムに精通していないユーザーとの対話は難しい。又、計測手順が定型化されており、ＬＯＣのように見積りを行なう者の技量に依存して大きく左右されない。従って、ＦＰが趨勢として規模算定の主たる尺度であることは肯綮できるであろう。これらに鑑みて、筆者は択一的ではなく、併用するのが望ましいと考える。その場合に、各々独自に並行的に測定し、その結果を突合するのが望ましい。しかし、測定の負荷並びにコストを勘案すれば、一方でまず測定し、後述する「変換係数」を使って他方を算出する便法でも可とする。

（1-2）ソフトウェア開発の生産性

生産性は、ソフトウェア開発実績では、直接的に測定できず、実績規模と実績工数から導出する。算定式は、実績生産性＝実績規模／実績工数である。生産性は、規模をＬＯＣで測定した場合はＬＯＣ/人月の尺度で、規模をＦＰで測定した場合はＦＰ/人月の尺度で、測定する。これは、技術者１人が１ヶ月当たりに生産したＬＯＣ又はＦＰを示す。つまり、生産性はソフトウェア開発の作業効率を示すものとして、ソフトウェア成果物の規模と工数との実質的な対応を捉える指標である。なお、ＬＯＣはプログラミング言語に依存するため、複数の言語を使った場合は、総（平均）生産性と共に、言語別に生産性を測定する必要があるが、ＦＰは理念的には言語に依存しないことから、言語別の生産性を測定する必要はない。また、いずれにおいても開発全体を通した総生産性と共に、精細には開発のフェーズ＞アクティビティ＞タスク・ベースで、各粒度別に生産性を測定することが望ましい[262]。

（2）ソフトウェア資産の開示事例

ソフトウェア資産の情報開示において、資産額だけではなく、それに関連する諸情報を開示する必要性を具体的に示すために、また前項までの４つの測定項目の体系的な理解も含めて、ここではソフトウェア測定の具体的な事例を示す。これにより、ソフトウェアの資産情報の比較可能性並び

[262] 補足すると、ＬＯＣの規模では難易度等の質的な面は反映されない。また、ＦＰも複雑度は加味しているが、それとは異質な難易度等が計測に含まれていない点は同様である。そのため、難易度等を識別しない規模によると、難易度等で混濁した生産性を導出してしまうので何らかの考慮を要するのだが、難易度等の定量化はまだ確立していない。

に信頼性を検証する。なお、当該事例にある設定条件及び数値は仮想である。実例ではなく、仮想の測定事例に何程の意義があるかが問われるべきと考えるが、そもそもこうした数値はこれまでのところ、企業名を特定できない統計的な調査結果はともかく、個々の具体的な開発事例としては殆ど公表されていないのが現況である。筆者は業界関係者から多少の情報を収集しているが、企業機密の壁に阻まれて挙示することはできないので、ここでは仮想の事例とせざるを得ない。

（2-1）開示事例のケース設定

設例は、比較参照し易いように、ほぼ同様のソフトウェア（機能並びにアーキテクチャ等）を、異なる方法で開発を行なう２つのケースを設定する。なお、本事例では採用する要素技術の詳細に立ち入る必要はないので殆ど触れないが、規模である行数はプログラミング言語に依存するので、それだけは明示する。まずは、２ケースにおける共通的な条件を掲げる。第１に開発対象は、中堅企業の基幹システムとする。第２に開発形態は、新規開発とする。第３に開発期間は、１年半（18ヵ月）とする。会計年度としては２期に跨るが、単年度会計の処理は考慮せず（工事進行基準等）、通算した取り扱いとする。今回の趣旨からは差し支えないと考える。

次に、２ケースにおける個別的な条件を掲げる。ケースＡ（A社）について、①ＩＴに関するプロフィールは、全般的にＩＴ統制を推進している、ＩＴに積極的な企業である。故に、新規開発は外部委託するが、自社で企画を精細に策定した上でであり、開発完了後の保守・運用は主に自社体制で行なうものとする。②開発言語は、Ｃ言語（制御系）とＪａｖａ（それ以外）[263]とする。③開発の分担は、大手Ｓｉｅｒへの一括委託である。開発作業は全面的に委託するが、いわゆる丸投げではなく、企画は自社で策定しており、又プロジェクト管理は自社の管理者が適切に行なう。ケースＢ（B社）について、①ＩＴに関するプロフィールは、組織的なＩＴ統制は行なわず、同業他社に追随的なＩＴに消極的な企業である。故に、新規開発は自社で簡単な企画を立てた後は、委託先のソフトウェア企業に任せ、その後の保守・運用も引続き開発を委託した企業に任せる。②開発言語は、C++とする。③開発の分担は、PMO[264]を含め複数のソフトウェア企業への分割委託である。開発費を低く抑えたいので、中小規模のソフトウェア企業に委託する。但し、１社では技術者の調達やプロジェクト管理に不安があるので、複数社を採用し、且つ統合的な管理のためにPMOをも依頼するものとする。

（2-2）開示事例の具体的数値

ソフトウェアの**規模**は、まず、２ケースに共通する各々のソフトウェアの機能規模は10,000ＦＰ

[263]「制御系」はハードウェア並びにOSと連携して処理制御を行う部分であるのに対して、「それ以外」は業務的な機能部分である。

[264] Project Management Officeの略で、言わばラインの長であるプロジェクト･マネージャーを補佐ないし監視するスタッフである。近年では大規模開発の場合に設置することが多くなっている。

である。次に、各々のケースについて、ケースＡ（行数規模）は、Ｃ言語 384,000 行、Ｊａｖａ 371,000 行である。行数はプログラミング言語に依存するので、言語別に測定する必要がある。行数の合計値（755,000 行）に余り意味はない。なお、行数は、実績値としては直接的に測定可能であるが、本事例は仮想なので、ＦＰ－行数変換係数を使用した（１ＦＰに対して、Ｃ言語 128 行、Ｊａｖａ 53 行）[265]。又、制御系を 3,000ＦＰ（全体の 30%）、それ以外を 7,000ＦＰ（全体の 70%）とする。計算は、Ｃ言語：3,000（ＦＰ）×128（変換係数）＝384,000（行）、Ｊａｖａ：7,000（ＦＰ）×53（変換係数）＝371,000（行）である。ケースＢは、C++530,000 行である。計算は、ＦＰ－行数変換係数を使用して（１ＦＰに対して、C++53 行）、C++：10,000（ＦＰ）×53（変換係数）＝530,000（行）である。従って、行数規模はケースＡの 70%強で相当少ないように見えるが、あくまで機能規模は同程度である。

工数は、ケースＡは、666.7 人月である。工数は、実績値としては作業管理の中で直接的に測定可能であるが、仮想のものなので規模と生産性から導出した。計算は、10000ＦＰ÷15ＦＰ/人月≒666.7 人月[266]である。ケースＢは、1,666.7 人月である。前ケースと同様の導出方法であり、計算は、10,000ＦＰ÷6ＦＰ/人月≒1,666.7 人月である。なお、Ａ社は大手Ｓｉｅｒへの一括委託なので、既述の通り、通常は工数等の情報は得られないが、懇意としている関係により、情報提供を得られたとする。

生産性は、ケースＡは、15ＦＰ/人月である。生産性の実績値は規模と工数から導出するが、仮想のものなので、ＪＵＡＳの調査報告書[267]の数値を参考にして設定した。大手Ｓｉｅｒの管理下で効率的な作業を遂行した想定で、高めの生産性を設定した。ケースＢは、6ＦＰ/人月である。同じくＪＵＡＳの数値を参考にして、複数のソフトウェア企業に分割委託したことに鑑みて、低めの生産性を設定した。

単価は、ケースＡの規模単価は、8 万円/ＦＰとする。大手Ｓｉｅｒへの一括委託なので、その場合に一般的な規模単価としたが、比較のために、人月単価換算すると、120 万円/人月である。これは、ＪＩＳＡの調査報告書[268]の数値を参考にして設定した。大手Ｓｉｅｒの受託という想定なので、高めの単価を設定した。ＪＩＳＡの調査報告書には人月単価しか掲載されていないので、実際には先に人月単価を参考にし、逆に規模単価を導出し、120 万円/人月÷15ＦＰ/人月＝8 万円/ＦＰとした。ケースＢの人月単価は、60 万円/人月である。複数の中小規模のソフトウェア企業への委託の想定なので、こちらは IT-price data bank のインターネット・サイトが提供するデータを参考にして設定した。当該企業はソフトウェアの人材供給会社であり、纏まって公表されているものはこれがほぼ唯一であるので使用した。提供データは時系列で変動するが、採取した情報は 2010 年３月時

---

[265] Jones(2008) 邦訳 pp. 79, 104

[266] 事例を簡略にするため、工数は実態ベースとコスト・ベースを同一と見做す設定としたが、実態ベースの工数を 750 人月とするならば、生産性は 13.33ＦＰ/人月（≒10,000ＦＰ÷750 人月）となる。

[267] ＪＵＡＳ(2011)p. 133-134

[268] ＪＩＳＡ(2010)p. 9-12

点のものである[269]。なお、大手Ｓｉｅｒと異なり、人ベースの契約なので、個々の技術者によって単価が異なるのが一般的であるが、上記は平均単価とした。

開発費（委託開発費）は、ケースAは、8 億円である。計算は、8 万円/ＦＰ×10,000ＦＰ＝8 億円（規模単価ベース。なお人月単価ベースでの算定式は、120 万円/人月×666.7 人月≒8 億円）である。なお、本来であれば、自社の管理者がプロジェクト管理を行なう想定なのでその人件費等も算入すべきであるが、本事例では開発費の大半を占める委託開発費に焦点を当て、且つ計算を簡単なものにするために、それ以外は除外した。又、開発にはハードウェア費用等も掛かるが、ソフトウェア開発費の測定事例であることから、これらも除外した。ケースＢも同様である。ケースＢは、10 億2 万円である。計算は、60 万円/人月×1,666.7 人月＝10 億2 万円（人月単価ベース）である。

（2-3）比較評価

２ケースのソフトウェア開発費ひいてはソフトウェア資産額の多寡は、何を意味しているのであろうか。機能並びにアーキテクチャも類同的なシステムであるにも関わらず、金額的にほぼ２億円の差異がある。その由因は明解である。単価の高い大手Ｓｉｅｒと単価の低い中小規模のソフトウェア企業に委託し、且つ単価の高下が正比例的には生産性の高下に対応していないからである。単価がA社：Ｂ社=2：1 であるのに対して、生産性はA社：Ｂ社=2.5：１であり、Ｂ社にとっては俗に言う「安物買いの銭失い」となってしまったのである。少なくとも規模の情報が開示されれば、同等規模であるにもかかわらず、A社のソフトウェアが割安で、Ｂ社のソフトウェアが割高であることが判明する。更に、生産性や単価情報が開示されれば、上記の通り、生産性と単価の各々の差異により、両ソフトウェアの金額に差異が生じたことが明解となるのである。以下では、同一企業における異時点間での比較と企業間での比較との両方の観点から、個々の測定項目についてもう少し見ていくことにする。

まず**規模**は、A社の近年の案件毎の開発規模が平均的に数百ないし数千ＦＰ程度で、10,000ＦＰを超える開発はそれほどなかったが、過去に何回も経験してきており、10,000ＦＰの開発は一定の経験のある大規模開発と言える。それに対して、Ｂ社の近年の案件毎の開発規模はA社を下回り、且つA社より開発件数も少なく、1,000ＦＰを超える開発があまりなかった場合には、10,000ＦＰの開発は尚更突出した規模の開発となる。一般的には、10,000ＦＰの開発は確かに大規模開発ではあるが、多数行なわれている。

次に**工数**は、A社の近年の案件毎の開発工数が平均的に数十ないし数百人月程度で、1,000 人月を超える開発は稀だった場合には、666.7 人月の開発は近年では比較的多大な工数となる。Ｂ社では、100 人月を超える開発すらなかった場合には、1666.7 人月の開発は尚更未曾有の工数となる。

[269] http://www.it-price.jp/Se_Detail.asp。なお、検索条件は①金額種別(契約金額/請求金額)、②年齢別/経験年数別、③言語別(COBOL/C/C++/JAVA/その他言語/ASM/全言語)、④業種別(製造/流通/サービス/農林水産業/運輸/金融/官公庁/報道出版/情報通信/建築土木/その他業種/全業種)である。

一般的には、500 人月以上の工数の開発は確かに大きな開発であるが[270]、多数の実施例がある。なお、工数の多大な開発では派生的に、要員管理等が格段に負荷増となり生産性に影響することを指摘しておきたい。

続いて**生産性**は、A社の近年の平均的な生産性が 18ＦＰ/人月だった場合には、10,000ＦＰの大規模開発で 15ＦＰ/人月の生産性とは相当効率的であった、と評価できる。B社の近年の平均的な生産性が 12ＦＰ/人月だった場合には、今回は6ＦＰ/人月だったので半減してしまったことになり、大規模開発故の非効率性が顕著である。未曾有の経験で、且つ管理が弱体であったことが露呈した結果と言える。広範な統計ではないが、ＪＵＡＳ調査報告では、＜10 人月の場合 20.12ＦＰ/人月、＜50 人月の場合 23.79ＦＰ/人月、＜100 人月の場合 11.56ＦＰ/人月、＜500 人月の場合 11.07ＦＰ/人月、≧500 人月の場合 7.49ＦＰ/人月となっている。平均は 9.47ＦＰ/人月であるが、何分、母数が少なく各 3、21、12、16、7、計 59 件[271]で、＜50 人月の場合の方が＜10 人月の場合より高生産性といった偏差が生じている。これを参考にするならば、A社は相当な高生産性、B社はやや低生産性だった、と言える。

単価は、両社を同じ尺度で見るために、人月単価で取り扱う。A社の近年の平均的な単価が 110 万円/人月だった場合には、今回は大規模開発のため、高スキルの要員を要するとのことで 120 万円/人月の単価アップを受け入れたとする。結果的にはそれが奏功したと言える。B社の近年の平均的な単価が 70 万円/人月だった場合には、今回は大規模開発で総額が多額となるので、極力費用を抑えたいとして 60 万円/人月という単価ダウンを強く求めたとする[272]。結果的に低スキルの要員での編成となり、低生産性を招来し、逆効果となったと言える。一般的には、50 万円/人月程度から 100 数十万円/人月までバラつきが多く、A社・B社ともその枠内であるが、単価として妥当かどうかは、生産性等と密接に関連しており、単価単独では一概に言うことは難しい。

個々の測定項目の講評を踏まえて、総括的な考察としては、同等の開発規模であれば、開発費の高低は単価の高低と生産性の高低によって決まることがわかる。例え単価が高くとも、それと比例的にあるいはそれを上回って生産性が高ければ、所要工数は少なくなり開発費は低くなるからである。しかも、「同等の開発規模であれば」とわざわざ断ったのは、規模が基底的だからである。更に、要員の増大に伴い管理者も段差的に増大し、それらの調整負荷や管理負荷が著増し、生産性は顕著に低下するのである。それらの明示的な結果としての指標が規模、工数、生産性、開発費なのである。これらの情報が開示されることにより、ソフトウェア資産額の多寡の由因が判明し、同一企業

---

[270] ＪＵＡＳの調査報告では、工数区分として、＜10 人月、＜50 人月、＜100 人月、＜500 人月、≧500 人月に分類している(ＪＵＡＳ(2010)p.134)。

[271] ＪＵＡＳ(2010)P.134。何れも、開発種別：新規に関しての数値で、生産性は加重平均とされている。

[272] 今日ではコストダウンのために、オフショア開発(インド、中国等に主としてプログラム製造を発注すること)が当たり前になっているが、今回のケース設定では計算を簡単にするために、そうした想定をしなかった。

における時系列での比較並びに他企業との比較を有意に可能とするのである。ソフトウェア資産は、企業の事業活動の運営を担い、直接的ないし間接的に企業の将来キャッシュフローに影響を及ぼすものであるからである。この資産に対する比較可能性並びに信頼性の向上のために、こうした情報の開示が不可欠である。なお、ここまでほぼ比較可能性に焦点を当ててきたが、測定値を示すことが信頼性を自ずと担保するものでもあることは言うまでもない。なお、これらの情報の開示を任意的に行なうようになる場合には、もし開示しない企業があるとすれば、それはＩＴに対して消極的か、芳しくない実績なので伏せておきたいか、委託開発（丸投げ等）でプロセスへの関与をしておらず工数や生産性等の情報を有していないか、そのいずれかであることが判然としよう。外部のステークホルダーにとって、それを看取できることも、開示情報の意義と言えるのではないだろうか。

これまで論じてきたように、ソフトウェア資産は単にその金額からでは、「価値」の評価を行なうことは難しく、ソフトウェア測定に基づくソフトウェア原価計算により資産測定を行い、測定項目の追加的・補足的な情報の開示（注記）によって比較可能性並びに信頼性は向上するのである。但し、今回は財務情報に直結する開示に重点を置いたが、機能やアーキテクチャ等の定性的情報をも開示することにより、更なる情報有用性が得られるのであり、本来的には両者を併せた開示が望ましい。それに関しては、前述したように別途主題的に詳細に論じたいと考えているが、重点の置きどころと開示場所の違いだけは予め確認しておく。本章で取り上げた限定的な測定項目はあくまでも注記という形での開示を求めるのであり、それ以外に関しては非財務情報という性格の情報なので内部統制報告書等での開示で可とすると考えているのである。

本章では、ソフトウェアの新規開発における測定に限定して取り上げたが、開発に関しても未だ取り上げるべき重要なことがある。ソフトウェア再利用、オープンソース利用、仕損である。これらに関しては、後続の各章で順次取り上げる。

本論

第４章　ソフトウェア再利用

１．ソフトウェア再利用の概観

（1）ソフトウェア再利用の概観

（2）ソフトウェア再利用の目的

２．ソフトウェア会計基準における再利用の取り扱い

（1）ソフトウェア会計基準の問題点１

（2）ソフトウェア会計基準の問題点２

（3）ソフトウェア会計基準の問題点３

３．ソフトウェア再利用に適合的な会計処理

（1）外部購入ソフトウェアの再利用の会計処理

（2）自社開発ソフトウェアの再利用の会計処理

（2-1）共通的な留意事項

（2-2）ケース毎の会計処理

（2-3）ソフトウェア再構築の会計処理

４．ソフトウェア再利用に適合的な会計処理の数値事例

第４章　ソフトウェア再利用

ソフトウェアの開発並びに保守には、ソフトウェア再利用（Reuse）が密接に関係している。しかしながら、開発等における再利用をも含めた複合的な会計上の取り扱いは少々複雑となるため、前章である第３章や後続の保守を取り扱う第９章では再利用を行なわない想定で考察を進めている。しかし、実態的には、おおよそ 1970 年代以降多少なりともソフトウェア再利用は行なわれてきており、それを含めた考察が欠かせない。それ故、本章ではソフトウェア再利用を主題的に取上げ、それに係る会計処理を考察する。

具体的には、ソフトウェア開発（保守を含む、以下本章では同様とする）における再利用の特性と、会計におけるソフトウェア再利用の識別の必要性という２つの視点から考察を行なう。まず１つ目の視点については、本章で取上げるソフトウェア再利用は、開発するソフトウェアへの既存のソフトウェア資源の利用であり、意図的に広範囲に利用する場合のことである。既存のソフトウェア資源の再利用の程度や方法、それによって軽減できた開発の工数並びに費用を精確[273]に捉えることにより、ソフトウェアの開発費用の内実が明らかとなる。これを踏まえて、２つ目の視点については、保有ソフトウェアの資産形成の効率性を厳密かつ精細に捉えることを意図する。開発したソフトウェアの一部として組込まれた再利用ソフトウェアは、現行のソフトウェア会計基準では適切に会計処理されず、会計的には捕捉が困難となっている。つまり、ソフトウェアの資産価値が適切に評価されないことを問題視するのであり、既存ソフトウェア資源の利用に際しての調査・検討に係る費用に加えて、利用するソフトウェアそのものも開発対象であるソフトウェアの一部分を構成するものとして、適切な価値を資産額に反映させる必要があると考える。

現行のソフトウェア会計基準における再利用の取り扱いには、幾つかの不備がある。大枠として費用あるいは資産としての処理の振り分けをしているのみで、実態に即した再利用の会計処理を実質的には取り扱っていないに等しい。換言すれば、再利用を識別するという着眼が欠落していると言える。しかし、実態に即して捉えるには、再利用の諸ケースを設定した上で会計上の取り扱いを考える必要がある。このようにソフトウェア再利用に着目することは、ソフトウェア会計を精緻化する意義を有すると考える。なお、ソフトウェア再利用を主題的に取扱っているソフトウェア専門書は、ビジネスの成否や効率性は問題にしても、会計処理に言及しているものは筆者の知る限り皆無と言ってよい。また、会計基準に関する先行研究でも同様に再利用を取り扱っているものは見当たらない。ただ１つ、内部管理的視点からの考察を後程取上げるが、それ以外に先行研究と言えるものはない。従って、会計処理に関しては筆者が独自に考察を進める他なく、本章では再利用に係る会計処理の筆者案を提示することとしたい。

本章の構成は、次の通りである。１.では、ソフトウェア再利用を概観する。２．では、ソフトウ

[273] ソフトウェア工学において、「精度」は精細さの度合い、「確度」は確実さの度合いとされている。本章ではこの両者の意味で「精確」を使用し、「正確」とは明確に区別している。

ェア会計基準における再利用の取り扱いには幾つかの問題点があることを指摘する。それに対して、３．では、ソフトウェア再利用に適合的な会計処理案を提示する。具体的には、外部購入ソフトウェアと自社開発ソフトウェアに分けて、再利用に係る会計処理を、各々ケース毎に詳細に提示する。４．では、ケース毎に仮設的ではあるが、数値事例を挙げ、筆者案の会計処理を一層具体的に明確化する。

１．ソフトウェア再利用の概観

（1）ソフトウェア再利用の概観

会計処理に先立って、その前提となるソフトウェア再利用を概観する。ソフトウェアの再利用とは、その名の通り、ソフトウェアの開発において既存のソフトウェアを（再）利用することである。開発実務では「流用」という用語の方が多く使われているように思われるが、主題的に取り扱う文献では専ら「再利用」という用語が使われており、引用等との調和を勘案して本章では主に再利用とする。まず、ソフトウェアの普及段階の比較的早い時期から、(a) 共通サブルーチン[274]が利用されていた。例えば、日数計算、西暦和暦変換、利息計算、利率計算等のサブルーチンのように、様々なところで利用するものを共通サブルーチンとするのである。これには、ソフトウェア開発において複数個所で同様の処理を行なうために仕様を共通化し、１つのモジュール（サブルーチン）として開発する場合と、既に開発済みのサブルーチンを新たな開発で利用し、当該ソフトウェアのコンポーネント[275]とする場合があるが、本章で取上げるのは後者である。前者は再利用ではなく、共通化と言った方がよい。後者が再利用の言わば初期段階であるが、再利用はそのような局限的な利用の範囲を超えて意図的で広範囲な利用をする場合がある[276]。現象形態としては、異なるシステム環境への (b) 移植（conversion 又は migration）、パッケージ・ソフトウェア等の (c) カスタマイズした利用等、旧（現行）システムをベースにした (d) 再構築（rebuilding）等がある。移植は、基本的に機能変更は行なわず、ハードウェアやＯＳ等の異なるシステム環境に適合的な改造をして使用することである。

再利用を利用のレベル(範囲)の観点から捉えると、上原三八らは、①システムの再利用、②プログラムの再利用、③プログラム部品の再利用、④仕様の再利用、と４つのレベルで捉えている[277]。①～③は狭義のソフトウェアを直接再利用するもので、利用の範囲をシステムの全体ないしより局

---

[274] 他に、共通マクロ、共通関数、共通モジュール等の呼称があり、各々分野等によって使われているが、本章ではそれらの総称として共通サブルーチンを使う。

[275] この文脈では狭義の部品を意味しているが、 コンポーネントは広義にはプログラム(群)ないし機能(群)をも意味する。

[276] ウィル・トレイツは、そのことを「再利用の問題を、単にサブルーチンやクラスの開発と取り違えて片付けてしまわないこと」(Tracz(1995)邦訳 p. 134)と言っている。但し、再利用を包括的に捉えるには、ミニマムな再利用として、それも含めなければならない。

[277] 上原(2000)pp. 115-116

部的・限定的な範囲として捉えている。④はそれらとは異質で、狭義のソフトウェアの直接的再利用ではなく、仕様（あるいは設計書等のドキュメント）の利用に留まるため、開発の実装作業は一般的な新規開発と変わらない。上記の現象形態と対応付けると、(a) は③、(b) は①（プログラムの移植は②）、(c) は①（開発予定のシステムの一部とするパッケージならば②）、(d) は④（部分的に②または③を含む場合もあるが、一般的に再構築は異なるアーキテクチャで行なうので、プログラムは限定的な利用に留まる）[278]、に相当すると言える。

（2）ソフトウェア再利用の目的

次に、ソフトウェア再利用の目的を取上げる。目的は再利用を行なわない新規開発に比べ、工数並びに費用を軽減することである。これに関して、カーマ・マックルーアは、「ソフトウェア構成要素とソフトウェア開発経験を最大限に再利用することによって、次のような著しい効果が得られる」として、「1. ソフトウェア開発の合理化と単純化／2. ソフトウェアに対する信頼性の向上／3. ソフトウェア費用の削減」を挙げている[279]。しかし、1. と2. は再利用するソフトウェアが模範的・良質的なものであることを前提として「効果」が得られうるので、予め再利用を意図して開発した又は結果的に成功裡に開発したソフトウェアのみが該当する。従って、そうした限定的な条件下でしか成立しないことは一般化できず、3. のみが全般的に該当する「効果」と言うべきだろう。従って、開発の工数並びに費用の軽減を目的とするというのが、共通的な目的としては妥当であろう。

なお、「ソフトウェア工場」（software factory）といった構想が喧伝されたり、時々に新規に開発することが殆どなくなるかのような極端な主張がなされたりしたが、その割にそれほど大々的には再利用が広範化していないという開発実態があり、それを分析することはソフトウェア実務の研究にとっては興味深いことであるが、会計的には影響しないので、立ち入らない。

2．ソフトウェア会計基準における再利用の取り扱い

現行のソフトウェア会計基準は、再利用を如何に取り扱っているか。会計基準は、再利用を確かに一応は取り扱っているのだが、再利用の特性を丁寧に見据え、対処してはいないのである。会計基準における問題点は、大きく3つ挙げることができる。

---

[278] 再構築を技術的に大別すると、リエンジニアリング方式とスクラップ・アンド・ビルド方式がある（上原(2000) p. 9, 114）。前者は、システムの再利用とも言え、機能は変更せずに、主に技術的な作り直しであるが、実施例はそれ程多くない。費用対効果が乏しいからである。本文では後者を念頭に置いている。既存機能の相当数を継承するが、機能追加や変更をアーキテクチャの変更と併せて実現するからこそ、費用対効果が生まれるのであり、大半はこちらを採用するとみて大過ないだろう。

[279] McClure(1989)邦訳 pp. 249-250

（1）ソフトウェア会計基準の問題点１

第１の問題点は、再利用の会計処理の全ての規定に共通していることであるが、費用あるいは資産に振り分けているだけである。全て新規に開発した場合と、再利用した場合が如何に異なるか、それを会計上は如何に取り扱ったらよいか、あるいはその取り扱いは必要ないか、そういった考慮がなされていない。会計基準はカスタマイズする場合を挙示しているが、それを単に修正作業一般と同列に扱っている。つまり、再利用の特性を看過しているのであるが、実際には修正とは別に、再利用特有の作業が発生するのである。また、既存資産の利用と引換えに、その開発費用を当該開発で負担する場合は、その費用の二重計上を避けるために振替処理を行なうのが適切であるが、そうした考慮がなされていない。これらは、会計基準が見落としている欠陥と言える。このように、会計基準は実態に即した再利用を実質的には取り扱っていないに等しい。それでは、ソフトウェアを再利用した開発において、内訳的に再利用を識別することはできないのであり、換言すれば、認識する必要性を看過していると言わざるを得ない。他方、開発実務では、多少なりとも再利用が行なわれている実態があるが、現行基準に則ると適切に会計処理をなし得ないのである。

（2）ソフトウェア会計基準の問題点２

第２の問題点は､想定されている事態が相当稀ないし局部的な事態を一般化していることである。実務指針 14 項及び 38 項(２)では、購入パッケージを小幅にカスタマイズ(仕様変更)し、それをそのまま自社で使用する広義の再利用を取り上げ、同 15 項及び 39 項では、自社開発ソフトまたはパッケージを大幅にカスタマイズし、開発するソフトの部品として利用する狭義の再利用のケースを挙げている[280]。しかし、カスタマイズの程度の違いによりケース分けすることにいかなる意味があるのか、大いに疑問である[281]。更に、何故、大幅なカスタマイズの場合に限り部品としての利用を

[280] 実務指針の 14 項では「外部から購入したソフトウェアについて、(中略)自社の仕様に合わせるために行う付随的な修正作業等」の費用は、当該ソフトウェアの取得価額に含める。同 38 項(２)の補足説明では、「完成したソフトウェアを購入する場合でも、例えば、(中略)自社の仕様に合わせて画面や帳票などを修正する場合などがある。これらの作業は、自社で行う場合と外部委託する場合がある。(中略)外部から購入したパッケージソフトウェアに対して(中略)自社の仕様に合わせるための付随的な修正作業等の費用は、購入ソフトウェアを使用するために不可欠な費用であり、有形固定資産の取得に要する付随費用と同様に」処理するとされる。同 15 項では「自社で過去に制作したソフトウェアまたは市場で販売されているパッケージソフトウェアの仕様を大幅に変更して、自社のニーズに合わせた新しいソフトウェアを制作する」ための費用は、資産要件を満たせば資産計上し、それ以外は費用処理とする。これに対して同 39 項で、「既存のパッケージソフトウェアの仕様を変更して自社の要望に合わせた新しいソフトウェアを制作する場合は、完成品のソフトウェアを購入したとは考えられず、むしろパッケージソフトウェアを部品として利用していると考える方が適切である」ことから、パッケージの取得費用は、仕様変更後の新ソフトウェアの利用により資産要件を満たすときを除き、費用処理が適当とされる。

[281] この会計処理は、市場販売目的ソフトウェアにおける「著しくない改良」と「著しい改良」(実務指針９項及び 33 項)と、同列の論法である。これがソフトウェア開発の大半を研究開発とみなす

想定しているか、疑問である。何故ならば、小幅にカスタマイズしたパッケージを部品として利用する、あるいは大幅にカスタマイズをしてそのまま自社で使用する場合も十分に考えられるからである。また、購入ないし既存のパッケージの自社仕様のカスタマイズを当然のように想定しているが、そもそもパッケージに対してそれは可能なのか。会計基準が実態を踏まえているようには思えないので、ソフトウェア業界の実態を取上げる。他で既述したことでもあるので、簡潔に記し、新たなことを注記するに留める。パッケージは、第一義的にはライセンス契約により、第二義的には提供形態により、カスタマイズを許容する場合と、許容しない場合とがある。許容する場合も、カスタマイズは専ら提供元が担当し、一元的に管理することが一般的である。そのため、提供形態はオブジェクト提供とすることが多い。もう１つの提供形態であるソース提供でも、契約による縛りは可能であるが、購入先がプログラムの閲覧や変更を容易にできることから、紳士協定的とならざるを得ない。従って、安全を考えれば、オブジェクト提供とすることが多いのである。それ故、会計基準が想定している事態とは、契約的にカスタマイズを許容し、かつ提供形態がソース提供[282]の場合に限られる。その場合も、購入先が自主的にカスタマイズすると看做す規定となっているが、そうとは限らない。仕様の習熟度合いからすると、提供元へのカスタマイズの依頼が、特に初期的には大いにあり得ることである。つまり、会計基準はいずれも相当稀ないし局部的な事態を一般化していると言わざるを得ないのである。なお、購入先の自由なカスタマイズまたは別様の利用を許容する場合でも、第三者への提供を許諾することは殆どなく、あくまで購入先の自社利用の範囲内となる[283]。他方、個別１品で所有権の移転を伴う購入の場合は、ソース提供であり、第三者への提供も任意であることは言うまでもない。

（3）ソフトウェア会計基準の問題点３

第３の問題点は、再利用のケース設定に重大な欠落がある。会計基準のソフトウェア分類基準は制作目的であり、資産認識の対象のソフトウェアは販売用と自社利用用とに大別される。そのうち自社利用のみの再利用が想定されており、カスタマイズの外部委託は視野に入っているが、販売用

根幹の規定からの派生系であることは、既に第２章で取り上げ、論破している。

[282] 汎用的な仕様で十分なユーティリティ的なソフトウェアはカスタマイズをしないが、業務システムのパッケージとなると、そうはいかないことがある。その場合に、購入先の自由なカスタマイズを許容するパッケージ販売を何故行なうかというと、売切りとし、手離れをよくしたいからである。パッケージのカスタマイズを許容する場合で専ら提供元が対応・管理することは事業的には中々難しい。例えば 10 顧客(企業)に販売し、各々異なったカスタマイズを行なった場合、マスターは１つでも、それから派生した単なる複製ではないサブ・マスターが 10 存在することになるので、制度改訂等があり、且つカスタマイズ部分に関係するときは、マスターだけでなく、各サブ・マスターに変更を施さなければならない。こうした保守を継続的に採算ベースで行なっていくことは非常に難しい。その事情を知りながら、敢えて本来的なパッケージ事業を行なうか（カスタマイズ並びに保守を専一的に対応・管理する)、さもなければカスタマイズを許容する売切り制を選択するのである。

[283] ここまでの議論は営利目的のパッケージに関してであり、オープン・ソース(フリー・ソフトウェア)に関しては有償・無償に限らず、少し異なった議論が必要であるが、それは後続の第５章で主題的に取り上げているので、ここでは立ち入らない。

ソフトウェアの再利用は想定されていない。この想定を取り入れる必要性は、主に２つの点から言える。１つは、販売を目的としたパッケージ開発には特にコストダウンが強い要請としてあり、開発費用の軽減を可能とする再利用の必要性は自ずと高いのである。価格設定の基礎となる原価ということでは、余程の新奇性があるものでない限り、価格競争というビジネス要件からして、強い要請であると言える。もう１つは、技術的な要請として、シリーズ製品であるパッケージであれば、シリーズ内での下位互換性[284]が求められるのが一般的であり、それへの対応という点では旧製品の再利用が最も堅実な方法である。従って、販売用ソフトの再利用のケース設定が欠落していることは、重大な瑕疵と言わなければならない。このように、再利用に関する会計基準の規定には、幾つもの問題点があると指摘せざるを得ない。会計基準で取り扱われている再利用のケースと会計処理を一覧に整理して示すと、以下のようになる。

図表１−４−１　ソフトウェア会計基準における再利用のケースと会計処理一覧

<table>
<tr><th>制作目的</th><th>取得形態</th><th>再利用方法</th><th>開発形態</th><th>ケース設定の有無</th><th colspan="2">再利用会計処理</th><th>資産計上の仕方</th></tr>
<tr><td>市場販売目的</td><td></td><td></td><td></td><td>×</td><td colspan="2">———</td><td>———</td></tr>
<tr><td rowspan="7">自社利用</td><td rowspan="4">購入</td><td rowspan="2">独立</td><td>ノン・カスタマイズ</td><td>○
（社会的な再利用、個別企業主体ではソフトウェアの購入）</td><td colspan="2">不要</td><td>購入ソフトウェアの資産計上</td></tr>
<tr><td>カスタマイズ</td><td>△
（小幅（付随的修正）のみ）</td><td>再利用作業費用</td><td>×</td><td>———</td></tr>
<tr><td rowspan="2">部品</td><td>ノン・カスタマイズ</td><td>×</td><td colspan="2">———</td><td>———</td></tr>
<tr><td>カスタマイズ</td><td>△
（大幅のみ）</td><td>再利用作業費用</td><td>×</td><td>———</td></tr>
<tr><td rowspan="3">自社開発</td><td rowspan="3">部品</td><td>ノン・カスタマイズ</td><td>×</td><td colspan="2">———</td><td>———</td></tr>
<tr><td>カスタマイズ</td><td>△
（大幅のみ）</td><td>再利用作業費用</td><td>×</td><td>———</td></tr>
<tr><td>再構築</td><td>△
（大幅なカスタマイズがこれを包含していれば）</td><td>再利用作業費用</td><td>×</td><td>———</td></tr>
<tr><td colspan="5"></td><td>カスタマイズ費用</td><td>○</td><td>資産要件を充足すれば資産計上</td></tr>
</table>

（○：規定あり、△：一部規定あり、×：規定なし）

（出典・筆者作成）

３．ソフトウェア再利用に適合的な会計処理

ここからは、ソフトウェア再利用に係る会計処理の筆者案を提示していくが、まず現行の会計基

[284] ソフトウェアが相互に利用可能である互換性のうち、旧製品(の仕様ないしデータ等)をそのまま継続的に利用できることである。

準との大きな相違に触れておく。第１に、再利用の対象であるソフトウェアとして、市場販売目的ソフトウェアにおける再利用をも漏れなく捉えることである。そのため、外部購入ソフトウェアと自社開発ソフトウェアとに大別し、自社開発は更に狭義の再利用と広義の再利用の一種である再構築に分けて取上げる。第２に、再利用の方法に着目することである。購入または自社開発のいずれでも、そのまま独立的に使用するか、あるいは開発で部品として利用するかは種々のケースが想定でき得る。その各々のケースに即した会計処理を行なうのでなければならない。

（1）外部購入ソフトウェアの再利用の会計処理

まず、外部購入ソフトウェアの再利用を取上げる。想定できるのは全３ケースである。（a）ノン・カスタマイズで購入する、（b）自社用にカスタマイズしてもらい購入する、（c）購入して自社でカスタマイズする、というケースである。具体的な会計処理と併せて、それらをまとめた図表に沿って説明する。会計処理の全体的な方針は、外部購入ソフトに対するカスタマイズ費用[285]は、カスタマイズを必要とする当該企業にとっての資産価値を増大させるものとして、ソフトウェア資産の取得原価に含めるものとする。しかも、会計基準のようにカスタマイズの程度により取り扱いを区分することはせず、資産要件[286]を充足すれば資産認識し、資産の取得原価を構成するとすることが妥当であると考える。

図表１−４−2　外部購入ソフトウェア再利用会計処理一覧

No.	購入形態	再利用会計処理	資産計上の対象
a	ノン･カスタマイズ	不要	パッケージ・ソフト購入価格
b	カスタマイズ済み購入	不要	外部委託費 パッケージ・ソフト購入価格 （上記はセットでの購入価格の場合もある）
c	購入後、自社カスタマイズ	再利用作業費用 カスタマイズ費用	再利用作業・カスタマイズ費用 ソフト購入価格

（出典：筆者作成）

ケース（a）はソフトウェアを新規開発する替わりにほぼ同等の機能を有するパッケージを購入し、カスタマイズはしない場合である。これは購入先が購入価格をもってソフトウェアを資産計上するが、再利用の会計処理は必要ない。

ケース（b）は（a）と同じパッケージであるが、そのままでは当該企業が求める機能要件等を充足しないので、カスタマイズを施してもらった上で購入する場合である。カスタマイズ及び再利用作業は受託側が行なうので、委託側の再利用の会計処理は不要となる。これらの費用に利益を上

[285] 本章で用いる「費用」という用語は全て、発生現象を意味し、cost と同義である。
[286] 会計基準におけるソフトウェア開発を研究開発とみなす規定との関連で、資産要件に関しても問題視しているが、本論第２章で既に取り上げているので、ここでは議論を混乱させないために現行会計基準の資産要件の規定を一応踏襲した議論を行なっている。

乗せした売価が委託企業にとって外部委託費用であり、同じくパッケージの売価がパッケージの購入価格となり、これらの合計額をソフトウェア資産に計上する。なお、カスタマイズ及び再利用の部分は、ケース（ｃ）と区別するために外部委託費として捉えるが、初期段階では購入パッケージに対する追加購入費としても差し支えない。また、追加購入費ではなく、それらの費用を含めてセットでの購入となる場合もあり、その場合はパッケージ購入価格のみとなる。

ケース（ｃ）は（ａ）（ｂ）と類似のパッケージであるが、違いは購入先の当該企業が自由にカスタマイズできることである。但し、このケースが実際には会計基準の想定に反して、かなり少ないことは弁えておかなければならない。この場合、費用総額はカスタマイズ及び再利用作業の費用に利益を含んだ売価としての上乗せがないので、（ｂ）より安価になることが考えられるが、（ｃ）を選択する主たるメリットはそこにはなく、パッケージ仕様に習熟し、自社で保守していけるようにすることである。これを重視する場合は、初期的には高価であっても自社でカスタマイズするであろう。費用重視であれば、売価としての上乗せと、仕様に習熟していく学習曲線との兼ね合いが（ｂ）とするか、（ｃ）とするかの判断基準となろう。再利用の会計処理は、カスタマイズ費用及び再利用作業費用となる。これらに、パッケージの購入価格を加えた金額を資産計上する。ここで、「カスタマイズ費用」とは購入ソフトの直接的な仕様変更部分に係る費用を意味し、実装作業は開発と同様である。「再利用作業費用」とは、購入ソフトの再利用に係る作業の費用を意味する。それは、外部調達したコンポーネントの仕様に習熟していないため、詳細な仕様を理解するまでの調査・分析と、無修正部分とカスタマイズ部分との整合性や影響の有無に問題がないことの確認をする作業であり[287]、一般的な開発とは明らかに異なる再利用に特有のものである。

ところで、ケース毎には提供形態がオブジェクトあるいはソースでの提供の場合があり、更には汎用的で不特定ユーザを対象とするパッケージと汎用的ではなく特定ユーザを対象とする専用性の強い場合とに分けることが可能であるが、会計処理の内容及び取得費用並びに開発費用の測定は全く同様となるので、細かくケース分けをしていない。また、ケース（ａ）（ｂ）は、広義には再利用であり、個別企業の枠を越えた社会的な再利用と言えるが、個別企業主体では単なるソフトウェアの購入である（カスタマイズを含めた購入として）。ケースとして抜けがないようにするために取り上げたが、事情を承知していれば、省略しても差し支えない。

（２）自社開発ソフトウェアの再利用の会計処理

次に、自社開発ソフトウェアの再利用を取上げる。社内のソフトウェア資源を再利用する場合、その取り扱いは幾通りもあり得る。ソフトウェア資源の再利用レベルやプロジェクトの編成形態、管理方式やカスタマイズの作業担当の違い等により取り扱いが異なり、更にその費用の取り扱いを

[287] 保守と類似しているが、それとの違いは、自らが開発したわけではないソフトウェアの全貌を把握し、他のソフトウェアとの親和性並びに将来性（保守性、拡張性）の良否の確認等、保守ならば既決事項を一から点検していかなければならないのである。

ケース分けし、整理して捉えることが可能である。しかし、会計上は当該プロジェクト(以下、「当該ＰＪ」と略記)の会計処理として、再利用に係る費用負担に着目したケース設定で必要十分である。従って、ケース設定に関して、一定の合理性を有するのは全７ないし８ケースである。なお、機械的な組合せとしてはあり得るが、合理性を欠くものは割愛する（その理由は後述する)。

図表１−４−３　自社開発ソフトウェア再利用会計処理一覧

No.	再利用態様	既存資産 開発費用負担	カスタマイズ 費用負担	再利用会計処理	資産計上の対象
d	ノン・ カスタマイズ	負担なし		再利用作業費用	再利用作業費用 新規開発費用
e		一部負担		再利用作業費用 既存資産開発費用 振替	再利用作業費用 新規開発費用 既存資産振替額
f	カスタマイズ	負担なし	負担なし	再利用作業費用	再利用作業費用 新規開発費用
g			一部負担	再利用作業費用 カスタマイズ費用	再利用作業費用 カスタマイズ費用(一部) 新規開発費用
h			全て負担	再利用作業費用 カスタマイズ費用	再利用作業費用 カスタマイズ費用(全部) 新規開発費用
i		一部負担	一部負担	再利用作業費用 カスタマイズ費用 既存資産開発費用 振替	再利用作業費用 カスタマイズ費用(一部) 新規開発費用 既存資産振替額
j			全て負担	再利用作業費用 カスタマイズ費用 既存資産開発費用 振替	再利用作業費用 カスタマイズ費用(全部) 新規開発費用 既存資産振替額
k	再構築	負担なし	全て負担	機能仕様の承継費用 再利用作業費用 (旧システム除却 除却費用)	機能仕様の承継費用 再利用作業費用 新規機能及び方式の全工程 における新規開発費用

（出典：筆者作成）

（２−1）共通的な留意事項

個々のケースを取り上げるのに先立ち、ケース設定の考慮点を挙げ、それを表現した図表の説明を兼ねることにする。

第１に、開発に再利用するソフトウェア資源の利用態様を「ノン･カスタマイズ」と「カスタマイズ」に大別したが、それらは再利用に関わる性格及び費用の取り扱いが異なるためである。また、購入ソフトと統一的な取り扱いとする意味もある。再利用する際にそのまま利用するか、仕様変更を行なった上で利用するかの区別である。

第２に、既存資産で開発に再利用するために提供されるコンポーネントが資産要件を充足し、予め資産計上されていることを前提とする。その既存資産の開発費用を、再利用する際に当該ＰＪが負担することが考えられるが、その費用負担の有無を「負担なし」「一部負担」に区別している。なお、再利用に際し、その費用の全てを当該ＰＪが負担することは考え難いので（共通化は再利用と

異なることは既述した)、割愛した。

第３に、再利用態様がカスタマイズの場合には、既存資源の仕様変更に係る費用が発生するので、これを当該ＰＪが負担するか否かにより、「負担なし」「一部負担」「全て負担」に区別している。なお、既存資産開発費用を一部負担しながら、まさに当該ＰＪのためのカスタマイズ費用を負担しないケースは合理性を欠いているので割愛した。

第４に「再利用会計処理」は、再利用に係る会計処理の対象となる費用を挙げている。「再利用作業費用」は既存資源の再利用に特有に係る調査等の作業費用を意味し、「カスタマイズ費用」は既存資源の直接的な仕様変更に係る費用を意味する。また、「既存資産開発費用 振替」は、既存資産の開発費用を負担するケース（ｅ）（ｉ）（ｊ）で、既存資産を開発及び管理するグループまたはプロジェクト(以下、「Ｇ又ＰＪ」と略記)から負担費用の振替処理を行なうことを意味している。

第５に「資産計上の対象」は、再利用の会計処理と併せて、当該開発対象のソフトウェアが資産要件を充足することを前提として、資産計上の対象となる費用を挙げている。

第６に費用負担に関して「一部負担」とし、負担割合の数値等を提示していないが、様々あり得るので、ここではそれ以上には踏み込まない。基本的には企業における決め事であり、当為的基準を一義的に定めるのは適切ではない。これに関しては、後続の４．(節) において会計処理の数値事例を掲示する際に、具体的な数値を仮設する取り扱いとする。

第７に、ソフトウェア資源の種別には「共通コンポーネント」と「個別コンポーネント」とがあり得る。前者は再利用に供する目的での開発または管理をしている資源であり、後者は再利用を目的とした開発及び管理は行なわないが、今回の当該ＰＪの開発に再利用する資源である。開発実務ないし開発後の管理上は重要となるが、いずれも再利用時点での会計上の取り扱いには大きく影響しないので、ケース分けに反映していない。ケース設定の考慮点は以上である。

（2-2）ケース毎の会計処理

続いて、具体的な会計処理は個々のケース毎に詳しく取り上げるが、重複を避け、主にケース毎の違いを中心に説明する。前提条件は、開発の一部として既存資源を再利用することであり、当該既存資源は資産要件を充足し既に資産計上されていることである。

ケース（ｄ）は、コンポーネントは無償供与でかつ無修正で再利用する場合であり、再利用の作業費用のみが発生する。コンポーネントの無修正の再利用には、プログラム製造等の作業は一切不要であるが、コンポーネントの仕様の調査・分析と利用に問題がないことの確認テストが必要となる。但し、パッケージ等の総合的な利用ではなく、ごく局限的な利用なので、新規開発作業に比して、比較的少ない作業を遂行するに留まる。それが再利用作業費用である。この費用は、カスタマイズや既存資産の費用負担の有無に関わらず、再利用する限り必ず発生し、以降の全ケースに共通している。そして、これに新規開発費用を加えた金額をソフトウェア資産として計上する。全てを新規開発する場合に比して、コンポーネントの開発費用分が軽減できることになる。なお、既に資

産計上されているコンポーネントに対する会計処理は、不要である。

ケース（ｅ）は、コンポーネントは無修正の再利用であるが、再利用と引換えにコンポーネントの開発費を一部負担する有償供与の場合である。よって、前ケースとの違いは、再利用作業費用に既存資産開発費用負担額が加わり、多少費用負担が増すことである。既存資産開発費用は、既存コンポーネントを部品として利用する開発であり、言わば製造原価に占める直接材料費に相当する。それに対して、直接労務費に相当するのが再利用作業費用である。ここで、既存資産開発費用に特異な処理が発生する。既存コンポーネントの開発費の一部を当該ＰＪに振替えると同時に、当該振替額だけ既計上のコンポーネント資産から減額する。既存資産との二重計上を避けるため振替を行なうのである。既存資産が既に減価償却されている場合は、未償却残高からの振替額の減額となる。未償却残高が振替額未満の場合は、減額可能額は未償却残高までとして同額を減額し、当該未償却残高を当該ＰＪに振替える。償却済みであれば、当初の利用価値を超えたプレミアム価値と捉え、既存資産及び当該ＰＪへの振替処理は不要となる。これらの振替額と全くの未償却の振替額とでは差が生ずるが、その差額分は、再利用して開発したソフトウェアの潜在的な価値が高いものと捉えることができるだろう。但し、既存資産開発費用を負担する場合にも振替を行なわないことが考えられるが、その場合は費用を負担しないと見做すことになる。しかし、より実態を反映させるためには、振替を行なうことが妥当であると考える。この振替処理は、既存資産開発費用を負担する以降の該当ケースに共通している。そして、新規開発費用にこれらを含めた開発費用総額を資産計上する。

なお、既存資産開発費用の振替を、井手吉成佳は次のように取り扱っている[288]。予め断っておくが、ここでは引用との兼合いで「流用」という用語を用いるが、再利用と同義である。受注ソフトウェアの価格決定の考察の中で、ソフトＡをソフトＢに部分的に流用する場合、例えば、Ａの開発原価は固有部分ａ（40 万円）とＢへの流用部分（60 万円）、Ｂの開発原価は固有部分ｂ（70 万円）とＡからの流用部分（60 万円）から構成されるとした上で、「無形固定資産として計上されるソフトウェア開発の支出額はソフトウェアＡの100万円とソフトウェアＢの固有部分ｂの70万円との合計である 170 万円である」としている。実際の支出額としては間違っていないが、この考察の問題点を２つの観点から指摘しよう。１つは、開発実態を数値上、忠実に表現しているかという点である。資産計上されるＢには、開発において流用した実態が反映されていないことに問題がある。Ｂの開発では、全てを新規に開発することをせずに、Ａの一部分を流用したのであるから、全てを新規に開発する場合に比して流用部分の開発費用を軽減できたことを反映させなければならない。開発費用の軽減が、まさに流用する目的であるからである。この結果が数値上も反映されなければ、開発実態に忠実であるとは言えない。他方、Ａに関しても、資源を利用されたということが反映されておらず不適切である。このように、Ａ並びにＢのソフトウェアに関わる流用という経済的実態

[288] 井手吉(2010)p. 181

を、適切に反映し得る情報とする必要がある。もう１つは、ソフトウェアの資産情報の質が良好であるかという点である。Ｂの資産額に流用部分の原価を全く反映しないならば、流用部分の資産価値が評価されないことになり、ソフトウェア資産の情報有用性は乏しいと言わざるを得ない。何故ならば、資産Ｂに流用部分に係る潜在的な価値が存在していることを、情報利用者は認識しようがないからである。仮に、Ｂの資産額に流用部分の原価を加えないならば、潜在的価値を知り得る他の方法により情報開示しなければ、Ｂの資産情報には不備があることになる。従って、これらの問題点を改善すべく、Ａの流用部分の開発原価を何らかの負担割合に応じてＢに負担させることが必要であり、振替処理が適切であることを主張する。なお、流用特有の作業並びにカスタマイズのケースを考慮していないことも指摘しておく。

ケース（ｆ）は、コンポーネントは無償供与であるが、仕様変更して再利用する場合である。但し、カスタマイズ費用は当該ＰＪでは一切負担しない。従って、当該ＰＪの再利用の会計処理は、ケース（ｄ）と同様である。なお、発生したカスタマイズ費用は、既存資産を管理するＧ又はＰＪで既計上コンポーネントに加算処理を行なう。

ケース（ｇ）は、類似的であるが、ケース（ｆ）との違いは、カスタマイズ費用の一部を負担することである。従って、負担割合に応じたカスタマイズ費用の処理と、この費用の資産計上が加わる。他方、カスタマイズ費用から当該ＰＪの負担額を除いた残額は、既存資産を管理するＧ又はＰＪで既計上コンポーネントに加算処理を行なう。

ケース（ｈ）も、類似的であるが、違いはカスタマイズ費用の全てを当該ＰＪで負担することであり、金額が違ってくる。従って、既存資産を管理するＧ又はＰＪでの処理は不要となる。それ以外は同様である。

ケース（ｉ）は、コンポーネントの仕様変更した再利用であるが、ケース（ｆ）～（ｈ）との違いは、既存資産の開発費用の一部を負担する有償供与の場合ということである。従って、負担割合に応じた既存資産開発費用振替額の振替処理と、この費用の資産計上を追加的に行なう。他方、既存資産を管理するＧ又はＰＪでは、既計上コンポーネントに対して、既存資産開発費用の振替額を減額し、併せてカスタマイズ費用の当該ＰＪ負担残額を追加計上する。当該ＰＪはカスタマイズ費用の一部を負担する。

ケース（ｊ）は、類似的であるが、違いはカスタマイズ費用の全てを当該ＰＪが負担することであり、金額が違ってくる。従って、既存資産を管理するＧ又はＰＪでのカスタマイズ費用の処理は、不要となる。それ以外は同様である。

ここで纏めて補足的に、既存資産開発費用を一部負担する場合の資産の取り扱いを考えてみる。負担額を既存資産額の一定割合とする場合に、更に減価償却が進んでいる場合には、当初資産額の一定割合とするか、未償却残高の一定割合とするかが精細には問題となる。それにより実際の金額は異なるからである。しかし、いずれにしても決定的に優位とは言えない。当該ＰＪで開発するソフトウェアに対する既存資産の再利用の貢献度合いは、容易ではないが、測定可能である。しかし、

既存資産自体にとっての利用価値の測定は相当難しい。何故ならば、まず個別コンポーネントは再利用を予め企図していないので想定外の利用であり、ユーザを含めた他ＰＪで既に十分に利用価値を発揮しているとすれば、それを超えたプレミアムな利用価値と捉えるべきであり、その場合には無償供与でも差し支えない。次に、共通コンポーネントは再利用を予め企図しており、再利用頻度等を企画・開発計画等で明確にしていれば、それに則った利用価値の測定が可能であるが、そうでない場合には、適当な決めをするしかないからである。故に、負担額の決定に際して規準となる負担割合ないし金額は、いずれも便宜的であり、特段の優位性はない。従って、一貫して採用し簡便であれば、いずれであっても差し支えない。但し、減価償却との関連を考慮に入れれば、当該再利用時点での未償却残高を規準とした負担額の決定が、会計上は比較的優位であるように思われる。既述の通り、減価償却手続を所与とした負担額の算出と振替処理を行なうので、これらと連携させるという点では一貫性があるからである。

また、関連して、仕様変更費用を一部負担する場合の取り扱いにも、類似の問題が生じる。共通コンポーネントに関して、仕様変更が当該ＰＪにとって不可欠であり、利用価値があるのは明白であるが、将来的な再利用価値やその頻度等を見積ることは中々難しい。従って、一部負担を具体的に如何にするかは、やはり便宜的たらざるを得ないであろう。

（2-3）ソフトウェア再構築の会計処理

ソフトウェア再利用のケースの１つではあるが、ケース（ｄ）～（ｊ）までとは大きく性格が異なるので、項を分けて、自社開発ソフトウェアの再構築を取上げる。第１に、再構築は広義の再利用の１種であるが、これ以外の再利用とは目的が異なる。再構築の主目的は、既存機能の承継と機能並びに技術の刷新という両面的なものである。よって、前述の工数並びに費用の軽減という狭義の再利用の主目的は、付随的ないし副次的なものに留まる。単なる継承ならば保守を継続するか、アーキテクチャに問題があるだけならばリエンジニアリング方式の再構築をすればよい。それを敢えて、多額の費用を投じ、スクラップ・アンド・ビルド方式で再構築を行なうのは、相当数の機能を承継するが、刷新（追加を含む）したい要件も少なからずあるからである。費用だけを忖度すれば、現行システムに拘らず、全く新規に開発したほうが安上がりなことすらあり得る。それにも関わらず、長年蓄積してきた業務ノウハウ等を承継したいからこそ、再構築とするのである[289]。技術論に終始しがちな再利用の諸書が揃って看過している重要な事項である。

第２に、再構築は再利用の範囲がどれ程であろうと、開発完了に伴い、旧システムを廃棄することである。これ以外の再利用には、再利用資源と再利用した新ソフトウェアの間にそのような連動的な関係はない。それ故、これに特有の会計上の取り扱いが必要となる。それは、必ず発生する廃

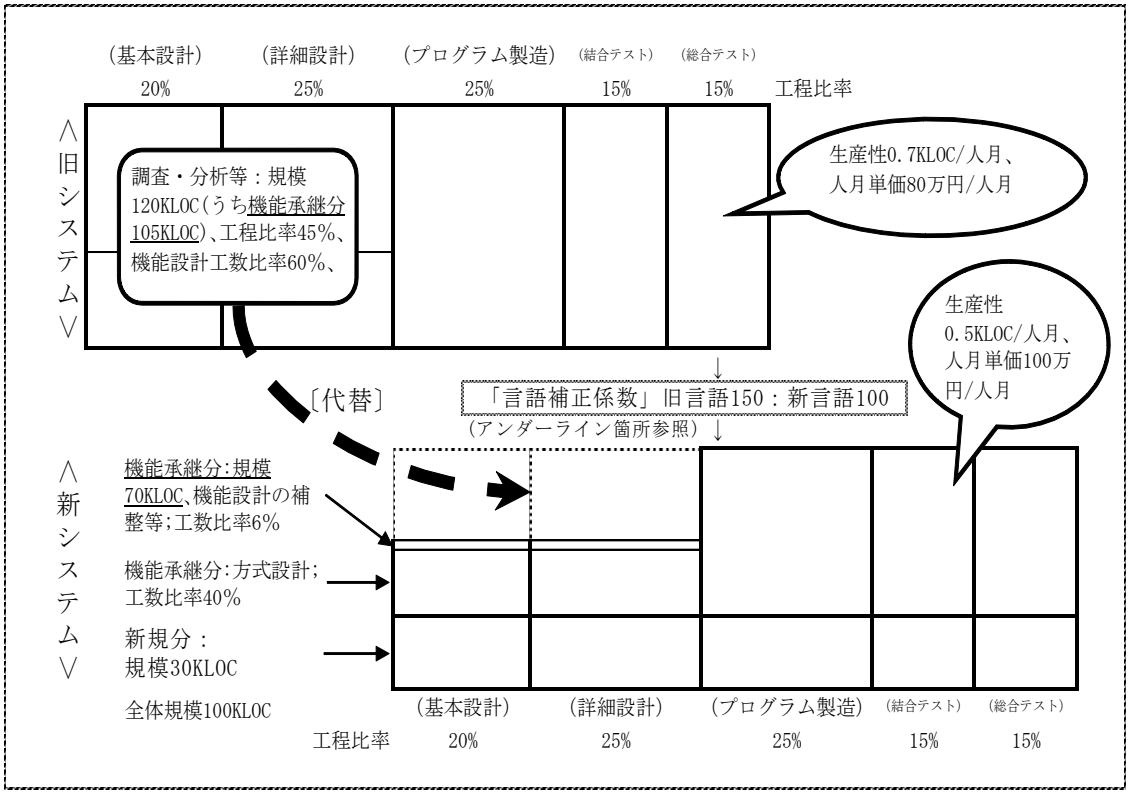
[289] ここには、「社内政治」の問題がしばしば介入する。例えば、守旧派と刷新派の抗争といった、ソフトウェアの仕様や方式に託けて代理戦争を行なう様相であるが、主題からは逸脱するので立ち入らない。

棄に伴う一連の定型的な処理であるので、それ以外の再利用のようなケース分けは不要となる。

第３に、再構築で独立的なＰＪを編成するため、当該ＰＪ自らが担当しないことはあり得ないので、複数のケースを設定する必要はない。

具体的な会計処理として挙げているのはケース（ｋ）で、スクラップ・アンド・ビルド方式により再構築する場合である。既存の機能要件の大半を承継するので、機能設計に対する追加・変更はそれ程多くないのに対し、技術要件（方式設計）はその殆どを新たに設計し直すので変更は広範囲に亘る。よって、再利用の会計処理は、機能仕様の承継費用及びそれの再利用作業費用である。前述のように、製造原価に占める直接材料費が機能仕様の承継費用であり、直接労務費が再利用作業費用に相当する。これらに加えて、新規機能及び方式の全工程における開発費用がシステム全体の費用を構成し、併せて資産計上する。また、旧システムは、新システムの稼働と同時もしくは一定期間並行運用しても遠からず廃棄するので、未償却残高のある資産はそれを全額「除却」し、償却済みであれば不要となる。そして、廃棄に係る一時費用を「除却」に係る費用として処理する。この処理は一般的にはシステム運用部門で行ない、当該ＰＪで行なう処理ではないが、再構築に特有の処理として挙示しておく。詳細な取り扱いは、本論第 10 章で主題的に取り上げる。これで、再構築に係る会計処理の概容を示せたと思う。

図表１−４−４　ソフトウェア再構築の模式図



（出典：筆者作成）

それでもまだ、再構築のやや複雑な態様を十分弁えて会計処理を行なうには、概容だけでは覚束ないと思えるので、他のケースとは異なり、数値事例の挙示に先立ち、数値を示しながら、且つ模

式的な図を掲示して、補足的な説明を加えることにする。

再構築ソフトウェアの機能の 70%は旧システムから承継し、30%を新規とする。新規分は、規模 30KLOC、工数 60 人月（＝30KLOC÷0.5KLOC/人月）、開発費 6,000 万円（＝100 万円/人月×60 人月）とする。機能承継分は、新システムの規模で 70KLOC とする。機能承継分の計算は、次のようになる。まず、基本設計並びに詳細設計における、機能の設計と技術（方式）の設計のアクティビティ比率（工数比率）は 60%と 40%とする[290]。次に、新旧システムのアーキテクチャーが異なるので、規模換算が必要となるが、簡便な計算を行なうため「言語補正係数」を使用し、旧言語 150：新言語 100 とする[291]。また、旧システム（旧言語）の生産性は 0.7KLOC/人月、人月単価は 80 万円/人月とする。更に、旧システムの全体規模は 120KLOC、承継機能分の規模は 105KLOC（＝70KLOC×1.5）、非承継機能分の規模 15KLOC とする。但し、調査・分析を行ない、承継機能分を抽出するためには、全体規模に対する調査・分析を行なうものとする。新システムの基本設計並びに詳細設計における機能設計の大半に相当する、旧システムの調査・分析等の工数並びに費用は次の通りとする。工数は、旧システムの生産性、設計の工程比率、機能設計の工数比率、調査・分析の遂行比率を用いると、18.5 人月（≒120KLOC÷0.7KLOC/人月×（20%+25%）×60%×40%）となる。遂行比率は、影響関係等の調査・分析を遥かに超えて、承継するための作業なので 40%と設定した。費用は、旧システムの人月単価を用いると 1,480 万円（＝80 万円/人月×18.5 人月）となる。これは、旧システムの設計要件に適合的な技術者をアサインするという想定であり、旧システム開発時の費用を遡及的に計算しようとしているわけではない。これで新システムの機能設計は、ほぼ遂行することになるが、多少の補整等を要するとし、それを新システムの機能設計の 10%とすると、工数 3.78 人月（＝70KLOC÷0.5KLOC/人月×（20%＋25%）×60%×10%）、費用 378 万円（＝100 万円/人月×3.78 人月）となる。

新システムの基本設計並びに詳細設計における承継機能分の方式設計は、工数 25.2 人月（＝70KLOC÷0.5KLOC/人月×（20%＋25%）×40%）、費用 2,520 万円（＝100 万円/人月×25.2 人月）となる。プログラム製造以降の工程は再利用ではなく、新規の開発となるので、工数 77 人月（＝70KLOC÷0.5KLOC/人月×（25%＋15%＋15%））、費用 7,700 万円（＝100 万円/人月×77 人月）となる。

機能承継分の全体的な工数は 124.48 人月（＝18.5 人月＋3.78 人月＋25.2 人月＋77 人月）、費用は 1 億 2,078 万円（＝1480 万円＋378 万円＋2520 万円＋7700 万円）となる。そのうち、再利用に係

---

290 参照できる統計があるわけではなく、筆者の仮定である。新奇技術を採用する場合、あるいは難易度の高い技術を採用する場合等は、方式設計の比率は遥かに高く設定する必要があろう。

291 アーキテクチャーの違いは、言うまでもなくプログラミング言語だけではないが、ソフトウェア資源の大半を占めるプログラムの言語が大きな要素であることは確かであり、それに着目することは１つの簡便な方法と言える。「言語補正係数」は、異なる言語で同等の機能を実装する場合の規模比率を指す。提示した比率は筆者の仮定である。これまでの計算例も、採用する言語を特定してはいないので、この計算例のみ具体的な言語とするのは不釣合いなので仮定とする。計算方法さえ的確に示せれば、数値等を変えて再計算することは容易であろう。

る工数 18.5 人月、費用 1,480 万円である。新規分を合わせると、開発総工数 184.48 人月（＝60 人月＋124.48 人月）、費用総額 1 億 8,078 万円（＝6000 万円＋1 億 2078 万円）となる。もし、全体を全くの新規開発とする場合は、規模・生産性からすると、開発総工数 200.00 人月（＝100KLOC÷0.5KLOC/人月）、費用総額 2 億円（＝100 万円/人月×200.00 人月）となるから、2,000 万円弱開発費を軽減できたことになる。但し、再構築は前述したように、費用軽減が主目的ではなく、ノウハウとしての機能の承継に重要な意義があるので、副次的に費用軽減効果も得られると言うべきであろう。幾つもの仮設を行なっているので、それらの仮設を別様に行なえば、各比率等が異なってくるので、、計算結果が異なることは言うまでもないが、再構築の測定の手順は十分に示せたと考える。

ソフトウェア再利用に適合的な会計処理を、現行会計基準における不備に対する指摘を踏まえて、諸ケースを詳論することにより十分に説明し得たと考える。それが財務報告にとって有する意義を纏めておきたい。ソフトウェア資産の取得原価は、その金額の多寡だけでは、当該ソフトウェアが如何なるものか、実のところ明瞭ではない。ソフトウェア開発の測定に関する考察は前章で行なったが、その際にも言及したように、開示情報としてソフトウェア測定項目の開示の必要性を痛感している。それに対する本章の意義は、再利用という開発方法の内容と併せて、その精細な会計上の取り扱いを示したことにある。僅かに振替処理等により複雑になるとは言えるが、実態に即した会計処理を可能にするものであるから、それは軽微な負担に過ぎないと考える[292]。

４．ソフトウェア再利用に適合的な会計処理の数値事例

続いて、３．で取り上げた諸ケースに関して、仮設的な数値事例を提示し、どのような会計処理を行なうかをより具体化する。これによって、ソフトウェア再利用に係る会計処理を包括的に提示示することになるであろう。

最初に比較のために、再利用を全く行なわない、全てを自作する新規開発の数値事例を挙げておく。開発規模が 100KLOC[293]、生産性が 0.5KLOC/人月とすると、工数は 200 人月（＝100KLOC÷0.5KLOC/人月）となる。人月単価（平均）を 100 万円/人月とすると、開発費は 2 億円（＝100 万円/人月×200 人月）となる。資産要件を充足しているので、全額を資産計上することにする。なお、測定（計算）並びに比較を簡便且つ明解にするため、人件費以外のその他諸費用は掛からなかったと見做し（共通的に設置されている開発環境で開発を行なったとすれば、それほど不自然なことではない）、以下全て同様とする。

---

[292] なお、この考察は細則主義に立脚したものとみなされるかもしれないが、ソフトウェア再利用の特性を適切に捉えるための会計処理の考察であり、原則主義・細則主義のいずれの立場に立脚しようとも、こうした考察を踏まえる必要があると考える。

[293] LOC を尺度とする規模測定では、ある程度の規模の場合には K(キロ)単位(1KLOC=1,000LOC)とするのが一般的である、更に規模が大きい場合には M(メガ)単位(1MLOC=1,000,000LOC)とする。

図表 1-4-5　ソフトウェア再利用会計数値事例

No.	再利用態様	既存資産開発費用負担	カスタマイズ費用負担	規模(KLOC)			費用(万円)							現行基準比率(%)
							筆者案						現行基準	
				新規	再利用	合計	新規開発費用	再利用費用				合計		
								再利用作業費用	カスタマイズ費用	振替費用	計			
a	ノン・カスタマイズ	― ― ―	― ― ―		100	100	2,400				0	2,400	2,400	100.0
b	カスタマイズ	― ― ―	全て負担	40	60	100	2,400		6,000		6,000	8,400	8,400	100.0
c	自社カスタマイズ	― ― ―		40	60	100	2,400	900	8,000		8,900	11,300	11,300	100.0
d	ノン・カスタマイズ	負担なし	― ― ―	90	10	100	18,000	90			90	18,090	18,090	100.0
e		一部負担		90	10	100	18,000	90		400	490	18,490	18,090	102.2
f	カスタマイズ	負担なし	負担なし	70	30	100	14,000	270			270	14,270	14,270	100.0
g			一部負担	70	30	100	14,000	270	1,500		1,770	15,770	14,270	110.5
h			全て負担	70	30	100	14,000	270	3,000		3,270	17,270	17,270	100.0
i		一部負担	一部負担	70	30	100	14,000	270	1,500	1,200	2,970	16,970	14,270	118.9
j			全て負担	70	30	100	14,000	270	3,000	1,200	4,470	18,470	17,270	106.9
k	再構築	負担なし	全て負担	30	70	100	16,598	1,480			1,480	18,078	18,078	100.0

（出典：筆者作成）

ケース（a）は、規模が 100KLOC、パッケージの購入価格は 2,400 万円とする（新規開発費用欄を流用しているが、購入価格である）。

ケース（b）は、規模並びに購入価格はケース（a）と同じだが、カスタマイズを販売企業に委託する場合である。パッケージの 60%（60KLOC）はそのまま（再）利用し、40%（40KLOC）をカスタマイズ（機能追加）したものとする（規模・新規欄を流用しているが、カスタマイズの規模である）。生産性は、パッケージ仕様に習熟しているので、0.8KLOC/人月とする。工数は 50 人月（=40KLOC÷0.8KLOC/人月）となる。単価は開発元企業の優位性から、やや高めの 120 万円/人月とする。カスタマイズ費用（外部委託費）は 6,000 万円（=120 万円/人月×50 人月）となる。従って、パッケージの購入価格を合算した購入価格総額は 8,400 万円（=2,400 万円＋6,000 万円）となる。

ケース（c）は、規模並びに購入価格はケース（a）（b）と同じだが、カスタマイズを自社で行なう場合である（販売企業とは別のソフトウェア企業への委託を含む）。パッケージの（再）利用率並びにカスタマイズ（機能追加）率は（b）と同じとする（規模・新規欄を流用しているが、カスタマイズの規模である）。生産性は、パッケージ仕様に習熟していないで、自社平均の 0.5KLOC/人月とする。工数は 80 人月（=40KLOC÷0.5KLOC/人月）となる。単価は販売企業の売価より低い自社平均の 100 万円/人月とする。カスタマイズ費用（外部委託費の場合もある）は 8,000 万円（=100 万円/人月×80 人月）となる。また、カスタマイズに先立ち、パッケージ全体の仕様調査等の再利用作業が必要となり、その費用が発生する。全体規模に対し、基本設計並びに詳細設計（工程比率は各々20%、25%とする）を部分的に遂行すると見做し（遂行率を 10%とするが、当該パッケージ

のようなソフトウェアへの馴染み具合やソフトウェアの難易度等により変動する)、費用は900万円（=100万円/人月×（100KLOC÷0.5KLOC/人月）×（0.2＋0.25）×0.1）となる。従って、パッケージの購入価格を合算した取得総額は11,300万円（=2,400万円＋900万円＋8,000万円）となる。

ケース（ｄ）は、既存資源のソフトウェア（規模10KLOC）をノン・カスタマイズでそのまま再利用し、新規開発を規模90KLOC行ない、全体規模は100KLOCとする。生産性は(ｃ)以降全て同じとする。新規開発は、工数が180人月（=90KLOC÷0.5KLOC/人月)、開発費が18,000万円（=100万円/人月×180人月）となる。再利用作業費用は90万円（=100万円/人月×（10KLOC÷0.5KLOC/人月）×（0.2＋0.25）×0.1）となる（遂行率は自社の既存資源なので(ｃ)より馴染みがあり、もっと低めとなることが考えられるが、同一の設定とした)。従って、開発費総額は18,090万円（=18,000万円＋90万円）となる。なお、このケース以降、現行の会計基準では再利用作業費用を識別せず、新規開発作業総体に含めて費用を算出するから、結果的に開発費総額としては同額になるとしても、内容的な捕捉は異なるのである。

ケース（ｅ）は、大半は（ｄ）と同じであるが、既存資源のソフトウェアを再利用することで、その開発費用を一部負担することが異なる。資産計上されており、未償却残高があり、便宜上当初費用の20%を負担するとし、振替費用は400万円（=100万円/人月×（10KLOC÷0.5KLOC/人月）×0.2）となる（減価償却の状態並びに負担率の設定如何で計算は異なる)。従って、開発費総額は18,400万円（=18,000万円＋400万円）となる。このケースの場合、現行の会計基準では既存資源の再利用における費用負担は考慮していないから、新規開発費18,000万円だけの取り扱いとなり、筆者案とは異なる（図表に参考までに現行基準比率を掲示し、異なるケースに関しては網掛けをしてある)。

ケース（ｆ）は、既存資源のソフトウェア（規模30KLOC）を再利用するが、既存資源の開発費負担はない。カスタマイズをしてもらい、但しそれを遂行する共通Ｇ又は他ＰＪが費用負担し、当該ＰＪの費用負担はない。新規開発を規模70KLOC行ない、全体規模は100KLOCとする。新規開発は、工数が140人月（=70KLOC÷0.5KLOC/人月)、開発費が14,000万円（=100万円/人月×140人月）となる。再利用作業費用は270万円（=100万円/人月×（30KLOC÷0.5KLOC/人月）×（0.2＋0.25）×0.1）となる。従って、開発費総額は14,270万円（=14,000万円＋270万円）となる。

ケース（ｇ）は、大半は（ｆ）と同じであるが、カスタマイズ費用を一部負担することが異なる。カスタマイズは既存資源の50%に及ぶものとし（但し規模は変わらないものとする)、費用負担を50%とすると、カスタマイズ費用(負担分)は1,500万円(=100万円/人月×(30KLOC×0.5÷0.5KLOC/人月）×0.5）となる（規模の変動並びに負担率の設定如何で計算は異なる)。従って、開発費総額は15,770万円（=14,000万円＋270万円＋1,500万円）となる。このケースの場合、現行の会計基準では既存資源のカスタマイズ費用負担は当該ＰＪに関しては考慮していないから、開発費総額14,270万円の取り扱いとなり、筆者案とは異なる。

ケース（ｈ）は、大半は（ｇ）と同じであるが、カスタマイズ費用を全て負担することが異なる。

カスタマイズ費用は3,000万円（=100万円/人月×（30KLOC×0.5÷0.5KLOC/人月）×1.0）となる。従って、開発費総額は17,270万円（=14,000万円＋270万円＋3,000万円）となる。このケースの場合、現行の会計基準ではカスタマイズ費用としては識別せず、新規開発作業総体に含めて費用を算出するから（再利用作業費用と同様）、結果的に開発費総額としては同額になるとしても、内容的な捕捉は異なるのである。

ケース（ｉ）は、大半は（ｇ）と同じであるが、既存資源の開発費を一部負担することが異なる。振替費用は1,200万円（=100万円/人月×（30KLOC÷0.5KLOC/人月）×0.2）となる（負担率は（ｅ）と同率とする）。従って、開発費総額は16,970万円（=14,000万円＋270万円＋1,500万円＋1,200万円）となる。このケースの場合、現行の会計基準では既存資源の開発費並びにカスタマイズ費用の一部負担は当該ＰＪに関しては考慮していないから、開発費総額14,270万円の取り扱いとなり、筆者案とは異なる。

ケース（ｊ）は、大半は（ｉ）と同じであるが、カスタマイズ費用を全て負担することが異なる。カスタマイズ費用は（ｉ）と同額の3,000万円（=100万円/人月×（30KLOC×0.5÷0.5KLOC/人月）×1.0）となる。振替費用は1,200万円（=100万円/人月×（30KLOC÷0.5KLOC/人月）×0.2）となる（負担率は（ｅ）と同率とする）。従って、開発費総額は18,470万円（=14,000万円＋270万円＋3,000万円＋1,200万円）となる。このケースの場合、現行の会計基準では既存資源の開発費の一部負担は当該ＰＪに関しては考慮していないから、開発費総額17,270万円の取り扱いとなり、筆者案とは異なる。

ケース（ｋ）に関しては、（2-3）で既に数値事例まで含めて取り上げておいたので、再言しない。

以上によって、３．の定性的な記述に対して、具体的な数値事例を各ケース別に挙示することにより一層明確化し得たと考える。全11ケースのうち、4ケースは明らかに現行の会計基準に則った会計処理とは異なるものになる。全般的に、再利用を行なわない新規開発に比べ、工数並びに費用を軽減できることは明白である。再利用の仕方、既存資源の開発費用負担並びにカスタマイズの費用負担により、軽減の程度は多様である。コスト効率に注目すれば、その点が重要であろう。しかし、財務会計の観点からは、筆者が特に重視したいのは、開発実態に即した「表現の忠実性」である。現行会計基準は、再利用に関してある程度は取り扱っているが、取り立てて開発実務に裏付けられたものとは思えない限定や局部的な事象の一般化を行ない、「表現の忠実性」を損なっている。それに対し、再利用を包括的に取り上げ、それに適合的な会計処理を数値事例と併せて提示した。振替処理といったソフトウェア会計ではこれまでにない処理を追加したが、他の分野ではありふれた処理であり、特段煩雑なものではないであろう。これらを施すことで、「表現の忠実性」を具現することが望ましいと筆者は考える。

本論

第５章　オープンソース利用

１．オープンソースの概要

（1）オープンソースの概観

（2）オープンソースの利用

２．現行会計制度による捕捉の可能性

（1）現行のソフトウェア会計基準にとってのオープンソース

（2）無償のソフトウェアをオンバランス化する会計制度的な根拠

３．オープンソースへの代替的アプローチ

（1）「オープンソース評価モデル」

（2）オープンソースの「公正な評価額 ＝ 取得原価」算定モデル

（2-1）類似の商用ソフトウェアによる算定方法

（2-2）新規自社開発ソフトウェアによる算定方法

（3）オープンソースの「公正な評価額 ＝ 取得原価」算定のモデル事例

（3-1）類似の商用ソフトウェアによるモデル事例

（3-2）自社開発ソフトウェアによるモデル事例

第５章　オープンソース利用

オープンソース（Open Source）[294]は、1990 年代半ばないし 2000 年代初頭迄の「運動的な」勢いは減速してきたが[295]、その代わりに一時的な流行の時期を脱して安定成長期に入り、商用のソフトウェアと「共存」し、広く利用され定着してきたと言える。その１つの大きなメルクマールは、メガバンクの Linux 利用である。三菱東京ＵＦＪ銀行は 2008 年にシステム統合を行なったが、そのアーキテクチャとしてハードウェアはメインフレーム系の IBM3090 台とし、ＯＳは Linux を採用したのである[296]。極めて高い信頼性、安定性、堅牢性を求められるメガバンクの勘定系システムがＯＳを Linux としたことは、それだけの信頼性等を認知された何よりの証左と言える。2002 年並びに 2011 年の２度の大きなシステム・トラブルで窮地に立たされたみずほ銀行は 2016 年 12 月にシステム統合を予定しているが、ＯＳはやはり Linux を採用することにした[297]。Linux は、一時期オープンソースの代名詞ともなったほどの、オープンソースの代表的なソフトウェアであることは言うまでもない。

ソフトウェア分野におけるこうした動向が、会計にどのように関連しているのか。端的に言えば、オープンソースは基本的には無償なので[298]、どれほど取得し利用していても現行の会計制度ではオフバランスのままであり、財務諸表等では一切情報開示されていないのである。筆者は、これを問題視している。それに対して、無償ならば、「空気」のようなもので、それをオンバランス化したり、会計的に捕捉したりする必要性や意義はない、といった否定論が想起される。だが、それは的外れの混同である。第１に、オープンソースは無償であるが、空気とは違い、人工物であり、人為的に制作されたものである。そうであるからこそ、第２に、オープンソースの利用に際しては、商用ソフトウェアと同様に、ライセンス契約を締結することになっている。後述するように、商用ソフトウェアとは契約内容に異質な点はあるが、空気とは違うことの決定的な証憑である。

第３に、オープンソースの利用と、それ以外の手段・方策はソフトウェア企画において各種代替案として同列的に取り扱われる。自社開発とするか、商用ソフトウェア（パッケージ・ソフトウェ

---

[294] 他に、正式名称と言えるオープンソースソフトウェア(Open Source Software：OSS)、運動理念の表明の意味合いが強いフリーソフトウェア(Free Software、その日本での略称がフリーソフト)、あるいはフリーウェア(Freeware)といった呼称があるが、本稿では通称と言ってもよいオープンソースを使用する(但し、引用文は除く)。

[295] 出現してからまだ 20 年程度だが、その初期には思想的な運動(自由の追求)を提唱する動きがあり(呼称にもそれが反映している)、先駆的なリチャード・ストールマン(Richard Matthew Stallman)の活動やエリック・レイモンド(Eric Steven Raymond)のマニフェストとも言える『伽藍とバザール』に代表されるが、ビジネス利用を中心に普及拡大してきたソフトウェアにとっては特異な性格を有している。但し、類似の動向は、ＰＣ革命や初期インターネット等、間歇的に起こっていることではある。

[296] 大和田(2009)pp. 70-74, 186, 192

[297] 日経コンピュータ(2013)p. 13 等

[298] 「基本的には」というのは、それ自体の供与はあくまで無償だが、周辺的には注 20 で取り上げるように、ディストリビューション等が有償化されている事情を踏まえてのことである。

ア）を購入するか（カスタマイズを含む）、オープンソースを利用するか、総合的に比較評価し、システム全体又はシステムの一部のコンポーネントに関して、何れかを採用するのである。そして、何れを採用しても、事業活動に貢献することは言うまでもない。ところが、自社開発や購入であれば、資産要件を充足すれば資産計上し、そうでなければ費用処理とするが、オープンソースはどれほど事業活動に貢献しても、偶々支出が伴わない故に、会計処理はされず、オフバランスとなってしまう。この著しいアンバランスは、妥当なことであろうか、筆者には疑問である。取り分け１つのシステムが自社開発と商用ソフトウェアとオープンソースのコンポーネントを一体化して構築されている場合に、全体として「将来の経済的便益」に貢献しているにも関わらず、前二者だけが資産計上されていることは、全体像が表現されておらず、「忠実な表現」に悖るのではないか。しかも、各々のコンポーネントがどのように実際に利用されているかは、システム・ログ情報を採取し解析すれば、具体的に捕捉可能であり、貢献度合の測定も利用頻度等として可能であるのだから、尚更ではないか（なお、誤解はないと思うが、この文脈以外では、オープンソースの利用というのは、ユーザが利用するという意味ではなく、開発ないし運用で主としてコンポーネントとして使うという意味である）。

第４に、会計基準等に則るならば、「企業会計原則」には「贈与その他無償で取得した資産については、公正な評価額をもって取得原価とする」（第三 - 五 - Ｆ）という規定がある。これに依拠することができるのではないか。あるいは、ＩＡＳＢの「財務報告に関する概念フレームワーク」における資産の定義・認識・測定に依拠するならば、やはり無償取得の資産計上の手立てが講じられるのではないか。筆者は、そのように見通しを立てている。そして、オープンソースをオンバランス化するために、「公正な評価額をもって取得原価とする」こと、同じくＩＡＳＢの「信頼性をもって測定できる」こと、これらを具体的に明確化すればよいのではないか、と考えている。

本章の構成は、次の通りである。１．オープンソースの概要において、（１）オープンソースの概観として、出現の経緯、オープンソースの定義、商用ソフトウェアとは異なるライセンス契約を取り上げる。（２）では利用状況として、ＪＵＡＳの調査報告を取り上げ、最後に代表的なオープンソースを挙示する。２．現行会計制度による捕捉の可能性において、現行のソフトウェア会計基準に基づくとオフバランスとなっているが、オンバランス化を可能とする会計制度的な根拠があることを確認する。（１）では会計基準に拠ると、オープンソース本体は無償であるため、完全にオフバランスとなっていることを確認する。（２）では「企業会計原則」並びにＩＡＳＢの「概念フレームワーク」を精査し、オンバランス化を可能とする会計制度的な根拠があることを確認する。３．オープンソースへの代替的アプローチにおいて、オープンソースの「公正な評価額 ＝ 取得原価」を算定することにより、オンバランス化する具体的な会計処理方法を提示する。（１）では、オープンソースに適用する「オープンソース評価モデル」を提示する。（２）では、それに基づく「公正な評価額 ＝ 取得原価」算定モデルを提示する。まず全体的な手順等を説明し、次により具体的なものとするために、類似の商用ソフトウェアがある場合のモデルとそれがない場合の新規自社開発ソフト

ウェアによるモデルを提示する。（3）では、各々の具体的な数値例により、実用性を検証する。これらを通じて、オープンソースを財務諸表においてオンバランス化し、ソフトウェア資産の全体像の「忠実な表現」を実現することが、意思決定有用性を高めることになることをも提言する。

１．オープンソースの概要

（1）オープンソースの概観

第１に、他の分野では余り類を見ない特異なものが、ソフトウェア分野で出現した歴史的経緯を取り上げる。「1991 年の秋、トーヴァルズは Minix から離れ、自作の新たなＯＳのカーネルのソースコード「Linux」をインターネットのニュースグループにリリースした」[299]。そして、「1994 年に、トーヴァルズは公式な最初の Linux であるバージョン 1.0 をリリースした。開発速度は 1990 年代を通して加速を続けた。／1990 年代末までには、Linux は大きな技術的な現象であると同時に、市場現象にもなっていた。きわめて複雑で洗練されたＯＳが、世界に散らばった数千人の開発者の自発的な貢献によって構築されたのだ。2000 年の中頃までに、Linux はウェブサーバの三分の一以上を占めるようになっていた」[300]。こうして、オープンソースの初期における代名詞とも言える Linux が出現した。そして、それを主要な契機として、多数のオープンソースが陸続と作られるようになった。余り普及せずに消えていくのが、現在でも圧倒的多数のオープンソースの命運ではあるが、それでも生き残り、利用されているものもざっと８万点以上に上る[301]。

第２に、オープンソースの定義の核心を確認する。オープンソースとは、その名の通り、①ソースコードが公開されている、②無償で誰でも利用できる、③自由に改変できる、ソフトウェアである。更にライセンス契約にもよるが、④その改変したソフトウェアを第三者に提供することも可能で、無償が基本だが、有償で提供して差し支えない場合もある[302]。これが、商用ソフトウェアとどれほど異なるか。商用ソフトウェアは、有償での販売であることは言うまでもないが、一般的には

---

[299] Weber(2004)邦訳 p.77。なお、「Minix」は高名なＯＳ研究者アンドリュー・タネンバウム(Andrew Tanenbaum)が教育用に作成した簡易版ＯＳである(同書 p.76、Tanenbaum(2006)邦訳 p. vi)。

[300] 同書 pp.77-78

[301] オープンソース・ポータル(SourceForgehttp:/www.sourceforge.net/)等参照。

[302] Golden(2005)邦訳 p.2 等を参考に、筆者なりに言い換えた。「核心」としたのは、付随的な定義は他にもあるからである。‘The Open Source Initiative’（オープンソースの定義）（八田真行訳）は、全箇条の項目を挙示すると、次の通りである。「1.再頒布の自由」、「2.ソースコード」の頒布許可、「3.派生ソフトウェア」〔頒布許可〕、「4.作者のソースコードの完全性(integrity)」〔変更されたソースコードの頒布制限可〕、「5.個人やグループに対する差別の禁止」、「6.利用する分野(fields of endeavor)に対する差別の禁止」、「7.ライセンスの分配(distribution)」〔再頒布制限不可〕)、「8.特定製品でのみ有効なライセンスの禁止」、「9.他のソフトウェアを制限するライセンスの禁止」、「10.ライセンスは技術中立的でなければならない」
(http://www.opensource.jp/osd/osd-japanese.html(アクセス 2014/05/26,11:25)、なお〔 〕は筆者の補足である)。

オブジェクト提供[303]であり、ソースコードは非公開である。たとえソース提供している場合でも、改変するには提供元の許諾が必要であり、一般的には提供元しか改変できない契約になっている。更に改変したソフトウェアの第三者提供を許諾することはない。ソースコードを非公開とするのは、特許権や著作権による防御と併せて、「企業機密」として競争上の優位性を担保することが主目的であるが、改変させないことをも目的としている[304]。両者の違いは、判然としている。オープンソースは、出現当初は取り分け「独占的ソフトウェア」を標的とし[305]、それへの対抗として活発に取り組まれたので、こうしたオープンな又はフリーな性格を強烈に有しているのである。

第３に、オープンソースの特異な性格を保証し、且つ普及を促進せしめた特異なライセンス契約を概観する。オープンソースを利用する場合には、このライセンス契約を締結するのである。実際には様々なバリエーションがあるが、代表的なライセンス契約の雛型は４つある。ＧＰＬ（General Public License）、ＬＧＰＬ（Lesser GPL）、ＭＩＴ／Ｘ（MIT/X Window System）、ＢＳＤ（Berkeley Software Distribution）である。ＧＮＵ（GNU’s Not Unix）[306]のＧＰＬは、２つの要求事項があり、「１．すべての派生物――つまり、ＧＰＬが適用されたコードを含んだあらゆるプログラム――は、それ自体もＧＰＬで配布しなければならない」、「２．オリジナルでも、派生物であっても、そ

---

303 このオブジェクトは、プログラミング言語で記述したソース・プログラムに対して、２進コードの機械語に変換されたオブジェクト・モジュールないしロード・モジュールの慣用的な総称である。

304 当然に契約上の縛りはあるが、ソースコードであれば、容易に解読し、改変もしくは流用することが可能だからである。なお、オブジェクト形態でも、リバース・エンジニアリング（逆コンパイル）により、ソースコードを復元することは技術的には可能である。リバース・エンジニアリングとは、機械語からソースコードに逆変換することであるが、ソースコードからオブジェクトに変換するコンパイラがコード形態の変換だけではなく、最適化(主として処理速度並びにメモリ使用に関して)を行なっているので、逆変換は一意的とはならず、原型復元は試行錯誤により近似的たらざるを得ない。なお、著作権法上は、リバース・エンジニアリングをすること自体は合法という説が有力であり、それを基に「クリーンルーム(Clean room)方式」又は「アイソレーション・ブース(isolation booth)方式」(解析班と開発班を完全に分離し、「表現の授受」が行なわれないようにする方式)によって開発したものを利用・販売等行なっても、元の著作権を侵害したことにはならない、という解釈がある(植松(2000)pp. 217-221)。但し、金井、小倉編著(2000)には、「開発段階にアセンブリ言語で書かれたソースプログラムを機械語に変換したオブジェクトプログラムの複製物を、著作権社に無断で逆アセンブリしてソースプログラムを抽出し、雑誌に掲載出版した行為が複製権侵害にあたるとされた事例(東京地判昭和 62 年 1 月 30 日判時 1219 号 48 頁)」(p. 229、藤田耕司執筆)が挙げられており、法解釈として確定しているとは言い難い。

305 かつて標的にされ、企業防衛の立場から当初は専ら敵対的に対抗していた「独占的ソフトウェア」の企業側の対応は、次第に態度が軟化してきている(Fogel(2006)邦訳 pp. 105-125、日本 JBoss ユーザ・グループ(2008)p. 329 等)。最早敵対一方では、販路を閉ざしかねない程にオープンソースが勢力を伸張した証とも言える動向である。そして、最大の標的であったマイクロソフトがついに、2012 年 10 月にオープンソースを開発する子会社マイクロソフト・オープン・テクノロジー社を設立した。14 年前に、社内文書「ハロウィン文書」で「Linux は癌だ」と敵視していたことからすれば、隔世の感がある（これらのことは業界では周知の事実なので、特に出典は不要と判断する)。

306 リチャード・ストールマンが GNU プロジェクトとフリーソフトウェア財団(Free Software Foundation)を設立し、フリーソフトウェアの運動拠点にしたが、それに端を発するものである(Fogel(2006)邦訳 pp. 7-9)。

れを再配布する場合には制限を追加してはいけない」、というものである[307]。ＧＮＵのＬＧＰＬ[308]は、「ＧＰＬよりも制限が緩く、フリーでないコードと容易に混ぜることができ」るものである[309]。ＭＩＴ／Ｘライセンスは、「基本的に「あなたはこのコードを自由に、無料で使うことができますよ」ということ」であり、「自分のコードをできる限り多くの開発者と派生物で使ってもらい、かつ独占的なコードで使われることを気にしないならば」適しており、「ＧＮＵ ＧＰＬと互換性があり、短く、簡単で、理解しやすいもの」である[310]。ＢＳＤライセンスは、「ＭＩＴ／Ｘライセンスと似ているが、以下の一節だけは異なっている」ということで、その箇所を挙げると、「このソフトウェアの機能や、このソフトウェア自体を使っていることを言及する広告媒体には、必ず以下の謝辞を記載しなければならない。「この製品にはXXXXが開発したソフトウェアが含まれています」」という「宣伝条項」を義務付けているのである[311]。なお、1999 年以降の「修正ＢＳＤライセンス」は、「宣伝条項を削除したもの」である[312]。いずれであれ、商用ソフトウェアのライセンス契約との違いは判然としている。

第４に、オープンソースの特徴的なことを付け加える。これが、紛れもなくインターネット時代の産物だということである。オープンソースの開発は、世界の各地から多くの技術者が参加して進められるが、これを可能にしたのはインターネットである。そうでなければ、対面したこともない多数のメンバーが、業務命令といった強制もなく、意思疎通を図り、開発を完遂し、保守を継続することはまず困難であろう。他方、今日ではＷｅｂシステムがシステム形態としては主流になりつつある、あるいは既になったと言えるが、これまでの比較的クローズだったシステムと比べ、そのオープンな性格故に、技術的には格段に高度なものである。不特定多数のユーザーが、様々に異なるプラットフォームで利用するからである。それを可能にした要因は多々あるとしても、技術的に支えているのは、少なからずオープンソースなのである。こうした意味で、インターネットを支持基盤とするオープンソースが、相乗作用的にインターネットの進化・発展に貢献していると言える。更に、その延長上で、2000 年代後半に注目されるようになったクラウド・コンピューティング（cloud computing）は、そのアーキテクチャの相当程度をオープンソースで構築・利用することで成り立っているのである。

（２）オープンソースの利用

オープンソースの利用の仕方[313]は、様々であり得るが、独立的なシステムとして利用する場合は

---

307 同書邦訳 p. 247

308 元々は「GNU Library GPL」の略であった（同書邦訳 p. 249）。

309 同書邦訳 pp. 249-250

310 同書邦訳 pp. 248-249

311 同書邦訳 pp. 251-252。なお、一部「である」調に、また具体的な組織名を「XXXX」に改変した。

312 同書邦訳 p. 252

313 本文で言う利用の仕方とは少々異なる視点から捉えた「利用の仕方」については注記に留めるが、１つは、ディストリビューション（Distribution）である。環境構築にはＯＳのコマンド操作等を行

比較的少ない。むしろ大半は、コンポーネントとして利用するのである。具体的には、開発における開発ツールとして利用するか、フレームワークとして開発時の利用と併せて、運用時にもソフトウェアの一部として利用し続ける。後者は一般的にはミドルソフト（ミドルウェア）と呼ばれる範疇である。この分野では、無償というコスト・メリットだけではなく、機能等で商用ソフトウェアを凌駕しているものも少なくない。そして、アーキテクチャの要所々々をオープンソースで構成するという利用の仕方は今日では広く普及している[314]。

オープンソースが実際にどの程度利用されているか、ＪＵＡＳの調査報告を取り上げる。「調査対象は、東証一部上場企業とそれに準ずる企業の計 4000 社で、各社 IT 部門長に調査票を郵送し、1039 社（有効回答率：26%）の回答を得た」[315]もので、これ以上の規模の同種調査はないので貴重と言える。「新規テクノロジーの導入状況」として、オープンソース（OSS）も対象となっている。

オープンソースのソフトウェア分野は、⑨ＯＳ（Linux、FreeBSD など）、⑩ミドルウェア（JBoss、Apache、Sendmail など）、⑳業務用アプリケーション（CRM など）、21.Office 系アプリケーション（OpenOffice など）に区分されている。カッコ内は調査票の注記にある例示である。回答の選択肢は、「A．現状」に関して５つが上げられており、「B．3 年後の予測（導入の場合○）」も付加されている。回答数値を、下記の表に一覧化した[316]。

図表１−５−1　OSS の導入状況

	回答数	導入済み	試験導入中・導入準備中	検討中	検討後見送り	未検討	導入率順位
⑨OSS・OS	1018	34.4	3.0	10.4	4.2	47.9	3
⑩OSS・ミドルウェア	1007	33.5	3.4	8.2	3.4	51.5	4
⑳OSS・業務用アプリケーション	1005	12.5	2.7	18.1	5.0	61.7	11
21.OSS・Office系アプリケーション	1008	8.2	3.5	20.2	10.1	57.9	16

（回答数：実数、それ以外の数値：比率%）

（出典：ＪＵＡＳ(2012)から数値抽出、表化筆者）

⑨と⑩は、「導入済み」企業が 30%を遥かに超えている。導入率の順位も上位を占めており、更に上位には「仮想化（サーバー）」（導入済み 48.9%）と「Web 会議」（導入済み 36.0%）があるだけである。それに対して、⑳と 21.は、10%前後と低調で、導入率順位も中位と下位である。また、調査結果の内訳に「売上高別導入状況」が示されており、数値は省略するが、売上高規模が大きい

なうので専門的な技術を必要とするが、その代替としてインストーラ(インストール作業を行なうソフトウェア)等がパッケージ化されたものであり、簡便にインストール可能なものである(Fogel(2006)邦訳 p.23)。チュートリアル的な説明書等も併録されている場合と、ドキュメント等は別扱いで提供される場合もある。もう１つは、ベンダーのサポート等を有償で受け、開発することである。商用ソフトウェアの購入により得られるのと同様なサポートを、オープンソースでは即自的には得られないことの代替である。何れも、無償のオープンソース本体とは異なり有償であるが、便益又は未経験で開発に難渋することを考えれば、有償でも意義があるのである。

[314] ＩＰＡ(2007)p.29。同書には実際の事例も紹介されている。なお、岡田(2011)p.XII(他全編)は、実例ではないが、更にオープンソースを多用する場合のソフトウェアの構築を推奨している。

[315] ＪＵＡＳ(2012)p.ix

[316] ＪＵＡＳ(2012)pp.3-4,10-11,16-18,A-3

企業ほど導入率が高い。こうした傾向は、オープンソースの大凡の状況を示しており、ソフトウェア先進技術に関する一般的なキャッチアップ傾向とも符合していて、順当な結果と言えるであろう。

オープンソースは、ソフトウェア分野において、特異な一分野あるいは傾向を占め続けることは間違いないが、概観の締め括りとして、実際にどのようなソフトウェアがあるか、代表的なものを挙示しておく。①OS:Linux、FreeBSD 等、②DBMS:PostgreSQL、MySQL、SQLite 等、③仮想化ソフト:Xen、KVM 等、④Web サーバ:Apache、Tomcat 等、⑤AP サーバ:Struts、Spring、GlassFish、JBoss、Open JPA、Jersey 等、⑥スクリプト言語（簡易言語）:Perl、Python、PHP、Ruby 等、⑦O/R マッパ（オブジェクト指向のオブジェクトとリレーショナル DB のマッピング）を行なうツール）:Hibernate、TopLink 等、⑧Web フレームワーク:Tapestry、Wicket、WebWork、DWR 等、⑨IDE（統合開発環境）:IntelliJ IDEA、Eclipse、NetBeans 等である。

２．現行会計制度による捕捉の可能性

（１）現行のソフトウェア会計基準にとってのオープンソース

現行のソフトウェア会計基準は、日本でオープンソースが本格的に利用されるようになった端境期に当たる 1998 年に制定されたので、オープンソースを全く想定していない[317]。それ以降、ソフトウェアに係る改訂としては、2006 年に企業会計基準委員会実務対応報告第 17 号「ソフトウェア取引の収益の会計処理に関する実務上の取扱い」が公表され、2007 年に企業会計基準委員会「工事契約会計基準」が公表されているが、何れも収益認識に係るものである。時期的にはオープンソースが相当程度に普及・利用されるようになっていたが、それに係る会計処理を追加・改訂するということはなかった。2010 年代に入って、オープンソースが更に利用拡大されているが、そうした事象・事態に対して基準を見直す動向は全くない[318]。従って、現時点でも、そして近い将来においても、オープンソースに関しては、会計基準では想定外の事象・事態のままであり続ける。

従って、利用するオープンソースそのものは無償であり、支出額がゼロなので、現行の会計処理では全く処理対象とされず、完全にオフバランスとなっている。有償のソフトウェアと一体的に「資産」を形成し、「将来の経済的便益」に貢献しても、会計処理の対象にはなっていない。

但し、そもそもオフバランスが問題となるのは、金融資産のように、価格変動が激しく、巨額の損失が発生する可能性のある高リスクの資産だからではないのか。オープンソースを含めてソフトウェアには、そこまでの高リスクはないと言えるのではないか。従って、オフバランスのままであっても取り立てて問題ではないとする意見があり得るかもしれない。だが、今日ではソフトウェア

---

317 これは、日本基準に留まらず、アメリカ基準(1985 年制定の SFAS86 号並びに 1998 年制定の SOP 98-1)並びに国際会計基準(1998 年制定の IAS 第 38 号「無形資産」)でも同様である。

318 企業会計基準委員会の第 197 回委員会（2010 年 3 月 11 日）、第 242 回委員会（2012 年 4 月 19 日）、第 251 委員会（2012 年 9 月 5 日）の審議等。

に関して、そのような意見は根拠薄弱になっている。ソフトウェアは、本論第６章で詳細に取り上げているように、システム・トラブルが大きな社会問題や信用問題になる高リスクの資産となっているのである。資産価値が乱高下する金融資産とはリスクの性格が異なり、利用上のリスクあるいはそれ以前の構築上のリスクではあるが、高リスクであることに変わりはなく、その意味でもオフバランスでよいということにはならない。そして、本章の主旨では、何よりも経済的実態に適合するようにオンバランス化するべきである、と考える。

（2）無償のソフトウェアをオンバランス化する会計制度的な根拠

「企業会計原則」には、周知のように、「贈与その他無償で取得した資産については、公正な評価額をもって取得原価とする」（第三 - 五 - Ｆ）という規定がある。それに関して「国庫補助金、工事負担金等で取得した資産」の取り扱いを定めた注が付されている（注 24)。これらから解すると、この規定が想定している無償取得とは、贈与等によるものであり、特定の或る取得が無償の場合である[319]。関連して出された「企業会計原則と関係諸法令との調整に関する連続意見書」においても、「固定資産を贈与された場合には、時価等を基準として公正に評価した額をもって取得原価とする」（連続意見書第三　第一 - 四 - ５）とある。「時価等を基準として」ということが、「贈与された場合」等以外では有償での取引（取得）が行われるものであることを意味していると解することができる。それに対し、オープンソースは特定の或る取得に限らず、「取引」（取得）全般が無償であるので、企業会計原則が想定している事象とは異なる、新たな事象であると言える。従って、企業会計原則の規定がそのまま適用できるということにはならないであろう。特定の或る取得が無償であるだけではなく、取得全般が無償であるものに対して、「無償取得」の範囲を拡大して適用することが必要になる。それは財としては有用で「将来の経済的便益」に貢献するのであるから、有償取得の資産に準じて資産要件を満たしているならば、無償取得の範囲を拡大して適用することが認容されるのではないだろうか。そうであれば、「時価等を基準」とした「公正な評価額を取得原価とする」ことができるのではないだろうか。

そこで、次には、オープンソースが資産要件を満たしているかどうかが問題となる。資産の定義・認識等に関して詳細に規定しているＩＡＳＢの「財務報告に関する概念フレームワーク」(2010 年 9 月）に拠ると、資産の定義のうち、①「資産に具現化された将来の経済的便益とは、企業への現金及び現金同等物の流入に直接的に又は間接的に貢献する潜在能力である」（4.8）ということに関しては、前述した通り、自社開発したソフトウェア並びに購入した商用ソフトウェアと遜色なく、オープンソースは充足する。例えば、製造業における工場の製造装置に組み込まれるソフトウェアは、物理的な機器類等を作動させ、あるいは制御・監視することを通して、機器類並びに作業員と協同

[319] 同原則では無形固定資産に関しては有償取得に限定しているとも解せられるかもしれないが（第三 - 五 - Ｅ)、それは「営業権」等(注 25)のことであり、無形であっても独立的な会計基準で明確に資産として認められているソフトウェアは無償取得の規定の対象となり得ると解せられる。

して製品の製造に貢献する。ネット通販のシステムであれば、商品情報を提供し、注文・決済等を行えるようにして、販売に貢献する。社内の業務システムであれば、社内業務の遂行並びに管理を情報面で担い、業務に貢献する。これらは、いずれも正しく「企業への現金及び現金同等物の流入に直接的に又は間接的に貢献する潜在能力である」と言える。

②「多くの資産、例えば、有形固定資産は、物的形態をとっている。しかし、物的形態は資産の存在に不可欠なものではない。(中略) それらから将来の経済的便益が企業に流入することが予想され、かつ、企業によって支配されている場合には、資産となる」(4.11) ということに関しては、無形であっても問題ではないということであるし、「将来の経済的便益」の「流入」は前述の通り他のソフトウェアと同様にあり得るし、商用ソフトウェアのライセンス契約と類似のライセンス契約を締結して利用するという意味で「支配」要件を充足する。なお、複製の利用であるから、有形物のような排他的な支配とはならないが、それは商用ソフトウェアでも同じことである。

③「企業の資産は、過去の取引又はその他の過去の事象から生じる」(4.13) ということに関しても、ライセンス契約締結が「取引」に該当すると言って差し支えないし、一定の労力を費やして環境構築することで利用可能となることは「過去の事象から生じる」ことを充足する。そして、「支出の発生と資産の取得とは密接な関係を有するが、これらは必ずしも一致するとは限らない。(中略) 関連する支出がなかったとしても、ある項目が資産の定義を満たし、その結果、貸借対照表に認識される候補となることを妨げるものではなく、例えば、企業に贈与される項目は、資産の定義を満たす可能性がある」(4.14) ということが、無償取得のオープンソースに大きく関わっている。「関連する支出がなかったとしても」「貸借対照表に認識される候補となることを妨げるものではな」いのであり、例示の「贈与」と無償取得という意味で類同と見做せるのであり、それ以外の全ての「資産の定義」を上記の通り満たしているのであるから、「貸借対照表に認識される」「ことを妨げるものでは」ない、と言えるのではないか。

そうであるならば、残るはただ１点、資産認識における「当該項目が信頼性をもって測定できる原価又は価値を有している場合」(4.38、4.44) を充足するかどうかではないか (測定 (4.54、4.55) の可能性に関しても同じことであろう)。

なお、ＩＡＳＢは 2013 年 7 月 18 日にディスカッション・ペーパー「財務報告に関する概念フレームワークの見直し」を公表した。それに関する意見募集をＡＳＢＪ（企業会計基準委員会）が同年 8 月 12 日から開始した。「見直し」によって資産の定義等が改訂される可能性は高いが、現時点では「見直し」の途上にあり、未確定である。そのため、確定的なこととして依拠することはできないが、筆者の論点を補強する事項を参考までに挙げておきたい。「企業は、経済的資源の使用を指図する現在の能力を有していない場合には、経済的資源を支配していない」とし、「次のようなものは企業の資産ではない」としている。「(ａ) 公共財 (一般道路など) へのアクセスの権利 (同様の権利が誰でもコストなしに利用可能である場合)」(3.29) を挙げているが、「コストなし」をどう捉えるかにもよるが、利用するのに一定のスキルを必要とすることを「コスト」と解すれば、オープ

ンソースはここでいう「公共財」には該当しないと言える。「（b）アクセスが制限されていない水中の魚。経済的便益の潜在的な源泉ではあるが、当該便益が誰にでも利用可能なので、どの企業にとっても経済的資源ではない。(以下略)」(同)を挙げているが、「アクセスが制限されていない」「誰にでも利用可能」ということは、ライセンス条項を許諾し、契約締結して初めて利用できるオープンソースには該当しないと言える。「（c）公知となっていて、誰もが多大な労力やコストなしに自由に利用可能である知識。どの者も、こうした知識を支配していない」（同）を挙げているが、「多大」かどうかは議論の余地があり得るが、やはり「労力やコストなしに自由に利用可能である」ということは、オープンソースは（a）に関連して触れたように、環境構築して利用するには一定のスキル並びに労力を必要とすることからしても該当しないし、「支配」に関してもライセンス契約で触れた通り、支配していると言える。以上のことから、「見直し」において、「企業の資産ではない」として挙げている例示には該当しないという意味で、消極的ではあるが、あるいは反証的に「経済的資源」であり、「資産」である、と言うことができるのではないかと考える。また、例示に先立つ規定に関しても、ライセンス契約を締結することは利用の意思があるからであり、利用しているということは一定のスキルがあってのことであり、従って「経済的資源の使用を指図する現在の能力を有してい」る場合に該当するので、「経済的資源を支配してい」ることになる、と言える。

従って、自社開発並びに購入のソフトウェアが資産計上を認められている限り、金額の測定可能性をクリアできるならば、オープンソースを自社開発や購入のソフトウェアと並び資産計上することができるのではないだろうか。それによって、「経済的便益」に貢献するソフトウェアの全体像の「忠実な表現」とすることができるであろう。なお今日では、公正価値評価という取り扱いとすることも考えられるが、事業用資産に関しては顕著な進展はないし、何よりも自社開発や購入と異なる取り扱いになってしまうことは望ましくないと考えるので、「公正な評価額を取得原価とする」方途を採用したい。〔ＩＡＳＢの議論に関しては、本章末尾のコメント参照。〕

３．オープンソースへの代替的アプローチ

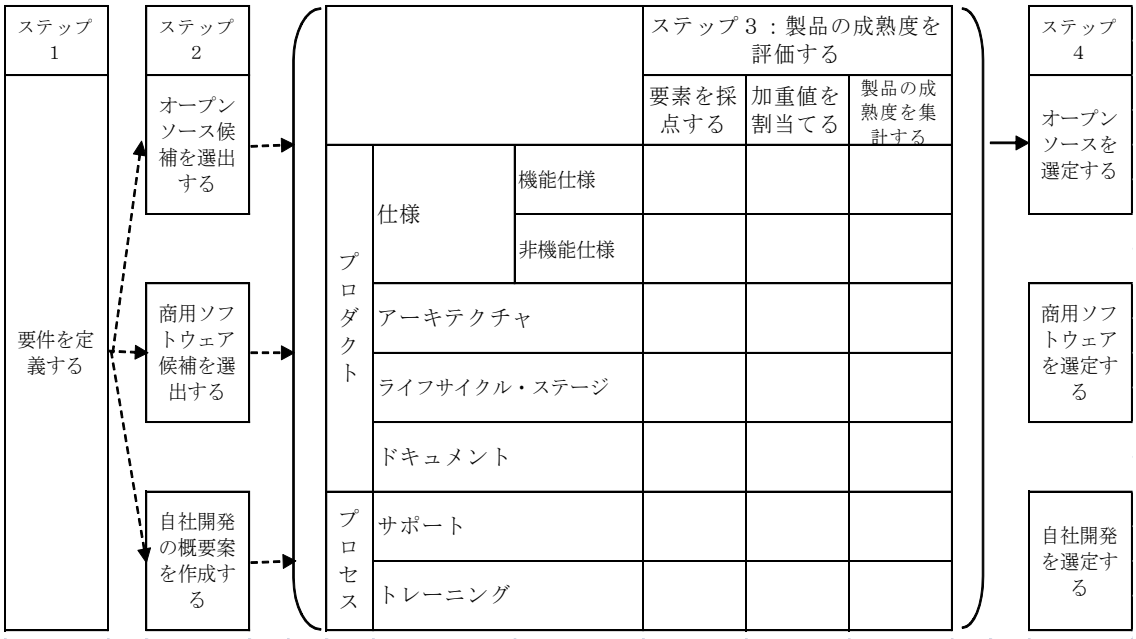
（1）「オープンソース評価モデル」

無償であるオープンソースを会計において捕捉するために、無償取得したオープンソースの「公正な評価額を取得原価とする」具体的な算定方法を提示したい。「オープンソース評価モデル」は、オープンソースを含めたソフトウェアの各種代替案を比較評価し、そのいずれかを選定するためのモデルであり、オープンソースを的確に評価するに相応しい項目を設定していることが特徴的な評価モデルである。それ故に、オープンソースの「公正な評価額 ＝ 取得原価」の算定に援用したいと考える。筆者案の「オープンソース評価モデル」は、バーナード・ゴールデン（Bernard Golden）

の「オープンソース成熟度モデル」(Open Source Maturity Model)[320]とアンソニー・J・ラタンゼ(Anthony J. Lattanze)の「ＡＣＤＭにおける商用製品の認定」(Commercial Element Qualification with ACDM)[321]モデルを参考とし、それらを基本的に典拠としているが、多少の改変を施したものである。

図表に沿って説明する。これは、第１章で取り上げたソフトウェア企画の典型的なプロセスとも言える。評価ステップは、大きく４つに分かれる。ステップ１では、求めるソフトウェアの要件を明確に定義する。ステップ２では、それに基づき、ソフトウェアの候補（案）を選出ないし作成する。ステップ３では、個々の評価項目に関する評価を行なう。ステップ４では、ステップ３の評価を受け、最終的に開発計画と突き合わせて、何れかを選定する。そして、オープンソースを選定した場合には、これらの候補（案）並びに評価点が、「公正な評価額 ＝ 取得原価」算定の基礎データとなる。

図表１-５-２　オープンソース評価モデル（全体像）



（出典：Golden(2005)と Lattanze(2009)を元に、筆者作成）

「公正な評価額 ＝ 取得原価」算定に援用する、ステップ３の評価点の算出方法は、次の通りである。まず、各評価点は、満点に対して、充足度合いあるいは適合度合いで採点する減点方式である。次に、加重値は、デフォルトに対し、当該ソフトウェアの特性を勘案して、今回採用値を設定

[320] Golden(2005)による。同書はオープンソースを選定するために、一般的なソフトウェア選定の枠組みを踏襲しながらも、オープンソースに特有の事情を考慮した評価項目を設定している点が大きな特徴である。但し、アメリカ国内の事情に即しているので、日本で適用する場合には当て嵌まらないことがあり、適宜変更することが必要となる。多少の改変を施した理由でもある。

[321] Lattanze(2009)による。なお、ＡＣＤＭ(Architecture Centric Design Method：アーキテクチャ中心設計手法)は同書の主題であり、ソフトウェアの設計をアーキテクチャを「中心」として行なう手法である。

する。そして、各評価点と採用した加重値を乗じて、計算値を算出し、それらを合計すれば、全体の評価点が求められる。なお、各評価項目の数や満点を、独自の判断で変更しても差し支えないが、理由は明確にする必要がある。また、全体の満点は100点となる設定であり、個々の変更を行っても、比較に利用しやすいように、全体の満点が100点となるように調整することが望ましいが、任意の点数としても理由が明確であれば差し支えない。但し、「オープンソース評価モデル」の評価において満点を100点としない場合でも、「公正な評価額 ＝ 取得原価」算定に援用する評価点としては100点満点換算をしなければならない。

図表１-５-3　オープンソース評価モデル（部分詳細像）

			ステップ３：製品の成熟度を評価する							
			要素を採点する				加重値を割当てる		製品の成熟度を集計する	
			評価項目	満点	評価点	合計	ﾃﾞﾌｫﾙﾄ	採用値	計算値	合計
(1) プロダクト	(1-1)仕様	(1-1-1) 機能仕様	(1-1-1-a)機能	30			1			
		(1-1-2) 非機能仕様	(1-1-2-a)品質	5			1			
			(1-1-2-b)性能	5			1			
			(1-1-2-c)セキュリティ	5			1			
			(1-1-2-d)可用性	5			1			
			(1-1-2-e)ユーザビリティ	5			1			
			(1-1-2-f)保守性	5			1			
	(1-2) アーキテクチャ		(1-2-a)普及度	5			1			
			(1-2-b)親和性	5						
	(1-3) ライフサイクル・ステージ		(1-3-a)ステージ	5			1			
			(1-3-b)更新継続性	5						
	(1-4)ドキュメント		開発者作成又は市販ドキュメント	5			1			
(2) プロセス	(2-1)サポート		作成元又は業者のサポート	10			1			
	(2-2)トレーニング		チュートリアル等	5			1			

（出典：：Golden(2005)とLattanze(2009)を元に、筆者作成）

次に、評価部分の詳細な図表に沿って、個々の評価項目に関して説明する。(1)プロダクトは、実質的な内容である(1-1)仕様と、それ以外に分けられる。(1-1-1)機能仕様の評価項目は(1-1-1-a)「機能」１つとし、総括的な評価としたが、個々の機能を複数列挙し、個別に評価する方式でもよい。(1-1-2)非機能仕様も評価に欠かせないものであり、当モデルでは主要なもの６つを挙示しているが、独自に設定しても可とする[322]。(1-1-2-a)品質は、狭義の品質とし、具体的にはバグ残存率等である。(1-1-2-b)性能は、処理の応答時間、単位時間当たりの処理件数等である。(1-1-2-c)セキュリティは、個々のソフトウェアでは、重要なデータは暗号化しているか、脆弱なセキュリティ・ホールを作らないようにしているか等である。(1-1-2-d)可用性は、稼働率（無障害で利用できる程度）や障害復旧時間（復旧に要する時間）等である。(1-1-2-e)ユーザビリティは、使い易さと言ってもよく、操作性や画面の見易さ等である。(1-1-2-f)保守性は、開発後に欠かせない保守のし易さである。これらに限らず、必要に応じて異なる項目を挙げても差し支えない。また、求めるソフト

[322] ＪＵＡＳ(2008)pp. 15-27 参照

ウェアによって、必要とする非機能仕様は相当異なるので、取り上げる評価項目並びに加重値の設定等は、適宜変更して差し支えない。

(1-2)アーキテクチャは、プロダクトの構造並びに実現する技術に関することである。(1-2-a)それがどの程度普及しているものか、また開発する全体のソフトウェアと、その一部にオープンソースを組み込む場合には両者の(1-2-b)親和性を評価する。(1-3)ライフサイクル・ステージは、評価対象のオープンソースが創生期、普及初期、普及拡大期、コモディティ期（安定期）、レガシー期のいずれにあるかである(1-3-a)。また、(1-3-b)更新継続性はそれと関連して、今後もバージョンアップが行なわれていくかの評価である。(1-4)ドキュメントは、仕様等を確認するためにどの程度整備され、利用できるかであり、作成元がそれをどの程度提供しているか、または解説書やマニュアル等が市販されているかの評価である。以上が、プロダクトに関することである。

(2)プロセスに関しては、利用する際に、どの程度外部の支援等が得られるかが主要な評価項目である。(2-1)サポートは、作成元の直接的な支援が得られるか、またはサポートあるいはコンサルティング等をビジネスとしているソフトウェア企業があるかを評価する。(2-2)トレーニングは、技術的に未習熟の場合、外部の教育等をどの程度受けられるかの評価である。なお、これは利用しようとするオープンソースが如何なる分野かによっても重要度が異なり、開発効率への影響も大きく異なるで、学習曲線の推移（予測）は大きな評価対象となる。

（2）オープンソースの「公正な評価額 ＝ 取得原価」算定モデル

次に、「オープンソース評価モデル」に基づいて、具体的に「公正な評価額 ＝ 取得原価」を算定する、オープンソースの「公正な評価額 ＝ 取得原価」算定モデルを提示する。

まず、「公正な評価額 ＝ 取得原価」を算定するに当たり、前提的な事項を述べておきたい。第１に、「連続意見書」にもあるように、基準となる「時価」があるならば（但し、当該オープンソース自体の時価ではないが)、それを優先的に適用することが望まれる。また、情報の硬度の点からも、その方が優位的であると言える。従って、類似の商用ソフトウェアがある場合は、それを対象として比較評価を行い、オープンソースの「公正な評価額 ＝ 取得原価」を算定する。類似の商用ソフトウェアがない場合は、オープンソースと類似のソフトウェアを、新規に自社開発すると仮定し比較評価を行い、「公正な評価額 ＝ 取得原価」を算定する。

第２に、ソフトウェア開発実務において、開発や導入を決定する際には、まずソフトウェア企画において複数の代替案を考案し、それらの比較検討を行った上で決定することになっている。商用ソフトウェアの購入、自社開発、オープンソースの利用、のいずれを選定するか、という比較検討・評価である。そこで、企画で比較検討に挙げられた候補がある場合は、オープンソースの「公正な評価額」算定において、それを比較評価の対象とする。それにより、「公正な評価額 ＝ 取得原価」算定の負荷軽減にもなる。もし複数の候補による比較検討を十分に行なわなかった場合には、「公正な評価額 ＝ 取得原価」算定の際に改めて候補を挙げることになる。

第３に、上記の企画における選定の際に、複数の類似の商用ソフトウェアが候補に挙がった場合には、まずは全ての候補を比較対象候補として個々に「オープンソース評価モデル」により評価を行なう。そして、オープンソースの評価額が最も低くなるものに最終的な比較対象を絞ることにする。複数の比較対象候補の絞り方は、評価の高いもの、評価の低いもの、複数の評価の平均、オープンソースの評価と近似的なもの、とする等の選択があり得るが、例えば入札等のビジネス慣行からすれば、評価額が最も低くなるものを選択することが穏当であると考える。

第４に、１つのソフトウェアないしシステムの構築に複数のオープンソースを利用し、それに対応する類似の商用ソフトウェアも各々ある場合には、個々に評価をして、その合算値により評価額を算定する。類似の商用ソフトウェアがない場合に、その部分を自社開発すると仮定した場合も同様である。

（2-1）類似の商用ソフトウェアによる算定方法

オープンソースと、類似の商用ソフトウェアとの比較評価並び算定は、次の通りとする。ステップ１は、類似の商用ソフトウェアの購入価格（見積額）を設定する。ステップ２は、各々を「オープンソース評価モデル」により評価する。ステップ３は、ステップ１で設定した商用ソフトウェアの購入価格（見積額）と、ステップ２で評価した各々の評価点により、オープンソースの「公正な評価額 ＝ 取得原価」を算定する。

算定式は次の通りである。オープンソースを「A」、商用ソフトウェアを「B」、Bの購入価格（見積額）を「x」とする。「オープンソース評価モデル」による各々の評価点は、Aが「a」、Bが「b」とする。その場合、Aの「公正な評価額 ＝ 取得原価」は、$y=(a/b)\,x$、となる。なお、各々利用に掛かる費用も発生するので、それを加味すると、算定式はもう少し複雑になるが、それに関してはモデル事例の算定で説明する。

複数のオープンソースを利用した場合には、各々の比較評価を行い、各「公正な評価額 ＝ 取得原価」を算定し、合算する。利用したオープンソースをA_1，A_2，…，A_nとすると、「公正な評価額 ＝ 取得原価」$=\Sigma\,(y_1+y_2+\cdots+y_n)$、となる。

（2-2）新規自社開発ソフトウェアによる算定方法

オープンソースと、新規に同様のソフトウェアを自社開発すると想定した場合の比較評価並び算定は、次の通りとする。ステップ１は、ソフトウェア測定を利用して、オープンソースと同様のソフトウェアを自社開発した場合の開発費（見積額）を算出する。ステップ１αは、自社開発費（見積額）を商用ソフトウェアの市場価格に換算する。商用ソフトウェアとの比較評価並びに算定との違いは、このステップが追加されることである。換算比率を「商用ソフトウェア換算係数」と呼ぶことにする。ステップ２は、各々を「オープンソース評価モデル」により評価する。ステップ３は、ステップ１αで換算した自社開発費と、ステップ２で評価した各々の評価点により、オープンソー

スの「公正な評価額 ＝ 取得原価」を算定する。

算定式は次の通りである。オープンソースを「A」、自社開発を「C」、Cの開発費（見積額）を「z」とする。商用ソフトウェア換算係数を「1/10」とする。「オープンソース評価モデル」による各々の評価点は、Aが「a」、Cが「c」とする。その場合、Aの「公正な評価額 ＝ 取得原価」は、$y=(1/10)(a/c)\ z$、となる。複数の場合の合算は（1）と同様である。

（1）の類似の商用ソフトウェアによる算定と同様にすれば、Aの「公正な評価額 ＝ 取得原価」は、$y=(a/c)\ z$、となるが、これでは（1）と比べて著しく高額となり、均衡を欠くことになる。そこで、筆者は一般的な商用ソフトウェアと自社開発との価格比(費用比)として、商用ソフトウェア換算というファクター（係数）を導入することが穏当であると考える。商用ソフトウェアの購入価格と、同様のソフトウェアを自社開発する場合の開発費との一般的な比率から、商用ソフトウェアの購入価格ならば幾らに相当するかの推定を行なうのである。なお、この換算に利用できる統計的な数値は、探し出すことができなかったので、業界の見聞から得た経験値を暫定的に使用する。両者の価格比(費用比)は概算値で「1/10」とし、これを商用ソフトウェア換算係数とするのである。統計的な数値が整備されれば、それを採用し、企業により異なる比率を採用する場合にはその旨を明記すれば差し支えない。以上のことを、モデル事例に数値を当てはめて、「公正な評価額 ＝ 取得原価」算定をより明確にする。

（3）オープンソースの「公正な評価額 ＝ 取得原価」算定のモデル事例

２つのモデル事例による算定を近似的な条件により比較可能とするため、一部オープンソースを利用した自社開発の場合とする、モデル事例を示す。まず、２つのモデルの共通条件を掲げる。

共通条件
・開発対象：業務アプリケーション・ソフトウェア（１つの独立的なシステム）
・開発費　：総額8,000万円
　　内訳：オープンソースの利用に掛かる費用1,000万円、それ以外の費用7,000万
・会計処理：8,000万円は資産要件を充足し全額資産計上する

（3-1）類似の商用ソフトウェアによるモデル事例

次に、このモデルの個別条件を掲げる。

個別条件
（1）類似の商用ソフトウェアに関して
・購入価格：2,000万円　　・利用に掛かる費用：400万円
（利用に掛かる費用は、購入に伴い附帯するベンダーのサポート・サービスが手厚かったため、オープンソースの場合(1,000万円)に比べ、低額で済んだものとする。）
（2）「オープンソース評価モデル」による評価点
・オープンソース：60点　　・商用ソフトウェア：75点

ステップ１は、類似の商用ソフトウェアの購入価格（見積額）を設定する。購入価格は2,000万円、利用に掛かる費用は400万円である。

ステップ２は、各々を「オープンソース評価モデル」により評価する。ここでは、先の「オープ

ンソース評価モデル」により評価した結果が、オープンソースは60点、商用ソフトウェアは75点であったとする。

ステップ３は、ステップ１とステップ２により、オープンソースの「公正な評価額 ＝ 取得原価」を算定する。算定式は、利用に掛かる費用を加味すると、次の比例式に展開できる。

（2,000万円＋400万円）：（y＋1,000万円）＝75点：60点

y＝（60点／75点）×（2,000万円＋400万円）－1,000万円＝920万円

従って、オープンソースの「公正な評価額 ＝ 取得原価」は 920 万円である。取得原価総額は、「公正な評価額 ＝ 取得原価」を加算すると、8,920万円（＝8,000万円＋920万円）となる。

商用ソフトウェアを利用して開発を行った場合には、開発費総額が 9,400 万円（＝7,000 万円＋2,000 万円＋400 万円）となるのに対して、オープンソースを利用する場合には、実際の支出額は8,000 万円であり、資産計上額は取得原価の 8,920 万円で会計処理を行なうことになる。評価点としては商用ソフトウェア比80％のオープンソースをコスト・メリットで採用したのだが、それは実際の支出額並びに資産計上額に反映されることになる。しかも、「将来の経済的便益」に貢献するソフトウェア資産の全体像が忠実に表現される。

（3-2）自社開発ソフトウェアによるモデル事例

同じように、このモデルの個別条件を掲げる。

個別条件

（1）新規自社開発の見積りに関して

・行数規模：300KLOC　　・生産性：1KLOC/人月（既定値）

・工数：300人月（＝300KLOC÷1KLOC/人月）　　・人月単価：100万円/人月

（2）「オープンソース評価モデル」による評価点

・オープンソース：60点　　・新規自社開発ソフトウェア：100点

ステップ１は、ソフトウェア測定によりオープンソースと同様のソフトウェアの自社開発費（見積額）を算定する。ソフトウェア測定による算定式は、開発費（見積額）＝見積工数×人月単価であり、見積工数＝見積規模／見積生産性である。従って、自社開発費は、30,000万円（＝300人月×100万円/人月）となる。

自社開発費用の見積りには、ソースコードを公開しているオープンソースと比較するので、ソフトウェアの規模としては行数が適している。なお、そもそも当該企業のスキルでオープンソースと同等の開発が可能かという問題があり得るが、それに関しては技術力の高いソフトウェア企業への委託開発で対処するものとする。

ステップ１αは、自社開発費（見積額）を「商用ソフトウェア換算係数」を使って商用ソフトウェアの市場価格に換算する。商用ソフトウェア換算係数は「1/10」とするので、自社開発費の商用ソフトウェア換算値は、3,000万円（＝30,000万円×(1/10)）となる。

ステップ２は、各々を「オープンソース評価モデル」により評価する。ここでは、先の「オープ

ンソース評価モデル」により評価した結果が、オープンソースは 60 点、自社開発ソフトウェアは100 点であったとする。自社開発は相当の費用を投じ、システム要件全てを充足するように計画し遂行するのであるから、満点とする。但し、予算等の制約条件があれば、減点する。

ステップ３は、ステップ１αとステップ２により、オープンソースの「公正な評価額 ＝ 取得原価」を算定する。算定式は、それぞれの費用の条件を加味すると、次の比例式が展開できる。商用ソフトウェアの利用に掛かる費用に相当するものは、自社開発では全体的に遂行する作業の中に含まれるので、開発費に織り込み済みであり、ゼロとする。

（3,000 万円＋0 万円）：（y＋1,000 万円）＝100 点：60 点

y＝（60 点／100 点）×（3,000 万円＋0 万円）－1,000 万円＝800 万円

従って、オープンソースの「公正な評価額 ＝ 取得原価」は 800 万円である。取得原価総額は、「公正な評価額 ＝ 取得原価」を加算すると、8,800 万円（＝8,000 万円＋800 万円）となる。

自社開発を行った場合には、開発費総額が 37,000 万円（＝7,000 万円＋30,000 万円）となるのに対して、オープンソースを利用する場合には、実際の支出額は 8,000 万円であり、資産計上額は取得原価の 8,800 万円で会計処理を行なうことになる。評価点としては自社開発比 60%のオープンソースをコスト・メリットで採用したのだが、それは実際の支出額並びに資産計上額に反映されることになる。しかも、「将来の経済的便益」に貢献するソフトウェア資産の全体像が忠実に表現される。

最後に、この会計処理のコスト／ベネフィットに言及しておきたい。コストに関しては、比較的些少である。ソフトウェア企画でオープンソースを選定する際に行なっていることを、定型的なモデルで行なうこと、自社開発ソフトウェアとの比較評価のために「商用ソフトウェア換算係数」を情報収集・蓄積することに新たなコストを要するくらいである。ベネフィットに関しては、目的適合性を充足し、忠実な表現となる情報開示をすることにより、利害関係者との関係改善ないし良好な関係構築に貢献することになる。そういう意味では、コスト／ベネフィットが過大な負担ないし著しい不均衡となることはないと考える。そして、オープンソースの「公正な評価額 ＝ 取得原価」算定モデルに基づく算定により、現行制度でオフバランスされているオープンソースをオンバランス化できるようになり、その意義は少なくないと考える。オープンソースを利用することは、開発費（又は構築費）を削減することはもとより、世界の叡智の結集とも言える先進的な技術を果敢に取り入れることであり、またそうするには高度な技術力を必要とするのであるから、企業のソフトウェアに関する積極的な取り組み姿勢並びに体得している技術力の水準を、オンバランス化することにより開示することになる。それは、外部利害関係者の意思決定への有用な情報開示となると言える。

これまで捕捉してこなかった無償のオープンソースを、「公正な評価額を取得原価とする」ことにより捕捉可能とする会計処理案を提示した。会計制度的な根拠としては、「企業会計原則」の無償取得に関する規定を範囲拡大し適用することを試みた。また、ＩＡＳＢの概念フレームワークにおけ

る資産の定義・認識・測定を取り上げ、それに依拠することで、無償取得の資産計上が可能となることを説示した（末尾参照）。そして、残された課題を、「公正な評価額」を「信頼性をもって測定できる」かどうかということに絞り込んだ。それに対しては、ゴールデンとラタンゼのモデルを典拠とし、筆者なりに改変した「オープンソース評価モデル」と、それをベースとする「公正な評価額 ＝ 取得原価」算定モデルにより、「信頼性をもって測定できる」ことを提示した。具体的には、類似の商用ソフトウェアがある場合と、それがない新規自社開発ソフトウェアによる場合の算定モデルに関して、各々具体的な数値事例を挙示し、実用性が十分にあることをも確認した。

最後に、測定属性について、本章の取得原価→資産計上という論理構成に対して、むしろ公正価値→資産計上という論理構成の方が適切ではないか、という指摘があり得る。それを取得原価としたのは、次の理由からである。第１に、オープンソースが単独で利用される場合もあるが（例えば開発ツール等）、自社開発のある纏まったシステムの１部のコンポーネントとして組み込まれる場合（商用ソフトウェアを含めて）の方がむしろ多いと言える。独立的な場合はともかく、コンポーネントとして組み込まれる場合には、その纏まったシステムとして会計処理の対象になるのであるから、そのうちの自社開発分及び商用ソフトウェアは取得原価で取り扱い、オープンソースは公正価値で取り扱うのは、異なる測定属性による測定額が混在することになり、ソフトウェア資産としての総額の精度が大きく減じることになるのではないか、と考えたのである。第２に、本論は総じて現行のソフトウェア会計基準に多くの異議を呈し、代替案を提示しているが、測定属性に関しては現行基準と同様に取得原価を採用し、それに統一している。その上で、可能な限りの代替案を提示することを企図している。従って、オープンソースだけを公正価値で取り扱うのは、統一性を損なうのではないか、と考えたのである。

そして、熟慮すると、取得原価を基礎としたオープンソースのオンバランス化の論拠として、ＩＡＳＢの概念フレームワークにおける資産の定義並びに認識の規定に依拠すると、企業会計原則とは異質なものを混在させているかもしれないが、企業会計原則の無償取得の規定だけでは論拠としては弱く、他に同質的な依拠できるものは探索し得なかったので致し方ないと考えている。また、取得原価又は公正価値の採否は、本章に留まらず、仕損会計等にも波及することであるので、今回の体系的な集成は取得原価ベースのままとし、今後、オンバランス化の同質的な論拠を含めて、異なる論理構成において主題的に考究することとしたい。

なお、ＩＡＳＢの概念フレームワークに関しては、本章で参照している議論以降の動きがあるが、それも現時点では正式決定には到っていない。従って、本章の議論での参照内容は少々古いものにはなっているが、直近動向によって論旨に影響するわけではないので、そのままとした。正式決定したならば、機会を見て、その内容を反映させるべく、改訂を行ないたいと考えている。

本論

第６章　ソフトウェア仕損

１．ソフトウェア開発プロジェクトの失敗の概観

（1）ケーパーズ・ジョーンズの『ソフトウェアの成功と失敗』

（2）ＪＵＡＳ「企業ＩＴ動向調査」における３大指標の調査

２．ソフトウェアの仕損に関する予備的考察

（1）仕損の予備的考察

（2）仕損捕捉の会計的構想

３．ソフトウェアの仕損に係る適正な会計処理

（1）定義並びに適用対象

（2）ソフトウェアの仕損の兆候の識別

（3）ソフトウェアの仕損の認識

（4）ソフトウェアの仕損の測定

図表１−６−1　ソフトウェア仕損の認識並びに測定のモデル

図表１−６−2　ソフトウェア仕損に関わる生産性、品質の範囲区分

図表１−６−3　ケース別仕損試算表

第６章　ソフトウェア仕損

ソフトウェア開発に関しては、ＰＭＢＯＫ（Project Management Body of Knowledge：プロジェクトマネジメント知識体系）を始めとするプロジェクト管理のモデルや方法、開発方式や開発技法が数多く提唱されてきており、実際のプロジェクトでも適用されている。それでも、プロジェクトの失敗は後を絶たない。プロジェクトの失敗はなくならないどころか、顕著に減少してもいない[323]。

ところが、プロジェクトの失敗は一般的に知られることはほとんどない。マスメディアで報じられるシステム・トラブルの多くは、情報漏洩やサイト攻撃、あるいは金融機関又は取引所などのシステム・ダウンであり、運用トラブルである。しかし、運用トラブルはあくまで問題が表面化した「氷山の一角」であり、且つ運用における直接的な不具合による場合もあるが、問題の大半はそれ以前のソフトウェア開発に基因しているのである。開発におけるプロジェクトの失敗が潜在的なリスクを作り込んでおり、それが顕在化すると運用トラブルとなって表面化すると言える。そして、経営戦略ないし事業活動を推進する上でのＩＴ（中核はソフトウェア）の重要性を考えれば、システム・リスクは事業リスクの大きな部分を占めている。そのリスクを知り得るために、ソフトウェア開発の成否に関わる情報が必要である。

まだ記憶に新しいみずほ銀行のシステム・トラブルに関説して、「はっきりいって、みずほ銀行を批判できる企業はそう多くはない。多くの企業の情報システムもいつ問題が起きるか分からない。トラブルの内容によっては、みずほ銀行の例など比較にならないほどの実害が発生するリスクがある」[324]と言われているが、筆者も同感である[325]。こうしたリスクを開示しない財務情報は、今日的には明らかに情報の不備・不足と言わざるを得ないと考える。

財務報告でソフトウェア開発プロジェクトの失敗が情報開示されることはこれまでほぼ無いと言える。原価計算においても、後述するように、ソフトウェア開発における仕損費の把握は困難として、的確に捕捉しないまま放置されてきた。それに加えて、現行の会計制度では、支出が伴えば、

323　『日経コンピュータ』における第２回システム開発プロジェクトの実態調査(2008 年実施)に拠れば、プロジェクト「成功」率は 31.1%(有効回答 814 社)であり、第１回調査(2003 年実施)より 4.4%の上昇とのことである(日経コンピュータ(2008)p.38)。なお、同調査の「成功」は、「品質、コスト、納期の３基準すべてで「当初計画通りの成果」を収めたプロジェクトだけを「成功」と定義した」独特の規準による。その理由は、「確かにプロジェクトの成否をどこで判断するかは難しい問題だが、本誌ではこの３基準を満たしていなければ確実に成功とは言い切れないと考え」た、と説明している(p.38)。同調査での「成功」の「逆数」が即「失敗」とは言えないが、それでも成功率からすれば、「失敗」が少なくないことは察せられよう。５年間の推移も、同誌が認めている通り、顕著な好転とは言えない。

324　日経コンピュータ編(2011)pp.3-4

325　具体的な根拠は、今日の大規模システムは行数規模で１億行を超えるが(日経コンピュータ編(2011)p.183)、バグ残存率を 0.02/KLOC と推定すると(一般的には良好な品質である)、2,000 個(=0.02/KLOC×100,000KLOC)超のバグが潜在していることになる。ごく軽微なものからシステム・ダウン等に到る重大なバグまでが散在しているとして、開発時に作り込まれたトラブル原因に限っても、これだけのものを抱えて日々運用している。こういったリスクは周知されなければならない。

開発に掛かったコスト[326]は、費用あるいは資産として会計処理されている。つまり、仕損を仕損処理せずに資産ないし費用として処理することを容認してきたのである。これでは、利害関係者がプロジェクトの失敗を知り得る術はないと言わざるを得ない。それ故に、企業活動の写像である財務会計は、ソフトウェア開発プロジェクトの失敗を適正に捕捉し、財務報告で開示させるべきではないのか。これが、筆者の問題提起である[327]。

本章では、ソフトウェア開発プロジェクトの失敗を会計的に捉えるために、ソフトウェア開発における仕損に関して、その兆候を識別し、続いて認識並びに測定を行なう会計処理案を提示する。それにより、これまで「正常な事象」と区分されることなく資産ないし費用として処理されてきたソフトウェア開発の仕損費を適正に捕捉するべきことを提案する。

本章の構成は、次の通りとする。まず、ソフトウェア開発プロジェクトの失敗とは如何なる事態なのか、会計的な考察の前提として、必要な範囲で説明する。次に、予備的考察として、ソフトウェアの仕損に関する先行研究を瞥見し、それらの取り扱いではソフトウェアの仕損を捕捉することが困難である点を明らかにする。続いて、それを踏まえ、減損会計に準じた会計的な構想を提示する。そして、本章の主要部分である「ソフトウェアの仕損に係る適正な会計処理」では、仕損の兆候の識別、認識並びに測定をどのように行なうかを提示する。併せて理解に供するために、試算例を掲示する。また、検証可能性並びに仕損会計処理を実施する際のコスト／ベネフィットに言及する。

１．ソフトウェア開発プロジェクトの失敗の概観

まず、本章が取り上げるソフトウェア開発プロジェクトの失敗という事態を概観することから始める。ソフトウェア業界で遍く共通認識となっていると言えるのは、プロジェクトの失敗とは、プロジェクト管理の３大指標である「ＱＣＤ」（品質(Quality)、コスト(Cost)、納期又は開発期間(Delivery)）が計画より大幅に悪化した場合である。つまり、品質が劣悪か、又はコストが大幅に超過したか、又は納期が遅延(開発期間が延長)したか、ということである。具体的な事象は、ケーパーズ・ジョーンズ(Capers Jones)の『ソフトウェア病理学』などに具に書かれている。また、エドワード・ヨードン(Edward Yourdon)の『デスマーチ　第２版』では、そうした悲惨な失敗プロジェクトという事態を「デスマーチ」（死の行進）という刺激的な名辞で形容し、それを冠した一書を物している。高名なソフトウェア工学者であり、1960 年代の「ソフトウェア危機」に際して開発技法で対処するために考案された構造化技法の考案者の１人であり、ソフトウェア業界を知悉したヨ

[326] 本章で用いる「コスト(cost)」という用語はすべて、費用あるいは資産に認識される以前の、発生現象を意味する。

[327] みずほ銀行のような事態に関しては、情報開示という観点から別途本格的に取り上げたいと考えているが、本章では仕損会計という観点からアプローチしようとしている。

ードンは、次のように言明している。「私の結論は、デスマーチ・プロジェクトは、常態であって、例外ではないというものだ。ソフトウェア開発者もプロジェクト・マネジャーも十分賢明で、合理的な方法でプロジェクトを管理したいはずだし、ユーザーや上層部も一世代前よりはるかにコンピュータに明るく、限りある資源で開発できるソフトウェアがどの程度のものかもわかってきたと思う。しかし、開発側もユーザー側も賢明になっても、デスマーチ・プロジェクトの発生はとめられない。ビジネス上の競争激化による圧力や、新技術の登場による競争激化が駆り立てるからだ」[328]。

なお、ソフトウェア開発プロジェクトの失敗に関する書籍は夥しい数出版されており、様々な「処方箋」が提示されているが、いずれも実効性に欠け、開発現場では今も「悪戦苦闘」を続けている。そうした観点からソフトウェアの歴史を捉えることは興味深いテーマではあるが、仕損会計の考察に必要最低限の概観に留めることにする。

（1） ケーパーズ・ジョーンズの『ソフトウェアの成功と失敗』

ケーパーズ・ジョーンズの『ソフトウェアの成功と失敗』では、プロジェクトの失敗を「絶対的失敗」と「相対的失敗」に大別し、「絶対的失敗とは、プロジェクトがその完成の前に中止される場合と定義」し、「相対的失敗とは、プロジェクトがその予定されたスケジュールあるいはコスト目標と比較して大きく超過した場合と定義」[329]している。そして「相対的失敗」の例として、「ソフトウェアプロジェクトの引渡しは行われたが、その予定コストおよびスケジュールにおいて 50%以上の超過を生じた場合」と、「ソフトウェアプロジェクトを引き渡した後、品質レベルが悪く、顧客の全面的利用が６カ月以上も遅れた場合」を挙げている[330]。

「絶対的失敗」は、プロジェクトの中止という外見的にも明白な事態なので定義は明瞭であるが、「相対的な失敗」は「大きく超過した」と大まかな定義に留まっている。例示では、予定に対し「50%以上の超過を生じた場合」や、予定より「６カ月以上も遅れた場合」を挙げている。実際には「成功」と「失敗」の中間に、「成功」とは言えず、「失敗」とも言えないプロジェクトが多くあるが、敢えて対比的に取り上げて[331]、際立たせ、問題を浮き彫りにしていると言えようか。

なお、ジョーンズは「相対的失敗」に関して、定義にはスケジュールとコストしか内包させていないが、例示では図らずも品質がトリガーとなった事態を挙げている。これも定義に含めたほうがよいであろう。ソフトウェア開発のプロジェクト管理で必ず取り上げられる３大指標だからである。すなわち、ソフトウェア開発プロジェクトの失敗とは、差し当たり、予定に対する大幅な、①納期遅延、②品質不良、③コスト超過である、と確認しておく。なお、OR 条件であり、いずれか１つでも失敗と見做すのが妥当である。

---

328 Yourdon(2004)邦訳 pp. viii-ix(傍点邦訳)
329 Jones(1995)邦訳 p. 4
330 同書邦訳 p. 4
331 同書邦訳 pp. 15-16 等

（2）ＪＵＡＳ「企業ＩＴ動向調査」における３大指標の調査

ＪＵＡＳの「企業ＩＴ動向調査」には、３大指標に関する調査が含まれている[332]。システム開発工期[333]に関しては、「予定通り完了する」、「ある程度は予定通り完了する」、「予定より遅延する」、「該当プロジェクトなし」のいずれかをプロジェクト規模（工数規模）毎に選択回答するように求めている。結果は、「予定より遅延する」が 100 人月未満のプロジェクト（回答数 n=649）では 22%、100～500 人月未満（n=341）では 32%、500 人月以上（n=199）では 42%である。規模が増大すると、遅延比率が増大する傾向を示している。

品質に関しては、「満足」、「ある程度は満足」、「不満」、「該当プロジェクトなし」のいずれかをプロジェクト規模毎に選択回答するように求めている。回答結果は、「不満」が 100 人月未満（n=651）では 13%、100～500 人月未満（n=341）では 22%、500 人月以上（n=199）では 31%である。規模が増大すると、やはり「不満」の比率が増大する傾向であることを示している。なお、品質は同調査では広義の品質を意味し、「満足」「不満」という定性的な回答を求めているが、失敗や仕損を定量的に把握し、会計処理に繋げていくには、採用し難い指標である。また、工期やコストが予定対比での回答を求めているのに対して、品質はそれ自体での評価を求めているが、失敗や仕損を把握する際には統一的でなければならないだろう。

コストに関しては、「予定通り完了する」、「ある程度は予定通り完了する」、「予定より超過する」、「該当プロジェクトなし」のいずれかをプロジェクト規模ごとに選択する。回答結果は、「予定より超過する」が 100 人月未満（n=648）では 15%、100～500 人月未満（n=342）では 29%、500 人月以上（n=200）では 40%である。コストの規模との対応傾向も、工期・品質と同様であり、いずれも開発規模が大きいと、それだけ失敗する傾向が高くなることを示している。

なお、同調査では、工期の「予定通り」と「遅延」の間に「ある程度は予定通り」という選択肢があり、工期に関する傾向性を３つの離散的な事態に分別して捉えようとしている。つまりは、調査の趣旨は失敗か否かという指標で捉えるのではないが、「予定通り」と「ある程度は予定通り」という選択肢があることからすれば、それら以外の「予定より遅延する」は失敗と見做しても差し支えないであろう。だが、「予定より遅延する」に関してそれ以上の調査項目はないので、予定よりどの程度の遅延かは不明である。品質とコストに関しても、同様である。傾向を知る意味では得ることの多い調査であるが、定量的なアプローチに関しては別途考えざるを得ないであろう。

２．ソフトウェアの仕損に関する予備的考察

---

[332] ＪＵＡＳ(2011)pp. 188-190

[333] 「納期」「工期」「開発期間」「スケジュール」を、引用との関連や文脈によって混用しているが、これらは同義的である。納期はどちらかと言えばソフトウェア企業が受託の立場で使うことが多く、工期はユーザ企業が自社の立場で使うことが多いが、ユーザ企業でもユーザ(部門)に対しては納期という場合もある。

（1）仕損の予備的考察

仕損は、原価計算基準の第一章五「非原価項目」及び第二章二七「仕損および減損の処理」並びに三五「仕損の計算および処理」に拠れば、正常仕損と異常仕損に分けられ、正常仕損は原価項目、異常仕損は非原価項目とされている。

太田昭和監査法人(1992)は、仕損費に関して、次のように扱うことを提唱している。「ソフトウェア開発がうまく行かないために支出した、再作成のための費用あるいは修正のための費用は仕損費となる。一般に正常な仕損費については原価に算入し、異常な仕損費は特別損失に計上するものとされている。ソフトウェア開発においては、この仕損費を把握することが難しく、仕損を正常と異常とに区分することは、さらに困難である。なぜならソフトウェア開発においては、つねに何んらかの修正・手戻りが発生するものである。これを仕損として認識してよいか問題があるからである。したがって、修正が大幅であって金額的にも多額になる場合を除いて、すべてソフトウェア開発費に含ませることが実務的である」[334]。補足すると、ソフトウェア開発では正規に必須の工程としてテスト工程を組み入れており、それは間違いなどの発生を当然視してのことであり、「修正・手戻りが発生する」ことは直ちに仕損ではないということである。この点に関しては、筆者も同感である。しかし、「修正が大幅」や「金額的にも多額になる場合」というだけで、その具体的な「規準」が示されていない曖昧な扱いに留まっており、事実上は仕損を扱わないに等しいものである。更に、大幅な修正に多額の費用が掛かる場合以外は、「すべてソフトウェア開発費に含ませる」という処理を容認していることと相俟って、プロジェクトの失敗は会計において捕捉されずにきたのであろう。

櫻井通晴は、「仕損費の計算および処理」に関して、次のように述べている。「仕損は、原則として、次の２つのケースから生じる。第１は、設計上のミス、コーディング・ミスなどで不完全なソフトウェアとして仕上がったが、手なおしによって完成品に回復できるケースである。ソフトウェアは工業生産物とは違って試行錯誤的に手なおしすることが多いため、この意味での仕損費の把握は困難である。第２は、手なおしをしようにも回復できないため、代品を製作するケースである」[335]。第１のケースに関しては「「原価計算基準」の 35 の(1)に準じて、当該ソフトウェアの仕損費（直接費扱）として処理し、製造原価に加算するのが妥当であろう」[336]とし、第２のケースに関しては「「原価計算基準」の 35 の(2)の１に準じて、旧製作品指図書に集計された原価を仕損費とし、新しい製作品指図書にたいして直接費として賦課すべきである」とする[337]。しかしながら、第１のケースは一方では「仕損費の把握は困難である」と言いながら、他方ではバグをすべて仕損と見做すに等しいものである。「試行錯誤的に手なおしすることが多い」ことに関連した、テスト工程を必須に組み

334 太田昭和監査法人(1992)p. 58(関連箇所 p. 57)
335 櫻井(1992)p. 75
336 同書 p. 75
337 同書 p. 75

入れていることを、十分に配慮してはいないように思える。第２のケースに関しては、事象としては明解であり、仕損の把握は容易であるから、捉え方は妥当と言えるが、原価計算ないし会計処理としては新旧の「製作品指図書」といった「工業生産物」に準じた扱いとするのは、そもそも開発プロジェクトで多くのコンポーネント[338]を開発する際に個々のコンポーネント単位に「製作品指図書」を発行するわけではないから、実務的な取り扱いとしては少々難点がある。

また、上記２つのケースは正常仕損に関することだが、異常仕損に関しては、「異常な仕損費は、非原価であるから、営業外費用か特別損失として処理すべきである。ただ、災害などのような質的な異常性が原因であれば判定が容易であるが、量的な異常性の判定は「重要性の原則」により、個々に判断されなければならない」[339]としながらも、「ソフトウェア開発においては工業生産物とは違い、異常工数の発生は通常の作業のなかで吸収されてしまうことが多い。また、どこからが生産活動でどこまでが研究開発であるかの設定もむずかしい[340]。そのため、現実には異常仕損費の把握がきわめて困難である」[341]として、結局は「困難」を理由にして具体的な処理方法などを提示していない。

そして、「しかし、効果的な原価管理のためには、異常工数の把握は必須である。これの効果的運用を図るためには、適正な標準工数を算定し、標準原価計算制度をもつことが望まれる」[342]と、異常仕損の把握を将来的な標準原価計算の実施に託すに留まっている。しかしながら、櫻井の構想する標準原価計算は工程別（フェーズ別）という粒度なので[343]、例え実施したとしても、粒度が粗過ぎて、櫻井の期待に反し、それによる異常仕損の把握は「困難である」。代替案を提示する際にも触れるが、ソフトウェア開発は余程小規模な開発でない限り、同一工程において複数の要員が作業をし、要員間にスキルの差が少なからずあり、また個々の機能や方式等で作業の難易度が異なるので、仕損の発生可能性は一律ではないのである。ところが、フェーズという粒度では、差異が平均化されてしまい、把握が困難になる。もう１つは、バグはテスト工程で表面化することが多いが、テストに問題がある場合もあるが、設計工程ないしプログラム製造工程でバグを「作り込む」場合が圧倒的に多いのである。バグの「作り込み」箇所を特定しなければ、的確な把握は困難である[344]。

---

338 コンポーネントは、この文脈では、広義のプログラム(群)ないし機能(群)を意味している。

339 櫻井(1992)p. 75

340 櫻井は、別の箇所では「ソフトウェアの製作は必ずしもすべてが研究開発活動のようにリスキーなものばかりではなく」（同書 p. 41）と言いながら、「どこまでが研究開発であるかの設定もむずかしい」という指摘に留まっている。また、この文脈で「研究開発」を持ち出すことは不適切ではないだろうか。そもそも研究開発であれば、「試行錯誤」の繰り返しと言える性格上、仕損は発生しない、あるいは仕損とは見做さないのではないだろうか。なお、研究開発と非研究開発の区別に関しては、本論第２章で詳細に取り上げているので、ここでは立ち入らない。

341 櫻井(1992)p. 76

342 同書 p. 76

343 同書 pp. 83-92

344 これに関して誉田(2010)では、ＮＥＣが考案し、実施しているソフトウェアの品質管理の手法である「ソフトウェア品質会計」を取り上げている（なお、現在は「ＳＩ管理会計」という発展形で実施されている）。目標管理において、数値的な計測に留まらず、バグの「作り込み工程」と「摘出工程」の両面で捉えて管理している。摘出は一般的にも捕捉しているが、「作り込み工程」を捕捉す

これらの予備的考察で明らかになったことは、次の通りである。ソフトウェア開発はテスト工程を必須の工程として組み込んだものであり、逐一のバグを仕損と見做すことはソフトウェア開発の特性を無視しており、適合的ではない。また、把握が困難とされる由因は把握の粒度にあり、フェーズという粒度での把握では不十分で、更に精細な粒度で捉える必要がある、ということである。

（2）仕損捕捉の会計的構想

これに対して、ソフトウェアの仕損を捕捉すべく会計処理案を提示したい。ソフトウェア開発プロジェクトの失敗が後を絶たず、ほぼ恒久的に今後もなくならないと考えており、仕損の処理が必要不可欠だからである。基本的な考え方を述べ、筆者案（以下、「仕損会計」と称す）のアウトラインを明らかにする。

第１に、仕損会計は、減損会計と類似的な性格のものとし、基準構成を同様にする。減損会計とは、ＩＡＳ（International Accounting Standards ）36 の「資産の減損（Impairment of Assets)」（以下、IAS36 と略記）を念頭に置いており、それを参考にして構想を具体化する。基本的な考え方は、減損会計と類似的に、資産の回収可能価額を帳簿価額に付することである。仕損会計は当初の認識時点における回収可能性の評価として、回収対象となる原価か否かを仕分け、回収不能部分を除くものであるのに対して､減損会計はその延長上の事後的な評価として位置付けるものである。回収可能性を評価し、不能分を帳簿価額から減じることは共通的であり、両者は連携関係を持つ一連の流れとなる。ただし、多少の相違点がある。１つは、両者は認識時点の違いから対象が多少異なる。減損会計の対象は既に資産認識された資産であるが、仕損会計の対象は、資産あるいは費用として認識される以前のコストとするので、費用として認識されるものを除外しない。費用処理される場合にも、開発に投じた費用を回収することに変わりはないからである。もう１つは、兆候はそれまでの企業努力（ソフトウェア開発を含む）の結果を含めて対象を当初（プロジェクト終了時点）ないし毎期末という時点で識別する点では同様だが、減損会計の場合には内外の諸要因が影響を及ぼすのに対して、仕損会計の場合には専ら内部要因（開発プロジェクトの成否）によることである。基準構成は、共通的に、第１に仕損の兆候を識別する、第２に仕損の認識を行なう、第３に仕損の測定を行なう、という３ステップを踏むものとする。

第２に、考え方以外に、減損会計と異なる点の１つは、減損会計では回収可能価額（Recoverable amount）を正味売却価額（Fair value less costs to sell(売却費控除後の公正価値)）と使用価値（Value in use）のいずれかとするが、ソフトウェアの仕損会計では正味売却価額は選択肢としない。理由は、売却か継続的使用かの判断は、事後的な評価においては同時期に起こり得るのであるが、当初の認識時点では使用（あるいは販売）することを前提に認識するからである。当初は、そ

る点は適切な着眼である。仕損に即応して言えば、「摘出工程」で仕損が発覚し、その工程で改修などの対応をするわけであるが、仕損を真に発生させたのは「作り込み工程」である。それを特定することで、仕損の測定は的確になるのである。

もそもの開発あるいは導入の意思決定がなされ、プロジェクトが遂行されるので、それは既に使用（あるいは販売）の意思決定が行なわれたことに他ならない。従って、回収可能性を示す価値は使用価値のみとする。

第３に、回収可能性を示す使用価値としては、基本的には、当該ソフトウェアの継続的使用により生ずるであろう将来キャッシュインフローの獲得のために、又は当該ソフトウェアの使用を可能とするために、必要なキャッシュアウトフローであり、当該ソフトウェアに直接帰属させることができ、又は合理的かつ首尾一貫した基礎により配分できる見積り[345]が妥当であると考える。ソフトウェアが資産認識されることを考慮すると、あくまで現行の取得原価の算定の修正であると位置付けているので、取得原価で認識される他の資産とも整合的であるからである。

第４に、減損会計では対象となる資産を、１つの資産全体あるいは資産グループ全体の単位で取り扱うが、ソフトウェアの仕損会計では、ソフトウェアの仕損の兆候の識別には１つのソフトウェア開発プロジェクト全体の単位で取り扱うのに対し、仕損の認識並びに測定には１つのソフトウェア（開発）を細分化した個々のタスク（プロセス）並びにモジュール（プロダクト）の単位で取り扱うことにする。そして、この単位を、規模、開発期間、品質、コストの項目で捉えるのである。

ソフトウェアを捉える場合には、プロセスとプロダクトの両面から総合的に捕捉しなければならない[346]。それに基づいてソフトウェアの仕損を捉える限り、プロセスの側面も捉えるのでなければならない。何よりも、ソフトウェアに係る費用の大半は開発作業に従事した開発要員の人件費であり、開発プロセスをどのように遂行したかに大きく左右されるからである。仮にプロダクトには問題がなく仕損がなくとも、その開発を非効率に行なったとすれば、ソフトウェア開発費は不必要に多額になってしまうのである。

そして、プロセスに関する仕損を捕捉可能とするためには、より精細な粒度であるタスク・ベースでの捕捉が必要であることを強調したい。開発実務では、粒度や厳格度は様々であるが、プロジェクト管理において作業の定量的な測定はほぼ必ず行なわれている、と言える。また、開発の作業区分は様々あるが、一般的にはフェーズ＞アクティビティ＞タスクという粒度で区分することになっている。順に作業粒度が細かくなることを示し、タスク・ベースが最小の作業単位となる。開発原価の見積りや実績の測定において、フェーズ・ベースで区分して捕捉することはごく当たり前に行われており、櫻井もこの粒度での標準原価計算を提唱している。それに対し、ケーパーズ・ジョーンズは、「アクテビィティベース」という捉え方を、見積り、更には実績測定をも包含したものとして提起している[347]。これらを踏まえて、更に精細な粒度であるタスク・ベースでの捕捉が必要である、と筆者は考える。開発原価の見積りや実績の測定で使われることは余りないが、開発プロセスにおける実際の作業はこの粒度で行なわれているので、開発実務では馴染みのあるものである。

---

[345] IAS36par.39「Composition of estimate of future cash flows」の(b)の規定に依拠している。
[346] 序論で言及し、本論第２章で取り上げている捉え方である(Tidd, Joe(1997)等参照)。
[347] Jones(2007)邦訳 pp.287-436、Jones(2008)

なお、プロダクトに関するモジュールは既述のコンポーネントの最小単位であり、プロセスと同様に精細な粒度で捉える趣旨である。

３．ソフトウェアの仕損に係る適正な会計処理

（1）定義並びに適用対象

まず、ソフトウェアに対して仕損処理を適用するにあたって用いる用語に関する定義を行なう。

（ａ）ソフトウェア開発プロジェクトの失敗とは、プロジェクト管理の３大指標である品質(Quality)、コスト(Cost)、納期又は開発期間(Delivery)（ＱＣＤ）が予定より「大きく超過した場合」である。これを大別すると、プロジェクトを中止・中断する場合と、ともかく継続して完了する場合とに分けられる。

（ｂ）ソフトウェアの仕損とは、１つはプロジェクトを中止・中断した場合であり、それまでに投じた費用のすべてが仕損費である。もう１つは、継続して完了した場合のプロダクト（成果物）が粗悪品、不良品であるか、又は、プロセス（作業）が非効率で余計な工数を費消し、引いては無駄なコストに帰結するような現象である。

（ｃ）計画比規準とは、ソフトウェアの仕損の兆候を識別する規準である。

（ｄ）過去平均実績緩衝規準とは、ソフトウェア仕損の認識並びに測定を行なう規準である。

（ｅ）異常仕損とは、ソフトウェアの仕損のうち、非原価項目として費用処理するものである。

（ｆ）特別仕損とは、ソフトウェアの仕損のうち、異常仕損より更に仕損度合いが甚大な特別損失として取り扱うものである。

（ａ）と（ｂ）の関連性は、本章では、ソフトウェアの仕損に関して、主としてソフトウェア開発プロジェクトの失敗並びにその具体的な現象とするが、厳密には相即的なものではない。失敗プロジェクトにおいても、仕損と仕損でないことがあり、精査しなければならない。逆に、開発は失敗でなく、成功裡に完了したプロジェクトでも、仕損が発生していた可能性はある。従って、失敗プロジェクトのみならず、プロジェクト全般を対象として仕損を取り扱うことが本来的である。そうでありながら、何故プロジェクトの失敗に照準を定めるかと言えば、筆者は、２段階で仕損処理の適用を構想しているからである。第１段階では、プロジェクトの失敗に照準を定め、そこにおける仕損を適切に捕捉する処理を確立し、会計慣行としての定着化を図る。続いて第２段階では、仕損処理をソフトウェア開発一般にも適用拡大する、ということである。２段階構想とする理由は、ソフトウェア開発実務並びにその会計実務の現状を考えれば、仕損の認識や測定を実施可能とする管理粒度を装備し、処理コストを負担することはどの企業でも容易に対応できるとは限らないと考えるからである。それに対し、プロジェクトの失敗という識別は容易であり、失敗を招来した経営責任において、仕損測定の負担を負うことは当然の責務であると考える。

（ｃ）と（ｄ）については、兆候の識別又は認識並びに測定の該当箇所で説明する。

（ｅ）と（ｆ）について、これらを設定した前提となる捉え方をまず説明しておきたい。原価計算基準における正常仕損は、ソフトウェアに当て嵌めるのは不適格だということである。ソフトウェア開発は、非定型的な「知識労働」であり、一般的な検査とは明らかに異なった相当程度の期間と工数を要するテスト工程を組み込んでいるのである。そして、成果物であるソフトウェアは完成品であってもバグ（不具合）がほぼ必ず潜在している。従って、軽微な正常仕損を捕捉することは難しいだけではなく、こうした特性から無用である、と考える。よって、後述するように一定の許容範囲を設定した範囲内にあるので仕損とは見做さず、ソフトウェア開発費に含めて会計処理するので、結果として原価計算基準に準じた取り扱いとなる。それ故、（ｅ）と（ｆ）を捕捉対象とする。（ｅ）と（ｆ）を区別するのは、甚大な仕損度合いである（ｆ）は開発時点の失敗に留まらず、運用に入っても障害などが発生する可能性が大きく、それが予め高い確率で予想できるので、それに見合った処置を施しておくことが相当である、と考える。

次に、適用対象は、研究開発目的以外のソフトウェア（開発）とする。日本の現行のソフトウェアに係る会計基準の区分で言えば、市場販売目的、自社利用、受注制作のいずれにも一律に適用するものとする。何故ならば、制作目的によって、ソフトウェア開発の仕方が異なることは基本的にないからである。開発の進め方、開発プロジェクトの編成など、いずれも大差ないのである。また、技術的な難易度は、個々には当然に差異はあるが、制作目的によって差異があるわけではない。更に、市場販売目的または自社利用で、自社開発といっても、自社の社員だけで開発することはほとんどなく、大なり小なり外部委託していることも変わりない。契約形式上は請負・準委任・派遣・出向という違いはあるとしても、あくまでソフトウェア開発の実態としては基本的に違いはないのである。むしろ、開発の工数や費用が増大した場合に、委託ー受託の契約内容如何によっていずれが負担するかは異なることが、一律に適用する際の留意事項である。追加発生費用などをすべて受託側が負担する場合には、委託側では仕損費が発生しないこともあり得るからである。なお、制作目的によって個別原価計算あるいは総合原価計算と異なる適用をした場合にも、異常仕損と特別仕損の捉え方は一律であり、変わるものではない。

また、ＩＡＳ38「無形資産（Intangible Assets)」に照らして言えば、外部取得（購入）と自己創設（自社開発）という取得形態による区分のうち、自己創設に適用するものである。

研究開発目的を対象外とするのは、そもそも研究開発の試行錯誤を仕損と見做すことは、研究開発の性格に照らして不適切であると考えるからである[348]。

（2）ソフトウェアの仕損の兆候の識別

[348] これに関連して、筆者はアメリカの会計基準も含めて現行のソフトウェア会計基準が研究開発目的のみならず、それ以外のソフトウェア開発をもある範囲研究開発と見做していることに、根本的な異議を持っており、これに関しては既に本論第２章で論駁しておいた。

ソフトウェア仕損の兆候を識別する規準は、「計画比規準」であり、ソフトウェア開発プロジェクトが失敗か否かという識別とし、規定は次の通りとする。なお、一般的にはプロジェクトの失敗は３大指標であるＱＣＤで捉えるが、筆者はそれにソフトウェアの規模を追加することを提言する。それによって、失敗が十全に捕捉可能となるからであり、理由は後述する。規定はOR条件であり、いずれかが該当すれば、プロジェクトは失敗と見做し、仕損の兆候があると識別する。なお、プロジェクトを中止・中断した場合は、兆候の識別に留まらず、直ちにすべてを特別仕損とする。

①ソフトウェアの規模が、概ね計画比20%を超過した場合、プロジェクトは失敗と見做す。
②ソフトウェアの開発期間が、計画より延長した場合、プロジェクトは失敗と見做す。
③ソフトウェアの品質が、概ね計画比20%超悪化した場合、プロジェクトは失敗と見做す。
④ソフトウェアのコストが、概ね計画比20%を超過した場合、プロジェクトは失敗と見做す。

まず、すべてを計画との対比としているが、失敗か否かを実務的に捉える場合[349]のソフトウェア業界での共通認識であり、業界慣行と言えるものなので妥当であると考える。次に、規準の数値を「20%超」[350]としたが、これは明快に提示するための参考値であり、原則主義的な規定とし、幅を持たせてもよいし、企業が適宜設定可能とするのでもよいと考えている。ただし、失敗と見做されることを回避する操作をさせない限度規定は必要であろう。いずれにせよ、選択理由を含めて開示することで、企業の姿勢や効率性を窮知することができる。

個々の説明に入る。①の規模は、ソフトウェアの機能が増加し、利便性・有用性が増した場合、より良いソフトウェアになったのだから、失敗ではないという抗弁を封じるためである。無条件にソフトウェアの価値評価をしているわけではなく、開発計画を策定し、それが承認されて開発を実施するという事業活動であるのだから、計画と違えた結果は失敗なのである。また、規模を規定に加えたのは、②開発期間や④コストが同様の意味で、規模が増加すれば期間やコストが増加することは対応関係としては合理性を有するが、それにより②や④の規定を無効化させたり、擦り抜けさせないためである。更に、後述するように、規模は仕損の測定では必須の基本算出項目である。

②の開発期間は、他の規定と異なって許容のアローワンスを持たせていないが、プロジェクトで

[349] 開発の事後評価では、計画対比以外に開発経験により習得できたこと等、または計画対比でもQCDより精細な事項に関して行なうこともあるが、この文脈では関係のないことなので立ち入らない。

[350] 例えば、①人月単価の契約で150H-180Hまでは固定で、それを上下する場合には時間精算するといったこと(アローワンス20%程度)、②プロジェクトの進捗管理で20%までの遅延は様子見とし、それを超えた遅延は対策を講じることが義務付けられるといったルールが比較的多いこと、③契約書上に明記されることはほとんどないが、請負において規模や工数の増加が20%程度を超えた場合には、追加契約交渉をする場合が慣行的に少なくないこと、④マイクロソフトでは開発期間に関して「予備期間」(想定外のことに対応する緩衝期間)を20%程度設定していること(Cusumano(1995)邦訳pp.283-284)、などを参考にしている。

遵守すべきミッションの筆頭に挙げられることであり、そもそも運用に入れず、利用できないということにおいて、掛け値なしに失敗だからである。

③に関しては、１つは、品質には広狭様々な捉え方があるが、仕損の捕捉では狭義の品質、すなわちバグの有無とする。広義の品質は、一般的な評価としては妥当でも、失敗あるいは仕損に照準を定めている場合には余計である。２つには、バグは「実数」で測定可能だが、広義の品質は主観的な「指数」で測定せざるを得ないものがあり、測定の硬度が劣るからである。なお、バグの具体的な指標は後で詳述する。

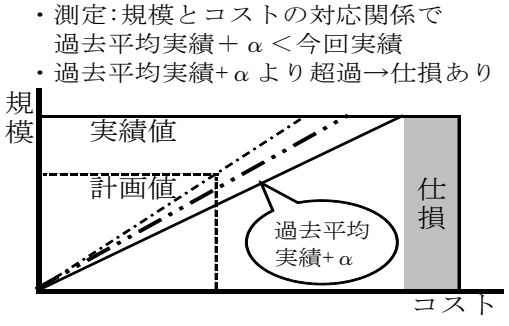
④のコストは、開発費総額で計画対比を行なう。

（3）ソフトウェアの仕損の認識

ソフトウェア仕損の兆候があると識別された場合には、次に仕損の認識を行なうことになる。仕損の兆候の識別は計画比で、１つのソフトウェア開発プロジェクト全体の対応関係により行なうのが妥当であるが、仕損の認識は今回実績を過去実績比で、ソフトウェアにおける個々のタスク（プロセス）並びにモジュール（プロダクト）の単位における対応関係により行なうことにする。仕損の兆候が識別された以上は、更に踏込んだ判定をする必要があり、計画値の設定には当為や期待といった作為が混入しており、実績値と性格が全同ではなく、また情報として不確実性を呈している。それに対して、過去実績値は情報として確実なものであり、性格は全同で、近似的な諸条件で達成できた数値としては、今回実績の妥当性を比較・判定する尺度として最も適格性を有していると言える。また、認識と測定を同様の規準で連続的に行なえる、いわゆる１段階アプローチ[351]が適切だと考えているので、先述のように精細な粒度で、測定を主眼とした規準とするためである。

なお、プロジェクトを中止・中断した場合は、この認識の規準に関わらず、すべてを特別仕損とする。測定に関しても、同様である。

図表１−６−１　ソフトウェア仕損の認識並びに測定のモデル



（出典：筆者作成）

仕損の認識並びに測定において用いる「過去平均実績緩衝規準」の考え方を説明する。仕損の認

[351] 平松(2012)p. 104 では、IAS36「資産の減損」の１段階アプローチに対して、日本の減損基準を２段階アプローチとして対比的に取上げている。また、秋葉(2011)p. 232 でも同様に、１段階方式と２段階方式として取上げている。

識は、今回実績を過去実績の平均値、例えば過去５ヶ年の開発実績の平均値、との比較で行なう。その際に、単純に絶対値での過去実績との比較では意味がないから、単位規模当たりの値（規模との対応関係）で比較を行なうことにする。同一尺度での比較なので有意だからである。更に、過去平均実績を超過していても、その超過を直ちにすべて仕損と見做すのではなく、予め許容範囲を設定し、その閾値を超えた場合にその分を仕損と見做すのである。図表に示したように（規模だけを表示したが、ＱＣＤに関しても同様である）、例えば図表の+αに該当する、超過が 20%以内は許容範囲とし、それを超えた場合に仕損と見做し、仕損を限定的に捉えるのである。正常仕損を捕捉対象とせず、品質に関して逐一のバグを捉えるのではなく、バグ全体の程度により判断するということである。

過去平均実績緩衝規準並びに個々のタスク（プロセス）並びにモジュール（プロダクト）の単位での捕捉を採用する理由の１つは、１つのソフトウェアを全体として取り扱う場合、個々のモジュールや作業では良悪のバラつきがあっても全体で平均化されてしまうが、個々の精細な粒度で取り扱う場合、格段に精確となり、「良」は捨象されるので、一定の緩衝領域を設定するのが穏当と考えるからである。２つには、プロジェクトの失敗の発生率を鑑みると、第１段階で余りに厳格な規準とすることは実施可能性を難しくするからである。３つには、現段階では、過去平均実績値が十分に信頼できる確実な情報として企業が保有しているとは限らないので、それを集積する猶予を配慮することが必要と考えるからである。ソフトウェア開発実務が改善され、ソフトウェア会計実務としても仕損処理が定着した第２段階では、予め許容範囲を設定しない「過去平均実績規準」に変更することが望まれる。

仕損の認識並びに測定に用いる具体的な項目は、生産性並びに品質とする。兆候の識別で取り扱った項目との対応関係は、これまでは明示的に規模との対応関係でＱＣＤを捉えたのに対して、認識並びに測定ではそれとは一見異なるように見えても、取り扱う項目にそれが織り込み済みなので明示的でないだけで、同様なのである。何故ならば、コスト並びに開発期間と、生産性とは、事象として、生産性の悪化が、進捗の遅延による開発期間の延長あるいは工数の増加によるコストの増加、に帰結するという意味で、生産性がコスト並びに開発期間を集約的に表現している項目なのである。また、ソフトウェアの定量化手法であるソフトウェア測定（Software Measurement）においても、生産性の実績値は、直接的に測定できず、実績生産性＝実績規模／実績工数という算定式により、実績規模と実績工数から導出するのである。それ故、生産性には規模と開発期間が集約され、それらの測定値を裏付けとしてコストを算出するわけである。先に、仕損測定で規模が必須の基本算出項目であると言及した所以である。

それに対して、品質はこれまでと全く同様である。具体的には、a. バグ発生率（規模単位当たりの発生件数）、b. バグ除去率（規模単位当たりの発生件数に対する除去件数）、c. バグ残存率（規模単位当たりの残存件数の推定値）の３項目とする。c. は最終的なプロダクトの品質を表す指標であり、a. と b. はプロダクトの中間的な品質指標であり、且つ品質に関するプロセスを表す指標でもあ

るので、この３項目で品質を捕捉すれば必要且つ十分であるからである。

なお、上記の図表では規模とコストの対応関係を示したが、実際には下記の図表に示すように生産性並びに品質に関して比較を行ない、過去平均実績＋αより超過している場合には、仕損と認識する。

（4）ソフトウェアの仕損の測定

ソフトウェア仕損が認識された場合には、引き続き仕損の測定を行なうことになる。実際には認識と測定は一連の流れで行なうが、説明上は分けて行なう。測定の規準は「過去平均実績緩衝規準」であり、測定の単位と項目は、タスク（プロセス）並びにモジュール（プロダクト）毎の生産性並びに品質である。認識で示したものと同様であり、認識の連続的プロセスとして、単位毎の実績値に対する、過去平均実績値により、仕損を測定する。

具体的な測定単位当たりの測定方法（計算方法）を提示する。各項目の定義は、次の通りである。

（g）実績率＝今回実績値／過去平均実績値。仕損費を円単位まで計算可能とするため、小数第６位までを有効とする。なお、仕損費は円未満切り捨てとする。

（h）今回実績値（単位規模当たり）＝今回実績値／今回実績規模

（i）過去平均実績値（単位規模当たり）＝｛（過去実績値$_1$／過去実績規模$_1$）＋（過去実績値$_2$／過去実績規模$_2$）＋…＋（過去実績値$_n$／過去実績規模$_n$）｝／n

（j）生産性実績率＝生産性今回実績値／生産性過去平均実績値

（k）品質実績率＝品質今回実績値／品質過去平均実績値

上記の各実績率以外の項目は、以降の計算式で導出されるものである。なお、項目（変数）は、日本語表記のほうが解りやすいので、記号化表記をせず、そのままとする。

判定式並びに計算式の前提は、４つの区分を設けることである。第１区分は、生産性実績率並びに品質実績率の「過去平均実績範囲内」とする。第２区分は、悪化程度が 20%以内を「許容範囲内」の緩衝域とする。第３区分は、「異常仕損」は悪化程度が 20%超～50%以内とする。第４区分は、「特別仕損」の悪化程度が 50%超とする。以下の斜字体の定数は、それに基づいたものである。区分範囲の境界値を異なる設定にする場合は、定数を変更すればよい。

仕損費を具体的にどのように計算するか、一覧的に挙示した図表に沿って説明する。

①生産性で異常仕損が発生し、品質は過去平均実績範囲内だった場合の判定式は、0.2＜1－生産性実績率≦0.5　＆　品質実績率－１≦0.0、とする。異常仕損費の計算式は、生産性異常仕損率＝0.8－生産性実績率、異常仕損費＝実績コスト×生産性異常仕損率、とする。

②生産性で異常仕損が発生し、品質は過去平均実績範囲外であるが、許容範囲内だった場合の判定式は、0.2＜1－生産性実績率≦0.5　＆　品質実績率－１≦0.2、とする。異常仕損費の計算式は、生産性異常仕損率＝0.8－生産性実績率、異常仕損費＝実績コスト×生産性異常仕損率、とする。

③生産性で特別仕損が発生し、品質は過去平均実績範囲内だった場合の判定式は、0.5＜1－生産

性実績率　＆　品質実績率－１≦0.0、とする。異常仕損費の計算式は、生産性異常仕損率＝0.3、異常仕損費＝実績コスト×0.3、とする。特別仕損費の計算式は、生産性特別仕損率＝0.5－生産性実績率、特別仕損費＝実績コスト×生産性特別仕損率、とする。

図表１-６-2　ソフトウェア仕損に関わる生産性、品質の範囲区分

		品質			
		過去平均実績範囲内	許容範囲内	異常仕損	特別仕損
生産性	過去平均実績範囲内			⑤ 品質異常仕損	⑦ 品質異常仕損 品質特別仕損
	許容範囲内			⑥ 品質異常仕損	⑧ 品質異常仕損 品質特別仕損
	異常仕損	① 生産性異常仕損	② 生産性異常仕損	⑨ 生産性異常仕損 品質異常仕損	⑪ 生産性異常仕損 品質異常仕損 品質特別仕損
	特別仕損	③ 生産性異常仕損 生産性特別仕損	④ 生産性異常仕損 生産性特別仕損	⑩ 生産性異常仕損 生産性特別仕損 品質異常仕損	⑫ 生産性異常仕損 生産性特別仕損 品質異常仕損 品質特別仕損

（出典：筆者作成）

④生産性で特別仕損が発生し、品質は過去平均実績範囲外ではあるが、許容範囲内だった場合の判定式は、0.5＜1－生産性実績率　＆　品質実績率－１≦0.2、とするる。異常仕損費の計算式は、生産性異常仕損率＝0.3、異常仕損費＝実績コスト×0.3、とする。特別仕損費の計算式は、生産性特別仕損率＝0.5－生産性実績率、特別仕損費＝実績コスト×生産性特別仕損率、とする。

⑤生産性は過去平均実績範囲内だったが、品質で異常仕損が発生した場合の判定式は、1－生産性実績率≦0.0　＆　0.2＜品質実績率－１≦0.5、とする。異常仕損費の計算式は、品質異常仕損率＝品質実績率－1.2、異常仕損費＝実績コスト×品質異常仕損率、とする。

⑥生産性は過去平均実績範囲外であるが、許容範囲内だったが、品質で異常仕損が発生した場合の判定式は、1－生産性実績率≦0.2　＆　0.2＜品質実績率－１≦0.5、とする。異常仕損費の計算式は、品質異常仕損率＝品質実績率－1.2、とする。異常仕損費＝実績コスト×品質異常仕損率、とする。

⑦生産性は過去平均実績範囲内だったが、品質で特別仕損が発生した場合の判定式は、1－生産性実績率≦0.0　＆　0.5＜品質実績率－１、とする。異常仕損費の計算式は、品質異常仕損率＝0.3、異常仕損費＝実績コスト×0.3、とする。特別仕損費の計算式は、品質特別仕損率＝品質実績率－*1.5*、特別仕損費＝実績コスト×品質特別仕損率、とする。

⑧生産性は過去平均実績範囲外であるが、許容範囲内だったが、品質で特別仕損が発生した場合の判定式は、1－生産性実績率≦0.2　＆　0.5＜品質実績率－１、とする。異常仕損費の計算式は、品質異常仕損率＝0.3、異常仕損費＝実績コスト×0.3、とする。特別仕損費の計算式は、品質特別仕損率＝品質実績率－1.5、特別仕損費＝実績コスト×品質特別仕損率、とする。

⑨生産性で異常仕損が発生し、品質でも異常仕損が発生した場合の判定式は、0.2＜1－生産性実績率≦0.5　＆　0.2＜品質実績率－１≦0.5、とする。異常仕損費の計算式は、生産性異常仕損率＝0.8－生産性実績率、品質異常仕損率＝品質実績率－1.2、異常仕損費＝実績コスト×（生産性異常仕損率＋品質異常仕損率－生産性異常仕損率×品質異常仕損率)、とする。

⑩生産性で特別仕損が発生し、品質では異常仕損が発生した場合の判定式は、0.5＜1－生産性実績率　＆　0.2＜品質実績率－１≦0.5、とする。異常仕損費の計算式は、生産性異常仕損率＝0.3、品質異常仕損率＝品質実績率－1.2、異常仕損費＝実績コスト×（0.3＋品質異常仕損率－0.3×品質異常仕損率)、とする。特別仕損費の計算式は、生産性特別仕損率＝0.5－生産性実績率、特別仕損費＝実績コスト×生産性特別仕損率、とする。

⑪生産性で異常仕損が発生し、品質では特別仕損が発生した場合の判定式は、0.2＜1－生産性実績率≦0.5　＆　0.5＜品質実績率－１、とする。異常仕損費の計算式は、生産性異常仕損率＝0.8－生産性実績率、品質異常仕損率＝0.3、異常仕損費＝実績コスト×（生産性異常仕損率＋0.3－生産性異常仕損率×0.3)、とする。特別仕損費の計算式は、品質特別仕損率＝品質実績率－1.5、特別仕損費＝実績コスト×品質特別仕損率、とする。

⑫生産性で特別仕損が発生し、品質でも特別仕損が発生した場合の判定式は、0.5＜1－生産性実績率　＆　0.5＜品質実績率－１、とする。異常仕損費の計算式は、生産性異常仕損率＝0.3、品質異常仕損率＝0.3、異常仕損費＝実績コスト×（0.3＋0.3－0.3×0.3）＝実績コスト×0.51、とする。特別仕損費の計算式は、生産性特別仕損率＝0.5－生産性実績率、品質特別仕損率＝品質実績率－1.5、特別仕損費＝実績コスト×（生産性特別仕損率＋品質特別仕損率－生産性特別仕損率×品質特別仕損率)、とする。

図表１-６-3　ケース別仕損試算表

ケース	生産性			品質			仕損費(実績コスト単位当たり)		
	実績値(FP/人月)	異常仕損率	特別仕損率	実績値(件/FP)	異常仕損率	特別仕損率	異常仕損費	特別仕損費	仕損費合計
①	9.00	0.050		0.006			50,000		50,000
②	9.00	0.050		0.007			50,000		50,000
③	5.00	0.300	0.083333	0.006			300,000	83,333	383,333
④	5.00	0.300	0.083333	0.007			300,000	83,333	383,333
⑤	12.00			0.008	0.133333		133,333		133,333
⑥	10.00			0.008	0.133333		133,333		133,333
⑦	12.00			0.010	0.300000	0.166667	300,000	166,667	466,667
⑧	10.00			0.010	0.300000	0.166667	300,000	166,667	466,667
⑨	9.00	0.050		0.008	0.133333		176,667		176,667
⑩	5.00	0.300	0.083333	0.008	0.133333		393,333	83,333	476,667
⑪	9.00	0.050		0.010	0.300000	0.166667	335,000	166,667	501,667
⑫	5.00	0.300	0.083333	0.010	0.300000	0.166667	510,000	236,111	746,111

（注）過去平均実績値：生産性12FP/人月、品質0.006件/FP,実績コスト1000,000円

（出典：筆者作成）

仕損を上記の①～⑫の通りに設定し、計算した試算結果の例は、図表に掲示する。注記した過去

平均実績に対して、各ケースは生産性又は品質で、異常仕損ないし特別仕損に該当する実績となり、ケース毎に異常仕損費又は特別仕損費を計上することになる。

①の数値事例は、生産性実績値が 9.00FP/人月なので、生産性実績率は 0.75＝9.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.006 件/FP なので、品質実績率は 1.000＝0.006 件/ FP÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.05＝0.8-0.75 となり、異常仕損費は 50,000 円＝1,000,00 円×0.05 となる。

②の数値事例は、生産性実績値が 9.00FP/人月なので、生産性実績率は 0.75＝9.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.007 件/FP なので、品質実績率は 1.167≒0.007 件/ FP÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.05＝0.8-0.75 となり、異常仕損費は 50,000 円＝1,000,00 円×0.05 となる。

③の数値事例は、生産性実績値が 5.00FP/人月なので、生産性実績率は 0.416667≒5.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.006 件/FP なので、品質実績率は 1.000＝0.006 件/ FP÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.3 となり、異常仕損費は 300,000 円＝1,000,00 円×0.3 となる。生産性特別仕損率は 0.083333＝0.5－0.416667 となり、特別仕損費は 83,333 円＝1,000,00 円×0.083333 となる。

④の数値事例は、生産性実績値が 5.00FP/人月なので、生産性実績率は 0.416667≒5.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.007 件/FP なので、品質実績率は 1.167≒0.007 件/ FP÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.3 となり、異常仕損費は 300,000 円＝1,000,00 円×0.3 となる。生産性特別仕損率は 0.083333＝0.5－0.416667 となり、特別仕損費は 83,333 円＝1,000,00 円×0.083333 となる。

⑤の数値事例は、生産性実績値が 12.00FP/人月なので、生産性実績率は 1.0＝12.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.008 件/FP なので、品質実績率は 1.333333≒0.008 件/ FP÷0.006 件/ FP となり、判定式に合致する。品質異常仕損率は 0.133333＝1.333333-1.2 となり、異常仕損費は 133,333 円＝1,000,00 円×0.133333 となる。

⑥の数値事例は、生産性実績値が 10.00FP/人月なので、生産性実績率は 0.833333＝10.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.008 件/FP なので、品質実績率は 1.333333≒0.008 件/ FP÷0.006 件/ FP となり、判定式に合致する。品質異常仕損率は 0.133333＝1.333333-1.2 となり、異常仕損費は 133,333 円＝1,000,00 円×0.133333 となる。

⑦の数値事例は、生産性実績値が 12.00FP/人月なので、生産性実績率は 1.0＝12.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.010 件/FP なので、品質実績率は 1.666667≒0.010 件/ FP÷0.006 件/ FP となり、判定式に合致する。品質異常仕損率は 0.3 となり、異常仕損費は 300,000 円＝1,000,00 円×0.3 となる。品質特別仕損率は 0.166667＝1.666667－1.5 となり、特別仕損費は 166,667 円＝1,000,00 円×0.166667 となる。

⑧の数値事例は、生産性実績値が 10.00FP/人月なので、生産性実績率は 0.833333＝10.00 FP/人

月÷12.00 FP/人月となり、品質実績値が 0.010 件/FP なので、品質実績率は 1.666667≒0.010 件/ FP ÷0.006 件/ FP となり、判定式に合致する。品質異常仕損率は 0.3 となり、異常仕損費は 300,000 円＝1,000,00 円×0.3 となる。品質特別仕損率は 0.166667＝1.666667－1.5 となり、特別仕損費は 166,667 円＝1,000,00 円×0.166667 となる。

⑨の数値事例は、生産性実績値が 9.00FP/人月なので、生産性実績率は 0.75＝9.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.008 件/FP なので、品質実績率は 1.333333≒0.008 件/ FP ÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.05＝0.8-0.75 となり、品質異常仕損率は 0.133333＝1.333333-1.2 となり、異常仕損費は 176,667 円≒1,000,00 円×（0.05＋0.133333－0.5×0.133333）となる。

⑩の数値事例は、生産性実績値が 5.00FP/人月なので、生産性実績率は 0.416667≒5.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.008 件/FP なので、品質実績率は 1.333333≒0.008 件/ FP ÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.3 となり、品質異常仕損率は 0.133333＝1.333333-1.2 となり、異常仕損費は 393,333 円≒1,000,00 円×（0.3＋0.133333－0.3×0.133333）となる。生産性特別仕損率は 0.083333＝0.5－0.416667 となり、特別仕損費は 83,333 円＝1,000,00 円×0.083333 となる。

⑪の数値事例は、生産性実績値が 9.00FP/人月なので、生産性実績率は 0.75＝9.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.010 件/FP なので、品質実績率は 1.666667≒0.010 件/ FP ÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.3 となり、品質異常仕損率は 0.3 となり、異常仕損費は 335,000 円＝1,000,00 円×（0.05＋0.3－0.05×0.3）となる。品質特別仕損率は 0.166667＝1.666667－1.5 となり、特別仕損費は 166,667 円＝1,000,00 円×0.166667 となる。

⑫の数値事例は、生産性実績値が 5.00FP/人月なので、生産性実績率は 0.416667≒5.00 FP/人月÷12.00 FP/人月となり、品質実績値が 0.010 件/FP なので、品質実績率は 1.666667≒0.010 件/ FP ÷0.006 件/ FP となり、判定式に合致する。生産性異常仕損率は 0.3 となり、品質異常仕損率は 0.3 となり、異常仕損費は 510,000 円＝1,000,00 円×（0.3＋0.3－0.3×0.3）となる。生産性特別仕損率は 0.083333＝0.5－0.416667 となり、品質特別仕損率は 0.166667＝1.666667－1.5 となり、特別仕損費は 236,111 円≒1,000,00 円×（0.083333＋0.166667－0.083333×0.166667）となる。

なお計算上、生産性実績率は極限的にはゼロに収束するが、品質実績率は拡散し、収束しないために、仕損費が実績コストを上回ることがあり得る。しかし、仕損に関してペナルティを課すことが目的ではないので、その場合には、実績コスト全額を仕損費と見做す取り扱いとする。

最後に、監査可能性ないし検証可能性に言及しておきたい。精細な粒度で精細な数値を捕捉するものであるが、コストに関しては給与明細など、委託開発であれば請求書並びに検収書などの経理書類、生産性並びに品質に関してはプロジェクト管理資料という証跡があり、算出・集計した処理手順が示されれば、追跡して検証することは、十分に可能である。

また、この会計処理のコスト制約であるが、コストに関しては、それまでの管理粒度が粗い場合には、精細化するためのコストを必要とする。しかし、開発実務の作業はタスク・ベースで行なうことが普及しているので、管理の粒度を精細化し、記録するコストに留まり、実際の作業そのものに及ぶものではない。これに対するベネフィットに関しては、目的適合性を充足し、忠実な表現となる情報開示を行なう企業であることを示し、利害関係者の信頼を得るという便益が得られ、その関係改善ないし良好な関係構築に寄与することになる。そういう意味では、コスト／ベネフィットが過大な負担ないし著しい不均衡となることはないと考える。

本章の冒頭で言及した問題提起に対しては、ほぼすべて応え得たものと思う。ソフトウェアの開発実務ではそれほど珍しいことではないにも関わらず、一般的には余り知られていないソフトウェア開発プロジェクトの失敗に対して、またこれまで捕捉が困難とされてきた仕損を、筆者は会計的に適正に捕捉可能とし、仕損費として取り扱う会計処理案を提示した。

仕損会計処理案は、減損会計に準じた会計的な構想に基づき、具体化した。考慮点は、次の通りである。①ソフトウェア業界の慣行である３大指標ＱＣＤでプロジェクトの失敗を捉えることを基本認識とすること、②仕損はプロジェクトの中止・中断と、継続・完了した場合の品質不良と作業非効率に大別すること、③適用対象は研究開発目的以外のソフトウェア開発の全てとすること、④仕損の区分は、非原価項目の費用処理する異常仕損と、仕損度合いが甚大で特別損失とする特別仕損とすること、⑤仕損の兆候の識別は計画比の４要件で行なうこと、⑥仕損の認識はより精緻に行なう必要性から、個々のタスク並びにモジュールを各々生産性と品質の指標で行なうこと、⑦認識規準は「過去平均実績緩衝規準」とし、認識と測定を連続的に行なう１段階アプローチとすること、である。

なお、兆候の識別や認識並びに測定の規準には、具体的な数値を示したが、あくまで参考値で、原則主義規定であると断った上でのことだが、これらに関しては実態との照合を更に行なう必要があるであろう。また、会計処理案の適用を２段階構想とし、第１段階ではプロジェクトの失敗に限定し、第２段階で失敗か否かに関わらず、全てのソフトウェア並びにソフトウェア開発に適用するという構想に関しては、実務の現状への配慮を理由としているが、段階分けをしないことも一案であろう。更に、異常仕損と特別仕損に区分すること、両者の境に挙げた数値により区分することの妥当性には議論の余地があるかもしれない。これらに関しては、更なる検討を深めていきたいと考えている。

また、本章では規模と工数並びにコストの関係を「正比例的」と見做しているが、近似的には間違いではないし、実務における大方の扱いでもあるが、精密には非線形的と捉えるべきである。そうした認識に則った開発見積り手法もあるが、余り普及していない。過去平均実績との対比も、一律の扱いをしているが、精密にはソフトウェアの規模や難易度、技術者のスキルなどを考慮すべかもしれない。これらに関しては、仕損の測定が複雑になること、いまだソフトウェア工学の分野で

も懸案事項となっていることを理由に見送らざるを得なかった。更に、本章は会計処理案の提示を目的としたため、情報開示をどのように行なうか、具体的な提示を行なっていない。これらに関して具体化することは、今後の課題としたい。

本論

第７章　システム運用改善

１．システム運用の概観

（1）システム運用全般の概観

（2）システム運用改善へのアプローチ

２．システム運用に係る現行の会計処理

（1）日本基準におけるシステム運用に係る会計処理

（2）アメリカ基準におけるシステム運用に係る会計処理

３．システム運用改善に適合的な会計処理

（1）ソフトウェアの定義改訂案の再確認

（2）システム運用改善の例示

（3）システム運用改善に適合的な会計処理

図表１−７−1　システム運用の概観

第７章　システム運用改善

本章は、ソフトウェア・ライフサイクルに沿って、システム運用、その中でも特にシステム運用改善を主題的に取り上げて考察する。

最初に触れておきたいことが２つある。１つは、用語に関して、「ソフトウェア運用」ではなく、「システム運用」を統一的に使用するが、それは運用がソフトウェアの開発・保守等の大フェーズにも増して、ハードウェアと一体的に稼働するフェーズであるため、両者を包含したシステム運用とする方が実態に適合的だからである。但し、ハードウェアと一体的とは言え、会計的に主として問題にするのは、ソフトウェアに係る事象であることは言うまでもない。

もう１つは、現行のソフトウェア会計基準[352]には、運用に関する規定はごく僅かなものしかないことである。従って、考察の途上で取り上げるが、それに依拠したりあるいは批判的に取り上げたりすることだけでは運用全体をカバーすることができないので、主として独自に考察することにならざるを得ないことである。現行の日本基準は、ソフトウェアの開発が主たる対象範囲となっており、ソフトウェアの保守さえ「市場販売目的のソフトウェア」に関する規定に限られており、「自社利用のソフトウェア」に関する明示的な規定はないという局限的なものである[353]。そもそもソフトウェアのライフサイクル全域を捕捉する基準にはなっていないのである。しかし、言うまでもないことだが、開発は確かに一時的には多額の開発コストを費消する目立った事象ではあるが、ライフサイクルにおいては保守並びに運用が圧倒的に長い期間を占めており、尚且つそれだけではなく、ＴＣＯ（Total Cost of Ownership：総保有コスト）という点でも開発コストを遥かに上回る場合が一般的と言えるくらいであり[354]、会計的に軽視してよいことではない。

端的に問題を提起しよう。会計基準の分類で言えば「自社利用」を対象としてのことだが、システムは運用において、購入ないし自社開発等を行なったソフトウェア等を予め資産計上している場合には、それが運用において利用されることで、実際に「将来の経済的便益」を具現する。通常の運用では、確かにそう捉えることで大凡差し支えない。だが、運用において新たに「将来の経済的便益」に寄与・貢献する、付加的な資産価値を生み出すことは一切ないのであろうか。運用改善には、そうしたことに該当することがあるのではないか。例えば、高機能・高性能のハードウェアにリプレースすることは性能向上等により運用改善となる。この場合、ハードウェアを購入等で取得すれば、ハードウェアは有形固定資産となる。あるいは、プログラムを改良し、機能の追加はない

---

[352] ソフトウェア会計基準としては、本章では日本基準並びにアメリカ基準を取り上げる。国際会計基準(IAS38)は、無形資産全般に関する基準なので取り上げない。

[353] ソフトウェアの保守に関しては、第９章で主題的に取り上げるので参照。

[354] 日経ＢＰシステム運用ナレッジ編著(2013)によると、2012 年度のＩＴ投資において、「回答した 491 社の平均では「新規開発コスト」が 24.3%、「保守開発コスト」は 30.8%、「運用管理コスト」は 44.9%だった。保守開発と運用管理を合わせると 75.7%と、ＩＴ投資の 3/4 がシステム開発後の保守・運用フェーズに投じられている」(p.9)、との調査結果が報告されている。

が、処理性能が向上すれば、利用者への処理応答は一層良好となり、利便性が増し、運用改善となる。この場合も、資産要件を充足すれば、ソフトウェアの資産計上となる。これらは、現行の会計処理で行なわれていることである[355]。それに対して、ハードウェアではなく、ソフトウェアだが、且つプログラムではないものによる運用改善の態様が、本章で詳細に取り上げるようにあるのである。ハードウェア又はプログラムによる前記の運用改善と同質の効果を発揮するのだが、ソフトウェア会計基準におけるソフトウェアの定義が完備的でないので、ソフトウェアとは看做されず、資産計上がなされていないのである。これは、均衡を欠く取り扱いであり、不適切ではないか。この運用改善の問題が、筆者が本章で主題的に取り上げることである。

本章の構成は、次の通りである。１．で、システム運用全般並びに運用改善へのアプローチを概観する。システム運用全般は、システム環境の構築と通常運用とトラブル対応に分けて概観する。また、運用改善に関して、幾つかのアプローチを概観する。これらによって、システム運用の全体像並びに運用改善が如何なる事象であるかを考察の前提ないし背景として捕捉する。２．で、システム運用に係る現行の会計処理を、日本並びにアメリカの会計基準における運用に関するごく限られた規定内容を確認する。３．で、それらの会計処理では適切に捕捉されていないシステムの運用改善に関する事象を取り上げ、現行の会計処理で資産計上される以外にも、付加的な資産性を有する成果物を産出することから、それに適合的な資産計上を行なう会計処理を事例に即して具体的に提示する。以上によって、これまで看過されてきた、独自的な意義を有するシステム運用改善に係る会計処理の在り方を提言する。

１．システム運用の概観

（1）システム運用全般の概観

システム運用全般は、大別すると、システム環境の構築、通常運用、トラブル対応に分けることができる。それらを順次概観すれば、ほぼ全体を遺漏なく捉えることができる[356]。簡潔に図表に一覧化し、多少の補足説明を加えることにする。

---

[355] 資産範囲の妥当性という問題はあるが、第９章で取り上げることでもあり、本章の主題とは直接の関係がないので、ここでは立ち入らない。

[356] 日立製作所監修(2009)では、一般的なＰＤＣＡ(Plan-Do-Check-Action)に倣って、システムの運用サイクルは(1)計画→(2)構築→(3)監視→(4)対処〔→次の(1)計画…〕、というように回っているとしているが(pp. 21-22)、このサイクルでは圧倒的な業務量を占める通常運用(後述するＩＴＩＬでは「サービスオペレーション」という段階)が(2)と(3)の間に埋没しており、また(3)→(4)が「改善」という側面に片寄って説明されおり、トラブル対応が適切に位置付けられていないので、支持できない。それ故、本文のような捉え方を採用する。

図表１−７−１　システム運用の概観

項番	項目名	内容
1-1	システム環境の構築	比較的短期間に一回的に行なわれる。以後は、部分的な改修・増設等があり、一定期間後に全面再構築が行なわれる。 ①物理的な設営：マシン室（データセンター）の設置等。 ②インフラの構築：ハードウェア、ネットワーク、ソフトウェア（基本並びにミドルソフトウェア）のインストール（各種設定含む）。 ③アプリケーション・システムの構築：アプリケーション・システムのインストール（各種設定含む）。
1-1-1	新システムのリリース	新システムの購入又は開発の都度行なわれる。また、既存システムの保守の都度行なわれる。 ①旧システムがある場合、その資源を保存し、運用環境を消去する（新システムのリリースに不具合がある場合等には、復旧させる）。また、データ移行をする場合が多い。 ②アプリケーション・システムの構築：アプリケーション・システムのインストール（各種設定含む）。なお、物理的な設営並びにインフラ構築を一部伴う場合もある。
1-2	通常運用	全般的にはシステムが存続する限り継続し、個別には個々のシステムを廃棄するまで継続する。 ①運用サイクル（オンライン処理は随時、バッチ処理は日次・週次・月次・年次等）に沿って行なわれるのが一般的である。 １日のサイクルは、早朝にオンライン・システムを立ち上げ、夕方ないし深夜にオンライン・システムを止め、そのあと夜間バッチ処理（データ等のバックアップ含む）を実行し、終了するのが典型的である。 今日では、365日24時間フル稼働も増えている。 ②運用管理システム等により、運用の自動化が増えているが、非定型的なＪＯＢは臨時的に手動でオペレーションされる。 ③オペレーション以外に、サービス・デスクの利用者へのサービス業務が大きなウェイトを占めている。
1-2-1	システム監視	通常運用と並行的に随時行なわれる。 ①サービスの提供並びにインフラの稼働が「正常性」の範囲内であるか、チェックする。 ②「正常性」の範囲は、ＳＬＡ等で定める。
1-3	トラブル対応	トラブルの発生都度、対応する。 ①大凡の手順は、(1)トラブルの一報ないし異常検知→(2)事態の初期的把握→(3)縮退運転等の対処→(4)関係部門等への連絡・指示→(5)原因調査→(6)応急処置→(7)本格対応→(8)復旧作業→(9)復旧の連絡、である。 但し、トラブルの性格、影響の大小等で具体的な対応は相当程度異なる。

（出典：筆者作成）

システム環境の構築は、オンプレミス（on-premises：自社構内設置）を想定した記載としているが、クラウド・コンピューティング等を利用するのであれば、内容は変わらないが、①物理的な設営[357]並びに②インフラの構築に関してはクラウド事業者等に委ねることになる。また、自社保有の場合でも、計画等は主として自社で行なうとしても（コンサルテーション等の委託を含む）、①並びに②の作業はほとんど各々（建築、電気、電話、フィールド・サービス等）の業者に任せることになる。なお、いずれの場合でもエンタープライズ系（企業の業務システム全般で組み込み系を除く）

[357] ＪＥＩＴＡ(2012)pp. 129-137 を参考にした。

を想定しているが、製造装置等の組み込み系の場合は、工場の設営全般の一部として位置付けられ、その一環として行なわれることになる。その場合は、②と③も予めセッティングされた状態の装置が搬入されることが多いが、②の一部ないし特に③は現調（現地調整）という形で現地作業が行なわれることもあり、大凡はエンタープライズ系に準じて①～③が行なわれると捉えて差し支えない（以下、逐一言及しないが、主としてエンタープライズ系を想定した記載としているが、組み込み系を含めたシステム全般のことと見做して大過ない）。

通常運用に関することでは、深夜でも無人化しない場合には、オペレーションの３交替制を採ることが一般的であり、そうしたことを含めて、管理以外の運用実務を社員に担当させることが回避され、相当程度に委託・準委任・派遣ないしアウトソーシングがなされているのが実態である。

システム監視[358]に関してであるが、「正常性」の範囲内と言って、一定の幅を持たせているのは、例えば一時的に高負荷の処理が実行され、他の処理の応答時間が通常より遅くなっても、高負荷の処理が終了すれば、旧に復する場合や、１台のサーバがハードウェア障害を起こしていても、複数台で並列的に処理することになっているシステム構成では、至急トラブル対応をしなければならない事象ではないといった許容範囲を設定し運用することを意味している。これを体系的に規定したものがＳＬＡ（Service Level Agreement）である[359]。その「正常性」の範囲を超えた場合には、トラブル対応をすることになる。あるいは、「正常性」の範囲は適宜見直しを行なうことになっており（一般的には年度単位等で）、システム監視で把握している状況を基に、運用改善を行なっていくことになる。それが嵩じれば、システム環境の再構築に発展していく。

トラブル対応は、利用者へのアナウンス、二次災害の回避、実働部隊が解決に専心できる体制作り（苦情等の割り込みで作業に集中できない状況が生まれやすいので）がポイントであるが、改善するための「好機」でもあり得る。なお、システム・トラブルはマスメディアで報道されるような社会的に甚大な影響を及ぼすものはそれほど多くはないが、一般に知られている以上には起きているのであり[360]、従ってトラブル対応はマニュアル化された「定常的な」運用業務であることは銘記しておくべきである。

（２）システム運用改善へのアプローチ

---

358 アールワークス編著(2012)p. 59 並びに野村総合研究所(2011)pp. 42-43, 46-49 を参考にした。

359 ＪＥＩＴＡ(2012)では、「ＩＴＩＬでは最終的な利用者との合意事項／社内システム部門との合意事項／外部ベンダなどとの合意事項をＳＬＡ／ＯＬＡ／ＵＣ、と区別しているが、「民間向けＩＴシステムのＳＬＡガイドライン」ではサービス提供者と利用者の二者間の合意を、すべてＳＬＡとした」(p. 4)、としている（ＯＬＡ：operational level agreement, ＵＣ：underpinning contract）。運用サービスの合意内容を主として問題にする限り、ＪＥＩＴＡの言う通り区別しなくとも差し支えないであろう。

360 トラブル並びにトラブル対応がどの程度の頻度で発生するか、実態は様々であるが、１つの具体的な事例としては野村総合研究所(2011)が挙げられる(p. 24, 28)。ＮＲＩ規模で１カ所のデータセンターで毎月数万件の障害が起きているというのがシステム運用の偽らざる実態である。

上記のようなシステム運用全般において、運用改善はどのような位置付けにあるのだろうか。日立製作所監修（2009）では、ネットワーク（監視）に即してのことだが、「運用業務には、ルーチンワーク（日常の運用業務）と保守（定期的な改善を実施する業務）の２種類」[361]がある、としている。そして、「保守は、ある程度の期間を想定してモニタリングを実施しながら稼働状況を把握し、サービスが正常に提供できていることを確認し、問題があるようならボトルネックとなる箇所を特定、メンテナンスとして、最適な環境に調整するという定期的な改善を実施する業務」[362]である、としている。通常運用においてシステム監視を行ない、それに基づいて正常な稼働状況の維持ないし原状回復、または運用改善を行なうことが「定期的な」一連の業務として組み込まれているのである。

ネットワーク（監視）に限定せず、通常運用における全般的なシステム監視並びにトラブル対応の延長上に運用改善を位置付けるものとして、アールワークス編著（2012）における「運用のサイクル」がある。そこでは、１監視→２（障害）対応→３分析→４－１改善／４－２拡張／４－３刷新、を掲げている[363]。一般的な運用サイクルというよりも、トラブルをトリガーとして、直接的なあるいは「対症療法的な」トラブル対応に留まらず、運用改善等に繋げていく業務の手順を定式化したものと言った方がよいだろう。危機を転換して好機と化すということであり、運用改善を積極的に位置付けるものである。また、同書では、「システム改善フローチャート」というものも掲げている（この場合の「システム改善」とは運用改善を意味する）。１システムリソース・ログデータ調査→２普段と傾向の異なるデータがあるか（Ｙｅｓは４へ）→３設計想定と異なるデータがあるか（Ｎｏは７へ）→４原因調査→５設定調整（問題改善）→６結果確認→７終了（再度１調査へ）[364]。これは、監視よりも能動的な調査を定期的に行ない、運用改善に繋げていく業務の手順を定式化したものと言える。そして、「設定調整」が「問題改善」と見做されていることに、筆者は運用改善の方策ないし態様として注目するのである。ハードウェアに関することも当然あるが、ソフトウェアに関することで、しかもプログラムではなく、各種環境定義類の設定調整（変更）が運用改善となることを意味しているからである。

更に、トラブルや調査をトリガーとした限定的なものではなく、運用全般に運用改善を構造的に組み込んだ枠組みとして、ＩＴＩＬがある。ＩＴＩＬは、ＩＴサービスのライフサイクルとして５つの段階を設定しているが、一般的なＰＤＣＡサイクルとは異なり、「ハブアンドスポーク設計を使用し、サービスストラテジをハブに、サービスデザイン、サービストランジション、サービスオペ

361 日立製作所監修(2009)p. 135

362 同 p. 135、傍点引用者。なお、この場合の「保守」は、同じ用語を使ってはいるが、ソフトウェアの開発・保守という場合の保守のことではなく、一般的な機器等の「保守」（メンテナンス）に近い意味で、あくまで運用におけるハードウェア並びにソフトウェアの「保守」（メンテナンス）のことである。少々紛らわしいかもしれないが、文脈的にそのように解しなければならない。

363 アールワークス編著(2012)p. 47

364 アールワークス編著(2012)p. 84。なお、付番並びに傍点は引用者。

レーションを回転するライフサイクル段階（スポーク）としている。継続的サービス改善は、サービス・ライフサイクルのすべての段階を取り巻いて支援している」[365]、という仕組みになっている。ＩＴＩＬ自体の普及はまだ広範囲に及んでいるという程ではないが、そもそもＩＴＩＬが膨大なＩＴサービスの経験を集約し、洗練させて「ベスト・プラクティス」化したものだとするならば（そしてＩＴサービスに関する ISO20000 という国際標準のベースとなり、我が国でもＩＴＳＭＳ（IT Service Management System）という企業の認証資格となっている)、このように運用全般に運用改善の契機が遍在している、と言うことができるのではないだろうか。

こうした仕組みにおいて、どのような運用改善があり得るのか、それを考える上で参考になるのが、前述したＳＬＡである。ＪＥＩＴＡ(2012)によると「民間向けＩＴシステムのＳＬＡガイドライン」では、サービス対象を(1)ＩＴサービス、(2)ＩＴプロセスマネジメント、(3)ＩＴリソースにカテゴリ分けし、各々のサービスレベル項目を設定している[366]。そして、ＳＬＡの設定手順は、サービスレベルの把握・調査→サービス対象の設定→サービスレベル項目の見直し→目標値の設定→評価・検証（→調整→目標値の設定）→契約[367]、である。ここで着目したいのは、ＳＬＡにおいて具体的な「目標値」を設定することであり、しかも適宜（一般的には年度契約が多いので、年に１度の契約更新時に）見直しを行ない、達成できるようにするために各種「調整」を行なった上で、「目標値」を再設定することである。これが、具体的な運用改善であり且つ数値的な測定可能性の高いことだということである。そのうち、筆者が取り分け注目するのは、ソフトウェアであり、しかもプログラムではない、各種環境定義類を目標値達成のために設定（調整・変更）することにより運用改善（パフォーマンスの向上、利便性の向上、堅牢性の強化等）を実現する場合である。

２．システム運用に係る現行の会計処理

（1）日本基準におけるシステム運用に係る会計処理

日本のソフトウェア会計基準における運用に関する規定は、１つは、実務指針 14.「ソフトウェアの導入費用の取扱い」として、「外部から購入したソフトウェアについて、そのソフトウェアの導入に当たって必要とされる設定作業及び自社の仕様に合わせるために行なう付随的な修正作業等の

---

[365] ＩＴＩＬ(2012a)p. 3。なお、ＩＴＩＬは、①サービス戦略を重視しており、それをコアとし、ＩＴサービス全般の②設計の段階と、それを実装する段階として、筆者が環境構築としたものに相当する③サービストランジションの段階と、通常運用並びにトラブル対応としたものに相当する④サービスオペレーションの段階を設定している。そして、⑤継続的サービス改善は①～④の全てに関与する仕組みとしているのである。

[366] ＪＥＩＴＡ(2012)pp. 52-53。(1)ＩＴサービスのサービスレベル項目、 (2)ＩＴプロセスマネジメントのサービスレベル項目、(3)ＩＴリソースのサービスレベル項目を、大項目として各々設定している。個別項目は 140 項目超、項目詳細は 480 超である(同 pp. 184-193, 194-219)。それだけ、運用改善の潜在的な可能性のある要素があるということである。

[367] 同書 p. 34。傍点は引用者。

費用は、購入ソフトウェアを取得するための費用として当該ソフトウェアの取得価額に含める。ただし、これらの費用について重要性が乏しい場合には、費用処理することができる。」、という規定である。これに関しては、更に 38.でより詳しく、「（１）購入ソフトウェアをそのまま導入する場合」として「追加の作業は簡単な導入作業程度である」ものと、「（２）購入ソフトウェアの設定等が必要になるケース」として、「財務会計ソフトの科目マスターの設定のように設定作業が必要となる場合、あるいは自社の仕様に合わせて画面や帳票などを修正する場合など」の例を挙げているものとがある、としている。これらの作業等の費用は、ソフトウェアを「使用するために不可欠な費用であり、有形固定資産の取得に要する付随費用と同様に」処理する、とある。なお、「導入」として同列に扱っているが、画面等の修正はパッケージ・ソフトウェアのカスタマイズで、ソフトウェアの保守であり、運用に関することではない。

もう１つの規定は、16.「その他の導入費用の会計処理　（１）データをコンバートするための費用」として「新しいシステムでデータを利用するために旧システムのデータをコンバートするための費用については、発生した事業年度の費用とする。」「（２）トレーニングのための費用」として「ソフトウェアの操作をトレーニングするための費用は、発生した事業年度の費用とする。」、というものである。併せて 40.では、これらの「ソフトウェアを利用するための環境を整備し有効利用を図るための費用は、原則としてソフトウェアそのものの価値を高める性格の費用ではない」ことから、発生時に費用処理をする、とある（傍点引用者）。なお、この点が後述する運用改善に係る主要な論点となる。いずれにせよ、「導入」に関する規定であり（筆者のいう新システムのリリースにほぼ相当する）、運用の大半を占める通常運用に関する規定はなく、ごく限られたものである[368]。

（２）アメリカ基準におけるシステム運用に係る会計処理

次に、ＦＡＳＢ(1985)は、1985 年に公表されたＦＡＳＢ(Financial Accounting Standards Board)のＳＦＡＳ（Statement of Financial Accounting Standards）第 86 号「販売、リースその他の方法で市場に出されたコンピュータ・ソフトウェアの原価の会計処理（Accounting for the Cost of Computer Software to Be Sold, Leased or Otherwise Marketed)」（現在の「FASB ACS 985-20-25-2 - Costs of Software to be Sold, Leased, or Marketed」（以降「ＳＦＡＳ86」と略記））で、

[368] ソフトウェアの資産性に関する検討委員会・委員長櫻井通晴、櫻井編著(1993)の「ソフトウェア会計実務指針[案]」は、日本のソフトウェア会計基準が設定される以前のものだが、明確にソフトウェアの会計上の取扱いを包括的に規定した指針案であるが、運用に関する規定はごく僅かに留まっている。パッケージ等の「客先導入の費用は、原則として、当期の期間費用とする」としているが、受注ソフトウェアを開発する受託側では「製造原価として処理することを妨げ」ず、またユーザ側においては「固定資産の購入に伴う付随費用が製造原価に加算されるのと同じ理論（連続意見書第三「有形固定資産の減価償却について」第一の四１）を適用すべきであると考えられる場合には、ソフトウェアの購入原価に加算すべきであることはいうまでもない」（pp.37-38）、としている。また、「保守費用の会計処理」において「顧客支援のためにかかった費用は、すべて発生時の費用であり、その理由は「収益的支出であり、資本的支出ではない」からである、としている。以上の通り、これは SFAS86 を踏襲した規定である。

世界初のソフトウェアに関する会計基準であるが、運用に関する規定はごく僅かしかない。Appendix C 52.「語彙」において「顧客支援（Customer support)」は「ソフトウェア製品を使用する顧客を援助するために企業が行うサービス。当該サービスには、据付援助、研修会、電話による応答、刊行物、現場訪問（on-site visits)、並びにソフトウェア又はデータ修正も含まれる。」と定義され、このコストは6.「関連する収益が認識される時点、若しくは、当該原価が発生した時点のいずれか早い時点において費用処理しなければならない」とされる。これは、パッケージ・ソフトウェアを提供するソフトウェア企業側の規定であり、ユーザ企業側の会計処理に関しては何も規定していない。

更に、ＡＩＣＰＡ（1998）は、1998年に公表されたＡＩＣＰＡ（American Institute of Certified Public Accountants）のＳＯＰ（Statement of Position）98－１「社内利用のために開発又は購入されるコンピュータ・ソフトウェアの原価の会計処理（Accounting for the Costs of Computer Software Developed or Obtained for Internal Use)」（現在の「FASB ACS 350-40 -Internal-Use Software」（以降「ＳＯＰ98－１」と略記））で、自社利用のソフトウェアを対象として公表されたものであるが、運用に関する規定は先に公表されたＳＦＡＳ86と多少異なっている。par.17でコンピュータ・ソフトウェア開発を「プロジェクト予備的ステージ（Preliminary Project Stage)」、「アプリケーション開発ステージ（Application Development Stage)」、「実現後／運用ステージ（Post-Implementation/Operation Stage)」の３つのステージに区分している。また、ステージ毎にそのステージで生じるプロセスが例示され、個々のプロセスに係るコストがどのステージに該当するかは、コストの発生したタイミングではなく、コストの性格により判断すべきとしている。そして、「実現後／運用ステージ」のうち「トレーニング（Training)」が運用に関するコストであるが、par.21並びに23で発生時に費用とすべきであるとされる。par.71でこのコストは、ソフトウェア開発コストではなく、それは企業がトレーニングされた従業員の継続的雇用をコントロールできていないために生じたのであり、便益をもたらす特定の将来の期間を特定することはできず、また償却期間も裁量によることになるからである、としている。

また、par.21並びに22でデータ移行について、新システムにより旧データの移行を行なう場合には資産とすべきであるが（この場合には、データもソフトウェアと看做すことになる)、旧システムから新システムへデータ移行を行なう場合には費用とすべきとしている。なお、両者の取り扱いの違いが如何なる理由によるかは不明である。 いずれにせよ、日本基準に先行するＳＦＡＳ86並びにＳＯＰ98－１にしても、部分的な規定であり、開発・保守と異なる運用の大半を占める通常運用に関しては何も規定がない。そして、運用には、「導入」という運用開始のごく限られた局面では一部資産性が認められ、資産計上されるが、基本的に元々の資産が「将来の経済的便益」を具現するのであって、「将来の経済的便益」をもたらす独自の追加的な効能はないと看做していることでは、日本並びにアメリカの基準は一致している、と言える。

３．システム運用改善に適合的な会計処理

（1）ソフトウェアの定義改訂案の再確認

運用改善に更に立ち入る前に、ソフトウェア会計基準におけるソフトウェアの定義が完備的でなく、改訂が必要であることを再確認しておく。ソフトウェア基礎論第１章で主題的に取り上げ、詳細に考察しているので、運用改善に大きく関係することに限定して、簡潔に再確認するに留める。１つは各種環境定義類に関することであり、もう１つはドキュメントに関することである。

日本のソフトウェア会計基準におけるソフトウェアの定義[369]は、対象として、各種環境定義類を包含していないが、欠かせないものである。これらは、プログラムと共に、「コンピュータに一定の仕事を行わせるため」に必要不可欠のものである。しかも、プログラムではない。従って、これをソフトウェアの定義に含めなければならないのである。

ソフトウェアの定義を上記のように改訂すると、明瞭に浮上してくることがある。プログラムの変更（追加、削除を含む）であれば、ソフトウェア保守の範疇だが、各種環境定義類の変更（追加、削除を含む）だけであれば、保守ではなく、運用の範疇で独自に行なうことがある（プログラムの変更に伴う各種環境定義類の変更は当然にあり得るし、その場合はソフトウェア保守で行なうことであるが)。例えば、機能の変更はなく、性能が低下してきたことを定義の変更により回復する等である。そして、単なる維持や原状回復ではなく、性能を向上させたならば、明らかに運用改善と言える。

もう１つは、ソフトウェアの定義にプログラムと併せて関連ドキュメントを挙げているが、その対象範囲が不明確であるが、運用マニュアル等、運用に関わるドキュメントも含まれるとしなければならない。

そうであるならば、運用マニュアル等の改善は、運用の範疇で独自に行なうことがあり、それによって利便性等を向上させたならば、明らかに運用改善と言える。

以上のことを再確認し、それを前提として、運用改善の具体相に分け入ることにする。

（2）システム運用改善の例示

システム運用改善に関する、幾つかの例示を行なうことにする。そして、各々の具体的な数値を推定して、次節でそれらに係る会計処理案を提示する段取りとする。

①第１の例示は、処理時間の短縮という運用改善に関するものとする。運用改善を具に示すため

---

[369] アメリカ基準のソフトウェアの定義はＦＡＳＢ(1985) 2．では、「コンピュータ・ソフトウェア製品、ソフトウェア製品及び製品という用語は、コンピュータ・ソフトウェア・プログラム、一群のプログラム及び製品の機能強化を含んで」いるというもので、やはり各種環境定義類は含まれていない。

に、詳細な想定事例とする。中堅企業で、本社の他に 20 の支社があるとする。同企業では、日締処理を行なっており、まず各支社単位に日締処理を行ない、全支社の日締処理が終了した後、全社の日締処理を行なう。各支社では定時間の 18 時に日締処理の準備に入り、当日必要なデータ入力等が全て入力済みか等を確認し、日締処理を起動する。日締処理が終了したら、正常終了を確認し、必要な帳票等を出力し、関係部署等に配布して、退社する。従って、日締処理担当者は、定時間後必ず残業をすることになる。準備に手間取り、処理起動が遅れれば、それだけ退社時刻も遅くなる。本社では、全支社の締処理の終了後に全社の締処理を行なうので、最も退社時刻が遅くなる。各支社の日締処理は、データ量等により所要時間は異なるが、単独で実行する場合は平均で約 20 分掛かるとする。従って、全支社分の処理時間は、処理多重度を１とすれば、順次処理起動しても、先行の支社の日締処理が終了するまでは待ちの状態になるので、単純計算で約 400 分（＝20 分×20）掛かることになるが、処理多重度をある値に設定しているので、約３時間で終わることになっている。それでも、日締処理担当者は定時間後毎日 30 分～３時間程度残業し（起動の遅速により）、本社担当者は全支社の日締処理終了後の全社日締処理（約 30 分）後なので、３時間 30 分程度の残業をする、とする。

この事態を運用改善しようとする場合、方策は幾通りか考えられるが、ここでは限定的な改善を想定する。第１の方策は、ハードウェアを高性能の上位機種にリプレースすることである。これによって、全支社の日締処理が大幅に短縮し、約１時間 45 分で終了することになったとする。これは明らかに運用改善である。上位機種のハードウェアを購入したとすれば、それを有形固定資産として計上することになる。運用改善という「将来の経済的便益」の具現として、具体的には、日締処理担当者の残業時間の短縮による残業代の削減等々に対応した会計処理である。

第２の方策は、処理時間の大半を占めるＤＢの更新処理を効率化するように、プログラムを改良することである。これによって、全支社の日締処理が短縮し、約１時間 15 分で終了することになったとする（ハードウェアのリプレースと併せて）。これも明らかに運用改善である。ソフトウェアの保守（改良）なので、ソフトウェアの資産計上とすることになる（現行基準の「著しい改良」には当たらない）。以上の２つの方策は、運用改善であるが、既存の会計処理の範疇に収まる方策である。

第３の方策は、筆者の言う各種環境定義類を変更することである。具体的には、処理多重度を上げること、並びにデータ量の多い支社の処理優先度を上げること、とする。ハードウェア並びにＯＳにより、処理多重度の上限がある場合があり、また支社数が 20 なので、処理多重度を最大 20 とすれば（且つ日締処理以外は実行されないとすれば）、起動次第全支社の日締処理が待ちとならず、処理は実行されるが、競合状態が発生し、必ずしも最大のパフォーマンスが得られるとは限らないことから、ベストの処理多重度は試行錯誤的にシミュレートしながら見出していく（机上の性能検証並びに実機による検証）。処理優先度も同様である。これらによって、全支社の日締処理が短縮し、約１時間で終了することになったとする（ハードウェアのリプレース並びにプログラムの改良と併せて）。第１、第２の方策に比べれば、短縮効果あるいは貢献度合として量的には少ないかもしれな

いが、これも明らかに運用改善である。運用改善という「将来の経済的便益」の具現（具体的には、日締処理担当者の残業時間の短縮による残業代の削減等々）に貢献するという、性格的には全く同様のことである。これも同様に資産計上すべきである、と考える。

根拠は、各種環境定義類の変更（処理多重度並びに処理優先度の変更）というソフトウェアの改善による運用改善だからである。現行基準のソフトウェアの定義には明示的に含まれていないが、各種環境定義類はプログラムと並び紛れもなくソフトウェアである。そして、プログラムというソフトウェアの資産性が認められている以上、同じソフトウェアである各種環境定義類の資産性が認められて然るべきである。そもそも購入や開発時のソフトウェアには、各種環境定義類はプログラムと併せて、含まれているものなのである。ソフトウェア開発において、運用設計等を行ない、作成している。その各種環境定義類の変更のみで運用改善に寄与したのであるから、プログラムの変更（ソフトウェアの改良）と同様の取り扱いがなされて当然であろう。

②第２の例示は、アールワークス編著（2012）で挙示されているパフォーマンスの改善という運用改善に関するものとする。１つの事例だけ挙示するが[370]、「システムは、「Ｗｅｂサーバー」「アプリケーションサーバー」「ＤＢサーバー」の３層構成である。Ｗｅｂサイト上でイベント[371]を頻繁に行うため、アクセス集中対策として、余裕を持った作りになっている」。「Ｗｅｂサーバーとアプリケーションサーバーをそれぞれの層で独立に負荷分散、データベースも業務データ用とセッション[372]情報用を別々に用意している」。「業務データ用ＤＢはクラスタリング構成[373]、セッションデータ情報用ＤＢは２台のサーバーによるミラーリング構成[374]であった」。しかし、「新規にサービスを始めたところ、サービスに遅延が起き、セッションＤＢがボトルネックになり、応答が返せなくなっていた」。原因は、２台のセッションＤＢが、「マスター／スレーブとも書き込みが終わるまで、処理が完了しない」「同期モードのミラーリングを行っていた」ので、「スペックの劣る１世代前の余剰サーバーを再利用していた」「スレーブＤＢのパフォーマンス不足に引っ張られる形で、応答遅延が起きていた」のである。「さらには、１時間ごとにトランザクションログの切り捨てや圧縮処理[375]を行っている」が、特に「圧縮処理はＩ／Ｏに大きな負荷を与える処理」なのである。

そこで、運用改善の「当面の処置として」、セッションＤＢのミラーリングを止め（設定変更）、

---

370 この事例に関しては、アールワークス編著(2012)pp. 144-146 から引用する。

371 操作、挙動・状態の変化、あるいはそれが発生する信号。

372 接続(ログイン)～切断(ログオフ)までの一連の操作や通信。

373 複数(多数)のサーバをあたかも１台の大きなサーバであるかのように構成し、個々のサーバで障害が発生しても支障のないようにする構成。

374 マスター(主)とスレーブ(従)のサーバを同一の複製状態とし、マスター・サーバに障害が発生しても、スレーブ・サーバで処理を続行できるようにする構成。

375 処理の稼働情報を一定期間保持するが、容量の都合で、古い順に切り捨てたり、空白や同一内容の繰返しを１個の空白や内容とその繰り返し回数とすること等で容量削減をしたりする(圧縮は、圧縮条件に該当するかという探索とデータ編集に一定の処理時間を要する)。

「マスターサーバー単独での処理に切り替えた」。また、「トランザクションログの切り捨て」はこれまで通りとするが、高負荷で「１時間ごとに行うような処理でもない」「圧縮処理を１日１度のみとした」（設定変更）。「このようにして、システムは安定稼働に入った」、とのことである。これらは、いずれも処理負荷を軽減する処置であり、筆者の主張する各種環境定義類の設定変更による運用改善に該当する事例であると捉えている。

なお、トランザクションログの圧縮処理はこれでよいが、セッションＤＢのミラーリングを止めたことは、同書も言う通り「当面の処置」であり、マスターサーバーと同等スペックのスレーブサーバーを後日用意し、ミラーリング構成とすることが一層の運用改善となるであろう。

十全な運用改善にはハードウェア並びにプログラムの改良が必要であるが、次善の策として環境定義の設定変更が寄与することを如実に示している、と言える。

③第３の例示は、稼働率の向上という総合的な運用改善に関するものとする。「民間向けＩＴシステムのＳＬＡガイドライン」では、稼働率は(1) ＩＴサービスのうちの①セキュリティサービスのサービスレベル項目として先頭に掲げられている[376]。だが、サービス対象の範囲として「サービス全体」[377]とされているように、まさしく総合的に達成するものである。「評価方法または測定方法」として、「稼働率（%）＝（月間の総時間－月間の累計サービス停止時間）/月間の総時間（計画停止時間を除く）」が掲げられており[378]、サービスレベル値の「サンプル値」として、「レベル０：規定無し」「レベル１（下位レベル）：95%以上」「レベル２（中位レベル）：99%以上」「レベル３（上位レベル）：99.5%以上」が掲げられている[379]。それらのレベルの稼働率がどの程度の事態を意味しているかを想定する際に、筆者はむしろ365日24時間フル稼働の場合に逆にどの程度の非稼働(停止)が許容されるかと捉えた方がわかりやすいと考えるので、試算してみる。レベル１の場合は 5%未満の非稼働は許容されるので、438 時間（＝365*24*0.05、18.25 日）未満はシステム・ダウンしてもよいということであり、レベル２の場合は 1%未満の非稼働は許容されるので、87.6 時間（＝365*24*0.01、3.65 日）未満はシステム・ダウンしてもよいということであり、レベル３の場合は 0.5%未満の非稼働は許容されるので、43.8 時間（＝365*24*0.005、1.825 日）未満はシステム・ダウンしてもよいということである。だが、いずれもハードウェア障害、ソフトウェアのバグ又は脆弱性、利用者の誤操作、外部からのウイルス等によるセキュリティ攻撃、自然災害等を含めたあらゆる障害要因に対処してのことであるから、それほど達成することは容易ではない。

---

376 ＪＥＩＴＡ(2012)p. 68, 194

377 但し、ＪＥＩＴＡ(2012)ではあくまで「セキュリティサービス全体」(p. 68、傍点引用者)という意味であるが、筆者は例え項目分類には収まりにくくなっても、稼働率はセキュリティに限定するものではなく、敢えて限定なしに対象範囲としてサービス全体と捉えた方がよいと考える(意図的な拡大解釈ないし改釈)。

378 ＪＥＩＴＡ(2012)p. 68

379 同書 p. 195

まずレベル１からスタートするとする。次に、翌年度等に運用改善としてレベル２にサービスレベルを引き上げるとする。続いて、翌年度等に更にレベル３に引き上げるとする。そのためには、より高機能・高性能且つ堅牢なハードウェアへのリプレース、システム構成の二重化（待機系の保持）、ソフトウェアの品質改善、利用者への教育徹底、強固なセキュリティ対策、遠隔地への運用センターの設置等を、レベルに応じて継続的に実施していかなければならない。その中には、システムの監視・調査・分析を行ない、それに基づいて随時あるいは定期的に最適な状態となるように各種環境定義を設定変更（調整）することも含まれる。

④第４の例示は、運用マニュアルの改善による運用マニュアルの利用率の向上という運用改善に関するものとする。運用マニュアルは、ソフトウェア会計基準におけるソフトウェアの定義でソフトウェアと認められているソフトウェアの「関連文書」である（ソフトウェア開発におけるプログラムを制作するための設計書だけが「関連文書」ではないはずである）。ソフトウェアの保守におけるプログラムの改良と同列的な意味で、「関連文書」の改善が位置付けられるはずである。この場合は、運用部門において、運用マターの取り組みとして、システムを利用者がもっと手軽に簡便に多く利用しやすいように、ひいてはそれがシステムの利用率を向上させることになることを企図して、運用マニュアルの改善を行なったとする。それによって、確かにマニュアルの利便性を高め、利用が増大したとする。利用者へのアンケート調査でも良好な結果が出たし、運用マニュアル改善前後で他に利用率に影響する要因の変化は見られなかったが、利用率に有意と言える増加が確認できた。そうであれば、運用マニュアルの改善によって運用改善が達成できたと言えるであろう。

（３）システム運用改善に適合的な会計処理

システム運用改善に係る会計処理は、これまでハードウェア並びにソフトウェア（そのうちのプログラム）の改良に関しては資産要件を充足すれば資産計上がなされたが、それ以外は全て費用処理されてきた。それに対して、上記２で例示した運用改善はその多くが資産計上をすることが妥当なものである、と筆者は考える。例示に沿って、個々に検討する。

①の例示では、第１の方策におけるハードウェアは購入費（導入費を含む）が 5,000 万円だったとすれば、それが資産の測定金額となる。第２の方策におけるソフトウェア（プログラム）の改良は、外部委託を行ない、委託費が 600 万円（＝100*6（人月単価：100 万円、工数：6 人月））だったとすれば[380]、それが資産の測定金額となる。第３の方策は、現行の会計実務では運用業務に係る人件費又は委託費として費用処理されるが、筆者は運用改善に寄与した各種環境定義類の改善という

[380] 外部委託としたのは、金額算出を簡便に行なうためだが、運用は実態的にも委託とすることが多いので(契約形態としては準委任や派遣とする場合もあるが、実質的な作業としては変わりはなく、費用算出方式も他に幾通りかあるが、採算のベースとなるのが人月単価×工数であることは変わらない)、単なる便宜上のことではない。また、社内で行なう場合には社員の人件費あるいは間接費の配賦等で算出要素(項目)が多少異なるが、大筋は変わらない。

ソフトウェアの改善として資産計上することが妥当であると考える。資産の認識としては、プログラムと全く同様である。資産の測定としては、ソフトウェアの開発・保守一般と全く同様であり、運用改善（成果物としての各種環境定義類の変更）に費やした主として人件費（委託費）が対象となる。また、運用業務において運用改善に費やした作業量等は、作業報告書等から識別し、測定することは比較的容易に可能であるから、特段の問題はない。例えば、運用業務を全て委託していたとし、年間契約で9,600万円（＝80*120（人月単価：80万円、常駐者：10名、工数：120人月））が委託費だとする。そのうち、運用改善に3名が2ケ月各0.5人月費やし工数合計が3人月だったとする（残り各0.5人月は通常運用に従事）。そうであれば、240万円（＝80*3）が運用改善に掛かったのであり、それが資産の測定金額となる。残りの9,360万円は通常運用の費用として費用処理する。

②の例示では、アールワークス編著（2012）には作業量や費用に関しては全く掲示されていないので、筆者が推定する他ないが、1つ目の事例はミラーリングと圧縮処理の設定変更に限定すれば、上級技術者1名（人月単価：120万円）が1ケ月従事する程度であろうから（工数：1人月）、費用は120万円（＝120*1）であり、それが資産の測定金額となる。（なお、アールワークス編著（2012）の文脈からすると、コンサルテーションを含めた総合的なサポートを受託し、その一環として対応したように受け取れるが、単純化のために、独立的な作業に読み替えた。）

③の例示では、稼働率のレベルアップに関して、あくまで各種環境定義類の設定変更に限定すると、サービスレベルをレベル１からレベル２に引き上げるには大凡の推定で、技術者5名（上級～下級技術者、平均人月単価：100万円）が6ケ月専任で従事する必要があろうから、費用は3,000万円（＝100*6*5）であり、それが資産の測定金額となる。レベル３に引き上げる場合には、技術者8名（上級～下級技術者、平均人月単価：100万円）が9ケ月専任で従事する必要があろうから（工数合計：72人月）、費用は7,200万円（＝100*72）であり、それが資産の測定金額となる。

④の例示では、運用マニュアルの改善に調査・分析を含め技術者1名が3ケ月専任で従事することになろうから（工数：3人月）、費用は300万円（＝100*3）であり、それが資産の測定金額となる。

例示による算定は以上だが、例えば「民間向けＩＴシステムのＳＬＡガイドライン」における項目詳細は480超であるから、システム運用改善に繋がる事象は多様且つ多方面に亘るものがあり、これまで運用業務全般の費用処理に含めていたものの中には精査すれば、運用改善に該当することは少なくないのではないかと推察される。それらを識別し、会計処理を異なるものとすれば、ソフトウェア資産の実態を的確に表現することになる。また、注記等で説明を加えれば、システムへの取り組み姿勢をより鮮明に開示することになる。

システム運用に関して、ハードウェアやソフトウェアの購入を除けば、これまで専ら人件費等を費用処理することが当然視されていたことに対して、筆者はかねがね疑問を抱いていた。一般的に

も、システム運用は「縁の下の力持ち的」[381]な役割と看做され、技術者にとっても開発業務に比べれば「達成感」のない、「使えて当たり前、トラブルが起きれば文句を言われる」という余り報われない業務と捉えられることの多いものであった。しかし、ＩＴ化が進み、ＩＴの利用が不可欠になると、システム運用こそが中心的な有り様となってきたと言うべきである。だからこそ、普及はまだそれ程進展していないが、運用ベースのＩＴＩＬが徐々に採用されるようになってきたのである。

そこで、本章はソフトウェア会計基準の分類で言えば「自社利用」を対象として、システム運用改善を主題的に取り上げ、運用においても改善がなされており、それに適合的な会計処理を明確にした。具体的には、ソフトウェア会計基準におけるソフトウェアの定義が完備的でなく、狭義のソフトウェアとしてプログラムしか対象としていないが、もう１つの大きな要素として各種環境定義類があることを指摘し、それを定義に含めることで運用改善がこれまでとは異なる意義を有することになり、その場合はプログラムの変更と同様の会計処理を行なうべきであることを提言した。また、ドキュメント（運用マニュアル等）を改訂して、利用者の利便性を向上させたならば、その場合も同様の取り扱いをして然るべきではないか。このように運用改善を捉え直し、会計処理を見直すことは、運用改善を一層促進し、またシステムに関する企業努力の促進にも繋がり得るのではないだろうか。こうしたことへの寄与の一助となればと、願っている。なお、筆者のソフトウェア分類とその各々のライフサイクルへの関与からすれば[382]、運用を行なうのは、市場販売のソフトウェアを購入した利用企業とサービス提供のソフトウェアの提供企業と自社利用のソフトウェアの利用企業であるが、それらの運用改善に係る会計処理は特に異なることはなく、同様に取り扱えばよい。資産の性格として収益の直接的源泉か間接的源泉かという違いはあるが、会計処理としては同一である（クラウド事業者等の運用に関しては、ソフトウェア実務としては興味深く、更に詳細を知っていきたいが、会計的に異なる取り扱いをする必要性はない、と考えている）。

むしろ、異なることがあるとすれば、本章では運用改善に関して、各種環境定義類の変更並びにドキュメント改善という事象に限定して会計処理を提言しているが、運用改善には他の手段によることも多く、例えば人的なサービスの改善といったことも少なくない（教育により利用を促進することや、サービスデスクの対応の良さにより利便性を高める等）。それらへの拡張は今回見送ったが、本章とは異なる別の論理構制を必要とするからである。ソフトウェアという無形資産とは異なる、いわゆる「インタンジブルズ」（intangibles）あるいは自己創設のれんに関わるものなので（組織能力（capability）等）、そもそもの資産性を論証することから始めなければならない。そうしたことへの拡張は、今後の課題と考えている。

[381] アールワークス編著(2012)p.179
[382] ソフトウェア基礎論第３章で提示しているので参照。

本論

第８章　クラウド・コンピューティング

１．問題の設定

２．クラウド・コンピューティングの概観

３．会計的問題（一）：事業者側における制作目的別分類の適合性

（1）現行の会計処理

（2）ＪＩＳＡの論点整理

（3）会計実務の調査（一）

（4）調査結果を踏まえた考察

４．会計的問題（二）：利用企業側におけるプライベート・クラウドの資産性

（1）現行の会計処理

（2）会計実務の調査（二）

（3）調査結果に関する問題整理

（4）調査結果を踏まえた考察

補遺Ⅰ　調査票　　クラウド・コンピューティング調査票（事業者向け）

補遺Ⅱ　調査票　　クラウド・コンピューティング調査票（利用企業向け）

第８章　クラウド・コンピューティング

クラウド・コンピューティング（Cloud Computing、以下特にフル表記が必要な場合を除きクラウドと略記）は、2000 年代後半にＩＴ分野で大きな話題となったが、今でも発展途上にあり、進化を続けている。1943 年にＩＢＭ社のトーマス・ワトソン（Thomas Watson、 Jr.）が「コンピュータの世界市場規模は５台程度だ（I think there is a world market for about five computers.）」と予言したが[383]、2006 年にサン・マイクロシステムズ（Sun Microsystems（当時））のグレッグ・パパドプロス（Greg Papadopoulos）はそれを承知の上で挑発的に、「世界は５台のコンピュータだけしか必要としていない（THE WORLD NEEDS ONLY FIVE COMPUTERS）」と酷似した発言を行っている[384]。言うまでもなく、その含意は全く異なっている。片やコンピュータ創生期に極めて高価且つ弾道計算等特定の用途にしか使われないコンピュータに対する将来性をネガティブに捉えた発言であり、片や広汎な普及を経たその先にクラウドに集約されていくであろう将来を予見した発言である。その間に、ＩＴの 60 年を超える歴史（電子回路の高集積化、計算センター、ＴＳＳ（Time Sharing Systems：時分割システム（１台のコンピュータを複数台の端末から「同時」利用可能とするシステム））、パソコン、ネットワーク化、等々）が介在しており、まさに隔世の感がある。そして、クラウド事業は、市場規模的には 933 億円程度[385]で、企業が自社で（委託を含む）システムを開発し運用する態勢を大きく変えるほどの潮流にまでは到っていないが、拡大基調に変わりはない。但し、パパドプロスの「予言」は外れるであろう。例えて言うならば、公共交通機関がどれほど安価で利便性を高めようとも、自家用自動車の利便性や保有する満足感や充足性を知ってしまった以上、それを手離すことはないであろうように、である。だからこそ、クラウドは、電気・ガス・水道あるいは公共交通機関や電話のような「ユーティリティ」ないしそれに近い[386]パブリック・クラウドから、これまでのアウトソーシングあるいはその発展形といった形態、更に実質的には自社保有と変わらないプライベート・クラウドに到る多様な形態の広がりを現出させているのである。

そして、その事業の在り様や利用技術に関しては、インターネット上はもとより、マスメディアや書籍等でも多くのことが取り上げられており、様々な関心から意義深いアプローチがなされている。ところが、筆者が専攻する会計分野ではクラウドに注目した議論は現在までのところ少ない。

---

383 日経ＢＰ社出版局編(2010)p.5

384 羽深・志田・田中(2011)p.13

385 調査会社ＩＤＣ JAPAN による、2012 年のパブリック・クラウド市場に関する調査を参照(http://www.publickey1.jp/blog/13/44893320173idc_japan.html)。プライベート・クラウドに関する同種情報は未見。なお、ＩＴサービスは 4.9 兆円、パッケージソフトは 2.4 兆円である(http://www.itmedia.co.jp/enterprise/articles/1301/31/news064.html)(安延(2009)pp.140,235をも参照)。

386 やや曖昧な表現に留めているのは、ユーティリティには必需品又は必需サービスという特性があり(従って法規制を伴っている)、クラウドをそれと同等と見做してよいかどうかはなお慎重を期する必要があると考えているからである。

クラウド事業はスケール・メリットを追求できる有望な事業であり、利用企業もその恩恵を享受できるのであるから、それらの経営管理やコスト／ベネフィット分析を行なう管理会計のアプローチは意義深いものである。それに対し、筆者はアンカー的役割を担う財務会計からのアプローチを本章では採用し、その視点からクラウドを捉えることにしたい。

そう定位した視点からまず言えることは、ソフトウェア会計基準が、クラウドが出現する以前に設定されたものであり、十分に適合的でないことは致し方ないことであるが、その点を主題的に取り上げているものは、筆者の知る限りＪＩＳＡ(2010)以外には見当たらないことである。一般的にはクラウドの事業や技術に注目が集まるのは当然のことではあるが、財務会計の立場からの関与が積極的になされてもよいのではないかと考える。

１．問題の設定

筆者が財務会計的な視点でクラウドの動向を見渡すと、クラウドが惹起する会計的な問題が大別すると２つある、と思える。１つは、ソフトウェアに係る会計基準の制作目的別分類により、市場販売目的と自社利用のうちのサービス提供目的のソフトウェアでは大きく会計処理が異なるので、クラウドにおいて同一のソフトウェアを両方の目的で利用している場合、実態とは異なり、択一的に何れかの会計処理をせざるを得ない、という問題である。しかも選択規準はないので、恣意的となり、財務情報が大きく異なったものになる、という問題である[387]。

もう１つは、それまで自社保有の環境でシステムを構築・運用していた場合には資産計上をしていたものが、クラウド利用に切り替えると、利用料金の支払いという費用処理に変更することになる。しかし、後述するプライベート・クラウドの場合には、例え形式的には単なる利用の形態であっても実質的には利用企業の保有資産として計上すべき事態ではないのか、という問題である[388]。

ソフトウェアの業界団体として機敏に対応しながらも、ＪＩＳＡの調査報告は、１つ目の問題に関して現行会計基準ベースで整理を行なうに留まり、後述するように筆者の想定している問題に十分対応し得ていない。もう１つの問題は、取り上げられていない。しかし筆者は、発展途上故に流動的なため、判断が難しくはあるが、会計基準の改訂等を含め会計的な新たな対応が必要か否か、

[387] 会計基準の構成(定義―認識―測定・評価―開示)で言えば、ソフトウェアの「制作目的別分類」という定義ないし認識の一部を主として問題とし、その規定に則った会計処理を含めて取り上げることになる。

[388] 会計基準の構成で言えば、認識(そのうちの特に「資産要件」)を主として問題とし、その規定に則った会計処理を含めて取り上げることになる。なお、これと関連して、2007 年のリース会計基準の改訂におけるファイナンス・リースの取扱いを想起させる(資産計上に関する例外処理の廃止)。それ以前からリースに関してよく言われる「所有から利用へ」ということもである。但し、リース会計には固有の歴史的経緯や諸論点があり、一方クラウドにはそれとは異なる歴史的な経緯(マシンがまだ高価だった時代における計算センターでの時間貸し等)や相違(利用時の事業者側の継続的な関与等)があるので、本章ではリース会計に包含させたり、直接的な依拠・援用は行わず、あくまで独立的な考察並びに論理構成としている。

俎上に乗せ、検討を行なうべきであると考える。そのため、２つの問題に関して、クラウドの事業者並びに利用企業に対し実態調査を行ない、その調査結果を踏まえて、より適合的な会計処理案を考案し提示する[389]。

本章の構成概要は、次の通りである。問題の設定を受けて、第１に、会計的考察に必要な範囲でクラウドを概観する。第２に、会計的問題（一）として、事業者側の会計処理において、制作目的別分類が適合的か否かという観点から、まず現行の会計処理並びにＪＩＳＡの論点整理を確認し、次に筆者の行なった会計実務の調査を報告し、更にその調査結果を踏まえた考察を行なう。第３に、会計的問題（二）として、利用企業側の会計処理において、利用企業の保有資産と見做すのが適切ではないかという観点から、まず現行の会計処理を確認し、次に筆者の行なった会計実務の調査の報告並びに問題の整理を行ない、それらを踏まえた考察を行なうことである。

それは、技術革新の激しいソフトウェア分野において新たに発生した事象・事態に対して、タイムリーに、且つ実態に即した会計的な対応を提案することで、実務的な混乱や齟齬を回避し、事業活動の発展に寄与する意義を有するものと考える。

２．クラウド・コンピューティングの概観

クラウド・コンピューティングは、2006 年にグーグルのエリック・シュミット（Eric Schmidt）が命名したようである[390]。

クラウドは、「パブリック・クラウド」（インターネットを介して各種コンピューティング資源をサービスとして顧客に提供する）と、「プライベート・クラウド」（イントラネットに基づく企業内ネットワークを使って同様のサービスを提供する）と、「ハイブリッド・クラウド」（両者の混合型）に分類することができる[391]。それらは、拡大的に包摂すれば、アウトソーシングやＡＳＰ（Application Service Provider）[392]やホスティングといった「従来から存在するサービスにも当てはまり」、実際にそれらを「「クラウドサービス」と呼び変える会社も少なく」ない[393]。これにはハウジングを加えてもよいであろう。

サービス形態の分類としては、ＳａａＳ（Software as a Service）、ＰａａＳ（Platform as a

---

389 なお、本章は現行会計基準をベースに、経済実態（クラウド）に適合的な部分的な改訂を提案するものであり、その拠って立つ会計観等の論議には立ち入らない。

390 日経ＢＰ社出版局編（2010）p. 7

391 同書 p. 45。当該箇所では、ＩＢＭ社が 2008 年の公表資料で提示した分類として紹介している。

392「ネットワーク上でソフトウェアを提供する」事業者のことである（ＪＩＳＡ（2010）p. ｉ）。これに対して、ハウジング（housing）は利用企業が運用環境を事業者から借りて運用は主として自らが行なうこと、ホスティング（hosting）は運用も事業者に委託することである。事業者はいずれの場合も複数の利用企業に対して定型的な環境並びに運用形態を設定して業務を請け負う。アウトソーシング（outsourcing）が利用企業と事業者の個別の独自的な請負であるのとの違いである。

393 同書 p. 257

Service)、ＩａａＳ（Infrastructure as a Service）に分類することができる[394]。なお、本章にとってはソフトウェアが関係するＳａａＳが主要な関心事である。

クラウドを成り立たせているあるいは関連する技術として、仮想化[395]が重要であり、大半のクラウド事業には欠かせない技術基盤である[396]。また、「利用企業にサーバー仮想化技術を導入するサービスを「プライベートクラウド」と称するメーカーも増えて」[397]おり、プライベート・クラウドにとって仮想化は欠かせないものである。

クラウドの利用に関する料金体系は従量課金制又は定額制であるが、その従量とは利用時間並びにサーバー・ストレージ・ネットワークの使用量等である。

企業の利用状況は、経済産業省(2012)によると、平成22年度の「クラウド・コンピューティング利用率」[398]は前年度の9.7%から16.0%まで上昇し、平成18年度の調査開始以来最大の上昇幅を示した、とのことである。利用形態をみると、ＳａａＳが76.5%で最も多く、ＩａａＳが15.8%、ＰａａＳが14.2%、その他が9.0%で、「利用形態が多様化していることがうかがわれる」[399]、としている。

ＪＵＡＳ(2012b)[400]によると、今回調査から追加した「プライベートクラウド」は「導入済み」が17.9%に上り、既に６社に１社以上が導入している計算となる」[401]、としている。但し、「「プライベートクラウド」の定義は曖昧で、仮想化したサーバーをデータセンターで運用しているだけで、「プライベートクラウド」と認識しているケースも多い。それが「導入済み」の割合をあげている可能性がある」[402]、と断り書きを付している。なお、「クラウドへの期待は、「ＩＴ資産を持つ」から「サービスを使う」への変化の現れ」[403]、と総括しているが、これに対する疑問は改めて取り上げる。

３．会計的問題（一）：事業者側における制作目的別分類の適合性

---

394 同書pp. 10-13。林(2012)pp. 24-25も参照。

395 仮想化とは、物理的な環境上に複数の独立的な仮想環境を構築し運用することであり、それにより利用企業は他の利用企業とは独立的に自ら特有の環境での利用が可能になる(倉西編(2012)pp. 16-19)。

396 尤もGoogleはサーバー仮想化技術を殆ど使っていない(但しネットワークの仮想化は利用している)(同書p. 258)。

397 同書p. 262

398 経済産業省(2012)pp. 45, 46。「クラウド・コンピューティング関連費用が「発生した」と回答した企業の割合」のことである。なお、調査対象期間は2010年4月1日～2011年3月31日までの１年間、対象企業は資本金3、000万円以上かつ総従業者50人以上の企業の中から無作為抽出により9500事業者で、回答企業は4832社、回答率は50.9%である。

399 同書p. 50

400 調査期間は2011年10月29日～11月21日、調査対象は東証一部上場企業とそれに準じる企業の計4000社で、回答数は1039社(有効回答率：26%)である(同書p. ix)。

401 ＪＵＡＳ(2012b) p. 4

402 同書p. 4

403 同書p. 4

（1）現行の会計処理

クラウド・コンピューティングに係る会計処理で問題となり得ることの１つは、１．の「問題の設定」で簡単に触れた通り、ソフトウェアの制作目的別分類並びにそれに伴う会計処理の相違である。つまり、市場販売目的と自社利用では資産認識の要件並びに開発原価の範囲が大きく異なっている。これまでも、筆者は問題視していたが、決定的な問題とはなっていなかった。ところが、今日的に、クラウドが普及拡大するに連れて、改めてこの問題が惹起されることになった、と筆者は捉えている。

具体的には、ソフトウェア企業がソフトウェアを開発し、パッケージ・ソフトウェアとして販売するとする。一方で、クラウド事業を行ない、他社製品並びにオープンソース[404]と共に、自社制作の上記のソフトウェアをリソースとしてサービスを提供したとする。前者は会計基準に拠れば市場販売目的のソフトウェアであり、後者は明らかに自社利用のサービス提供目的のソフトウェアに分類される。別々のソフトウェアであれば、理論的な不整合に目を瞑れば、会計実務で大きな問題は生じなかったが、同一のソフトウェアが対象である場合には、問題となる。市場販売目的と看做せば、大半は費用処理し、「最初に製品化された製品マスター」完成後の軽微な部分を資産計上することになる。自社利用のサービス提供目的と看做せば、資産要件を充足すれば、ほぼ全てを資産計上することになる。しかし、両方の目的を兼備している場合には、どうすればよいのであろうか。いずれかを「主」と看做して、それに則った会計処理をする他ないであろう。しかし、開発計画において、あるいは開発完了時点で、いずれを「主」又は「従」とするかはそれほど明確であろうか。あるいは、当該企業の意図ないし判断と、事業遂行における利用の実態的な「主」「従」が異なった場合、その齟齬を会計的に補正する手立てはないはずであり、実態と適合しない会計処理となる。それを、筆者は問題であると考える。そもそも、同じく収益の直接的な源泉であるにも関わらず、市場販売目的と自社利用のサービス提供とを異なる分類とし、大きく異なった会計処理を行なうことを規定した会計基準に問題があったのであるが、それが図らずも実態的に露呈してきた、ということではないか。これが、１つ目の問題である。

（2）ＪＩＳＡの論点整理

ＪＩＳＡ(2010)は、次のように問題提起を行なっている。「「研究開発費等に係る会計基準」の公表当時は、ＡＳＰ市場の黎明期にあったといえよう。／このため、ＡＳＰのビジネスは、上記の会計基準等においてビジネス特性に基づく独立した分類項目として取り扱われず、自社利用のソフトウェアの範疇での記述に留まっている」[405]。「この結果、ＡＳＰ事業用ソフトウェアの会計処理は、

[404] オープンソースに関しては、本論第５章で主題的に取り上げているので、参照。

[405] ＪＩＳＡ(2010)p.5。なお同調査報告では「ＳａａＳ・ＡＳＰ」を主として用い、クラウド・コンピューティングは時々使われる程度だが、今日の慣用では前者を包含した後者が一般的であり、

当該事業を手掛ける企業内部の個別の判断に留まることとなり、次章の調査結果のとおり、自社利用のソフトウェアで会計処理を行う企業が一番多いものの、市場販売目的のソフトウェアとして扱う企業もまた多い結果となったのではないかと考えられる」[406]。

調査結果として、具体的には、サービス提供形態としては「一般的なＡＳＰサービス」が 85.7%、「一般的なＡＳＰサービスと通常のパッケージ販売の混合型」が 28.6%、「パッケージソフトの期間利用型」が 14.3%、「パッケージソフトの期間利用型と通常のパッケージ販売の混合型」が 4.8%、「その他」が 4.8%であった（複数回答）[407]。「一般的なＡＳＰサービスにおいて適用している会計処理」は、「自社利用」が 44.5%、「市場販売目的」が 36.1%、「併用」が 19.4%であった。また、「パッケージソフトの期間利用型」の会計処理は、「市場販売目的」が 66.6%、「自社利用」が 16.7%、「併用」が 16.7%であった[408]。「ＳａａＳ・ＡＳＰ事業用ソフトウェアを「市場販売目的のソフトウェア」と分類している企業は 13 社あるにも関わらず、資産計上時期について研究開発段階の終了時点を資産計上時点としていると回答した企業がゼロであったことは特筆される」[409]、としている。そして、「会計処理上の論点・課題」として、「ＳａａＳ・ＡＳＰにおける自社制作のソフトウェアの資産計上（市場販売目的または自社利用目的）の判断基準が監査法人でも明確でないこと」、「今後、クラウド、ＳａａＳ・ＡＳＰ含めソフトウェア業界におけるサービス提供は多種多様になり、その中核を担うであろうＳａａＳ・ＡＳＰにおいては、より明確な会計基準を設けてもらいたい」[410]、という回答があったとのことである。

ＪＩＳＡ(2010)では、調査結果を踏まえ、暫定的に、「一般的なＡＳＰサービス」は自社利用のソフトウェア、「パッケージソフトウェアの期間利用型」は市場販売目的のソフトウェアに分類し、それに則った会計処理を行なうべきとする[411]。「暫定的」としたのは筆者であるが、それは論点整理の「策定方針」における前提が現行基準等の「枠内で設定すること」であり、また「策定方針の議論のなかでは、現行の会計基準上の取り扱い（「研究開発費等に係る会計基準」（実務指針とＱ＆Ａを含む）及び関係税法・通達）にとらわれずに、あるべき会計処理を追究し、その結果を事業者団体の立場から提言すべきことも検討したが、事例の収集が不十分であることと時間的な制約から見送

本章はそれに従い、限定が必要な際には特定の用語を使うことにする。また、ＪＩＳＡはＳａａＳとＡＳＰを同義と捉え(p. 5)、ソフトウェアの業界団体としてクラウドのうちソフトウェアに関わることに着目して「ＳａａＳ・ＡＳＰ」という用語を主として使っている。

406 同書 p. 5。調査期間は 2009 年 12 月 21 日～翌年 1 月 20 日、回答数は 64 社(回答率 : 10. 9%　内、上場企業 34. 8%)である(p. ⅰ)。

407 同書 p. 13

408 同書 p. 21

409 同書 p. 22。なお、「ただし、調査票の回答の選択肢に「研究開発段階の終了時点」を用意していなかったこともこのような回答結果になった要因に含まれているものと想定している」(p. 22)と補記している。

410 同書 p. 12

411 同書 pp. ⅱ, 26-27

った」[412]、からである。従って、「事例の収集」が十分になされ、「時間」を掛ければ、もっと異なった「提言」となった可能性があることになろう。なお、「暫定的」ではあっても出された見解は、依然として大きな難点を抱えている。個々のソフトウェアが「一般的なＡＳＰサービス」又は「パッケージソフトウェアの期間利用型」に截然と区別できる場合はよいが、「一般的なＡＳＰサービスと通常のパッケージ販売の混合型」等の場合にはどのような会計処理を行なえばよいか、何も触れておらず、未解決のままとしているからである。

考察を進めるために、筆者はクラウドに関する実態調査を行なうことにした。そこで次に、調査内容を整理し、それに基づいて更に考察を進めることにする。

（3）会計実務の調査（一）

筆者の問題意識に基づき、クラウドの実態調査を行なった。本論文において、他の章と異なり、本章でこのような調査を行なった理由をまず説明しておきたい。他の章は、文献並びに筆者が面識のあるソフトウェア業界関係者からの情報提供により、十分に考察を進めることができたが、本章の問題に関しては、関心の所在が異なるため、インターネットやマスメディアあるいは書籍や業界誌等からの情報だけでは不十分であり、自らが調査を行なうことにしたのである。調査の概要は、次の通りである。調査期間は 2013 年 2 月 20 日～同年 3 月 31 日、調査対象は、クラウド事業者 83 社、利用企業 152 社の計 235 社である。対象選定規準は、クラウド事業を行っている企業並びに利用企業で、ＩＲ部門等があり、問い合わせ可能な企業とした（具体的な企業としては『日経コンピュータ』等の業界誌のクラウド関連記事並びにインターネット検索で特定した）。調査依頼はメール又は電話、１社は郵送で行い、調査に応じる又は調査内容を見た上で判断すると回答があった企業に調査票をメールで送付し、１社は郵送、３社は企業側の希望により対面調査を行なった。調査票は、「事業者向け」（33 項目）と「利用企業向け」（21 項目）の２種類であり、前者は調査（一）と（二）の両方、後者は（二）に関するものである。調査票はソフトウェア要求工学の手法[413]を使って作成し、調査項目は事業態様又は利用態様を始めとして、締結する契約における会計に関わる事項、更に実際に行なっている会計処理内容についてとし、可能な限り選択式として、該当するものがない場合には自由記述式とする方式で回答できるものとした。

回答数は、事業者７社、利用企業 15 社の計 22 社、回答率は全体で 9.4％であった。回答企業の概要は次の通りである。業種分類は a:建築・土木、b:素材製造、c:機械器具製造、d:商社・流通、e:金融、f:重要インフラ、g:サービス、h:ＩＴとし、売上規模（2011 年度単体ベース）はＡ:100 億円未満、Ｂ:100 億円以上～500 億円未満、Ｃ:500 億円以上～1000 億円未満、Ｄ:1000 億円以上～

412 同書 pp. 7-8

413 ソフトウェアを開発する際に、その全ての基礎となる要求を「抽出」→「分析」→「仕様化」→「妥当性確認」というステップで確定させていくが、その具体的な手順等が定式化されており、その中に「ユーザー調査」の手法も含まれている(Gottesdiener(2005)邦訳 pp. 80-86)。

1兆円未満、E：1兆円以上とした[414]。事業者に関しては、業種は全て同じｈで、売上規模はＡ４社、Ｄ３社、そして上場６社、非上場１社である。利用企業に関しては、業種はｄ１社、ｅ４社、ｆ１社、ｇ５社、ｈ４社、売上規模はＡ８社、Ｂ２社、Ｃ２社、Ｄ２社、Ｅ１社、そして上場６社、非上場９社である。全く返信のなかった企業は不明であるが、断わりの返信のあった企業の大半は、「企業機密なので応じられない」あるいは「個別の調査には応じられない」ということを理由にしていた。業界誌の取材には宣伝効果がある等の判断で応じたのに対して、対照的であった。なお、回答結果を一般的な傾向と見做すことには慎重でなければならないが、参考となるないし示唆的な内容も少なくなかったので、以下に回答内容を整理して提示する。調査（一）と（二）の調査票は本章末尾にそのまま掲載する。また、回答企業の大半が企業名を伏せる等を回答条件としていたので、回答内容はほぼそのまま載せるが、企業名を挙げるないし特定できる取り扱いはしないように配慮した。

① クラウド事業者へ「自社で制作したソフトウェアを販売しているか」という質問に対して、「している」という回答は３社であり、「自社で制作したソフトウェアをサービス提供しているか（SaaS、PaaS、ASP 等）」という質問に対して、「している」という回答は４社であった。それら以外は、クラウド事業専業あるいはクラウド事業でも IaaS 形態でソフトウェアは営業対象になっていないと解せられる。筆者が最も知りたかったソフトウェアの販売とサービス提供の両方を行なっている企業は２社であった。

② 更に、両方を行なっている場合に、より踏み込んだ内容として、「自社で制作した同一のソフトウェアを、販売並びにサービス提供の両方で取り扱っているか」に対して、その２社は「取り扱っている」と回答した。但し１社は当初の回答では「していない」とし、再度の追加的な問い合わせに対し「特定の顧客に対して」限定的にしており、自社の「標準的ビジネス」ではないと断っている。

③ その場合に「どのような会計処理をしているか」に対しては、１社は「市場販売目的の会計処理をしている」との回答であった（1 社は回答なし）。より立ち入った対面調査にも応じて頂いた回答を加えると、従来からパッケージ販売を行なっており、近年 SaaS 事業を行なうようになったが、特に開発時を含めて区別しておらず、合算で処理しており内訳はわからない、とのことであった。また、「SaaS もライセンス契約を締結して提供するので、サービス提供というより、従来のパッケージ販売と同様に考えている」とのことであった。なお、期せずして統一的な処理を行なっているが、会計基準に則って選択適用し、別々の処理をしなければならないとすれば、困惑するのではないかと思われる。

④ 会計処理に関して「どのような判断基準で処理しているか」に対しては、自社での判断という回答が１社、「税理士と相談して決めている」という回答が１社であった。

[414] 業種分類並びに売上規模分類は、ＪＵＡＳ(2012b)とＪＩＳＡ(2010)に準じた。

⑤　販売とサービス提供の両方を現在は行なっていないが、今後行なう場合、「どのような処理を想定しているか」に対しては、「結局は専門家と相談して決めるのが合理的だと思う」という回答が１社あった。また、「統一的な処理を望むか、あるいは具体的な指針を望むか」という意見的な質問に対しては、「具体的な指針を望む（実務指針の改訂、判断基準の具体化）」という回答が１社あった。

（4）調査結果を踏まえた考察

調査（一）に関しては、得られた回答並びそれを踏まえて筆者が進めた考察は次の通りである。

第１に、ソフトウェア会計基準の制作目的別分類が経済事象と適合的でないことである。自社利用のソフトウェアの資産性は「収益獲得又は費用削減」を根拠としているが、そのうちサービス提供は収益の直接的源泉であるのに対し、社内利用は間接的であり、測定も容易ではない。一方、市場販売目的のソフトウェアは収益の直接的源泉であり、サービス提供と類同的で、営業形態が異なるだけである。それ故、収益との対応の直間性による分類としなければ、分類が不適合であることは明らかではないだろうか。

第２に、これまでは分類が不適合であろうとも、対象が異なり、別々に処理していた限りでは、問題が表面化することはなかった。ところが、クラウド事業により同一のソフトウェアが販売もされ、サービス提供もされるようになったため、明確に問題が露呈してきたと言わなければならない。まだ、実際に問題視する意見等はＪＩＳＡ以外には公表されていないが、早晩クラウド事業者の会計実務では恣意的な処理が増大し、比較可能性が損なわれることが懸念される。

第３に、ＪＩＳＡの現行基準ベースの整理にしても、既述の通り、「一般的なＡＳＰサービスと通常のパッケージ販売の混合型」等に関しては、何ら指針を示していない。混乱を回避するためには、強いて解釈的だが、「売上規準」（資産計上時点で判断しなければならないから、何れの売上予測を「主」として見込むか）等により「主」「従」を識別して、「主」たる事業による会計処理を指針化するべきであろう。それでも、見込み違いが生じた場合の補正の手立てはない、という問題は残る。

第４に、結論的には、元々の「ボタンの掛け違い」を匡し、ソフトウェアという経済事象に適合的な分類に改め、且つ統一的な会計処理に変更するべきである。

４．会計的問題（二）：利用企業側におけるプライベート・クラウドの資産性

（1）現行の会計処理

クラウド・コンピューティングに係る会計処理で問題となり得ることのもう１つは、自社保有のソフトウェア等による運用に替えて、プライベート・クラウドを利用する場合の会計処理である。ユーザ企業はこれまで、購入ないし自社開発によって調達したソフトウェアを保有し、要件を充足すれば資産計上を行ない、システム運用を行なってきた。それに対し、その一部ないし全部をクラ

ウドの利用に切り替えたとする。クラウドの利用は、基本的には従量課金制または月額定額制である。パブリック・クラウドの場合は、不特定多数の利用者の一員であるから、サービスの利用に関する費用処理で特段の問題はない。しかし、プライベート・クラウドの場合、当該企業用に品揃えをし（任意ではなく指定範囲内の選択の場合もあるが）、運用をするのであるから、その利用の有り様並びに契約内容如何によっては、利用企業の保有資産として資産計上する必要があるのではないか。例え、環境設置場所は利用企業からは離れたとしても、実態としては限りなく利用企業の裁量によって専有（専用）の環境構築を行なっている事態は、これまでの利用企業保有の資産と実質的には変わりないのではないか。これが、もう１つの問題である。

この問題に関しては、ＪＩＳＡでは取り上げておらず、事業者側の業界団体の立場では致し方ないかもしれないが[415]、筆者の知る限り同様な問題意識での先行研究や提言等は見当たらないので、独自に考察を進めることにする。

なお、未だ流動的であるプライベート・クラウドを考察する際、留意すべきことがある。一方の極には企業が自社保有しているシステム環境を仮想化すること且つそれだけのことをプライベート・クラウドと称する態様があり、中間的には従来からあるアウトソーシング、ハウジング、ホスティング等の態様があり、もう一方の極には限りなくパブリック・クラウドに近い今日的な仕組みの中で、業務システムとしては「自社用」であるような態様がある。前者の極に近いものは恐らくこれまでと同様に会計処理としては資産計上を基本としているであろうから、そうであれば問題はないと言える。前者の極からは離れ、中間的な態様ないし後者の極に近づくに連れ、問題となり得るのである。このような整理をしておかなければ、同じプライベート・クラウドと称しながら、異なった態様を想定して議論が擦れ違う、あるいは喰い違うことになりかねないのである。

（2）会計実務の調査（二）

調査（二）は、事業者並びに利用企業の双方から回答が得られたので、相当程度実態把握に近づき得たように思われる。

① プライベート・クラウドに関して、「行っている」事業者は４社であった。形態として、利用企業にとって「オンプレミス」(on-premise：自社構内に環境設置）か「オフプレミス」(off-premise：クラウド事業者のデータセンター等に環境設置）かという質問に対して、「両方」が３社、「オフプレミス」が１社であった。オフプレミスの場合、物理リソース等は利用企業にとって「共有型（共用型)」か「専有型（専用型)」かに対して、「共有型（共用型)」だけが１社、「両方」が３社であった。但し、１社は「ほぼ専有、共有はごく一部」と補記している。サービス範囲として、システム環境の提供以外に、「システムの設計&構築有」と「監視&運用サービス有」を４社とも回答した。サービスの種類として「オーダーメイド可」が３社、「レディメイドのみ」が１社であった。

[415] しかし、利用企業側の団体であるＪＵＡＳの調査報告ＪＵＡＳ(2012b)でも取上げられていない。

② 他方、利用企業では、「クラウドを利用しているか」に対して、「利用している」企業が 15 社で回答企業全てであった。そのうち、パブリック・クラウドに関しては、「利用している」が 12 社、「利用していない」が３社であった。プライベート・クラウドの利用に関しては、「利用している」企業が８社、但し１社は対面調査で確認したところアマゾンのクラウド・サービスを利用しているとのこと、もう１社はセールスフォース・ドットコムのサービスを利用しているとのことなので、プライベート・クラウドの利用とは見做し難く、「利用していない」が７社であった。利用形態として、「オンプレミス」が４社、「オフプレミス」が５社であった。但し、「オンプレミス」のみが１社、「オフプレミス」のみが２社、両方の利用が３社であった（それ以外は回答なし）。更に、「オフプレミス」の場合、物理リソース等の利用は「共有型（共用型）」が２社、但し１社はネットワークが閉域網、「専有型（専用型）」が３社であった。そのうち２社は「これまでのアウトソーシング／ハウジング／ホスティングと類似のもの」との回答であった。

③ プライベート・クラウドの料金体系に関しては、事業者側では、「多様」が１社、「設計するインフラのボリュームにより月額制」が１社、「原価プラス」が１社、オンプレミスの場合は「一括売り」価格でオフプレミスの場合はボリュームによる「月額制」が１社であった。利用企業側では、「月額定額制又は年額定額制」が５社、そのうち４社は「定額で利用できる範囲」はストレージ容量や通信量等の制限があるとのことで、「制限なし」は１社であり、「従量課金制」が２社、但し１社は定額制との併用で、利用割合は定額制が９割、従量課金制が１割とのことであり、「その他」が１社、具体的には「ユーザーＩＤ数に応じた料金」であった。

④ プライベート・クラウドの場合、パブリック・クラウドと異なる「契約上の規定（例えば解約制限の制約等）はあるか」に対して、事業者側では、「ない」が１社、「ある」が３社であった。「ある」場合、２社は契約期間に関することで、１社は「パブリック・クラウドは一律１ヶ月、プライベート・クラウドは 12 ヶ月、24 ヶ月、36 ヶ月からお客様が選択」、１社は「パブリック・クラウドの最短契約は６ヶ月だが、プライベート・クラウドの場合は 12 ヶ月」とのことであった。もう１社はオンプレミスは「一括売り」なので「販売契約形態の根本がそもそも異なる」とし、オフプレミスは「大きな差なし」であった。利用企業側の回答では、「ない」が３社、「ある」が３社であった（2 社回答なし）。「ある」場合、１社は「１年毎に契約の見直しが可能」、１社は「オンプレミスの機器はリースの為途中解約すると違約金が発生する」、１社は「①基本的には解約不能（契約で定められていて、解約する場合には違約金がかかる）、②損害賠償の範囲が、パブリックに比べて狭い（専用であるぶん、事業者側が負担してくれる部分がパブリックより狭く、それ以外は自己負担となる）」との回答であった。

⑤ 続いて、ソフトウェアのライセンス契約に関する質問である。事業者側では、「利用企業の仮想環境（専用部分）で、他社製の商用ソフトウェアを搭載・利用可とする場合、他社とのライセンス契約はどのように締結するか」に関して、１社は「他社と自社が契約締結」、３社は「他社と利用企業が契約締結」であった。オープンソースに関しては、「オープンソース財団等と自社が契約締結」

が１社、「オープンソース財団等と利用企業が契約締結」が３社であった（商用ソフトウェアと同様）。専用の仮想化ソフトウェア（利用企業固有のシステム環境等）に関しては、「自社が契約締結」が３社、「利用企業が契約締結」が１社であった。共用の仮想化ソフトウェア（事業者のシステム環境内で個々の利用企業用の環境を構築・設定する等）に関しては、「自社が契約締結」が３社、「その他」が１社、但しまだ実際には「そのようなケースはない」とのことであった。

⑥　それに対して、利用企業側では、「クラウド事業者のデータセンターにおける利用企業の仮想環境（専用部分）で、クラウド事業者製の商用ソフトウェアを搭載・利用する場合、ライセンス契約はどのように締結しているか」に関して、「クラウド利用の包括的な契約に含まれている」が８社、「当該ソフトウェアに関するライセンス契約を締結」が２社、但し「包括契約内で特定条項」として、と回答している（5 社回答なし）。クラウド事業者製以外の他社製の商用ソフトウェアに関しては、「クラウド事業者が他社と契約締結」が５社、但し１社は「他社製ソフトウェアの調達を含めて頼んでいるが、ユーザの意向で利用するなら、ユーザが調達し他社と契約するということも一部ではある」と補記しており、「利用企業が他社と契約締結」が４社、但し１社は「その他」の回答項目をも重複回答し、具体的には「大手ＳＩｅｒがアプリベンダーを担いだ場合は大手ＳＩｅｒが契約窓口となる」と追記している。なお「契約窓口」が手続代行的なことに留まるのか、契約の当事者ともなることなのか（「その他」をも重複回答しているという意味ではそうとも解せられる）、定かでない（5 社回答なし）。オープンソースに関しては、「オープンソース財団等とクラウド事業者が契約締結」が６社、「オープンソース財団等と利用企業が契約締結」が１社、「その他」が１社で、「現時点では該当なし」と補記している（7 社回答なし）。専用の仮想化ソフトウェアに関しては、「クラウド事業者が契約締結」が５社、「利用企業が契約締結」が３社であった（7 社回答なし）。共用の仮想化ソフトウェアに関しては、「クラウド事業者が契約締結」が７社で、「利用企業が契約締結」という回答はなかった（8 社回答なし）。

⑦　プライベート・クラウドとパブリック・クラウドの複合的形態であるハイブリッド・クラウドに関しては、事業者側では、「サービスを提供している」が３社、「提供していない」が１社であった。料金体系に関しては、３社とも「各々の合算」であった。利用企業側では、「利用している」が３社、「利用していない」が 12 社であった。料金体系に関しては、「各々の合算」が２社であった（1 社回答なし）。

⑧　これ以降は会計処理に関することであるが、利用企業側がパブリック・クラウドを利用している場合、「どのような会計処理をしているか」に対して、「費用処理」が８社であった（4 社回答なし）。費用科目は、「ソフトウェア保守」、業務委託費の「その他諸費」（ネットワーク料は「通信費」）、「システム業務委託費」、「賃借料」、「諸雑費」、「システム外注費」と区々様々であった（2 社回答なし）。プライベート・クラウドを利用している場合、「費用処理」が８社であった。費用科目は、業務委託費の「その他諸費」（ネットワーク料は「通信費」、なお、パブリックとの「切り分けはしていない」とのこと）、「通信費」、「システム業務委託費」、「賃借料」、「諸雑費・通信費」と区々

様々であった（3 社回答なし）。「資産計上」が１社あったが、「費用処理」との両立てで、具体的には「ハードのリース案件」とのことなので、ソフトウェアに関しては該当しない。

⑨　プライベート・クラウドに関して、実際に行なっている会計処理ではなく、意見的なこととして、「どのような会計処理が適切と考えるか」に対しては、事業者側では、「事業者が資産計上（利用企業は費用処理）」が２社、「利用企業が資産計上（事業者は費用処理）」が１社、オンプレミスは「利用企業が資産計上（事業者は費用処理）」でオフプレミスは「事業者が資産計上（利用企業は費用処理）」が１社であった。利用企業側では、「事業者が資産計上（利用企業は費用処理）」が７社、「利用企業が資産計上（事業者は費用処理）」が２社、「考えていない」が１社、「その他」が１社で、具体的には「事業者が主体的に構築」する場合は利用企業では「費用処理」、「ユーザ企業が主体的に構築し、データセンタにホスティングする場合は利用企業が資産計上するのが適切」とのことであった（4 社回答なし）。

なお、事業者側と利用企業側で傾向値として相応に対応付いていないのは、標本数が限られているため、十分に趨勢を代表し得ていないからであると思われる。しかし、経済産業省の調査はもとより、ＪＩＳＡ並びにＪＵＡＳの調査でも標本数は遙かに大きいが、問題を剔抉できるような踏み込んだ調査が行なわれていない状況では、一学徒の調査として意義はあると考える。また、「門前払い」の企業が多かった中で、関心を示し、積極的なご協力を頂いた企業並びにご対応頂いた方々には厚く感謝したい。加うれば、上記等の有力な機関・団体が更に広範囲に詳細に踏み込んだ実態調査を行なうことを望みたい。

（3）調査結果に関する問題整理

前節では企業からの回答を項目毎に整理することに終始したので、次に調査の帰結へ向けて問題の整理を行なうが、筆者の意見的な考察となることを予め断わっておく。

①　まずパブリック・クラウドの利用に関しては、利用企業の独自性はほとんどなく、不特定多数の利用者の一員であるので、利用料が費用処理されている現状に特段の問題はない。唯一、勘定科目が区々様々で、クラウド利用を窺い知ることが容易ではないので、比較可能性という点からも、何らかの統一的な科目による処理が望まれるところである。

②　これ以降は専らプライベート・クラウドに関して、取り上げる。利用する際の料金体系に関しては、パブリック・クラウドと大きく異なることはないと言える。しかし、利用に関する契約に関しては、パブリック・クラウドと異なっている点がある。１つは期間的には比較的長いこと、２つには解約不能な契約があること、３つには事業者側に起因する障害・事故発生に対する損害賠償の範囲が狭いことなどである。特に２つ目は、今回の調査では限定的で全てに該当するわけではないが、当該企業専有（専用）であることとして示唆的である。また、オンプレミスに関しては「一括売り」としている場合があることは特筆に値する。

③　ソフトウェアのライセンス契約に関しては、クラウドに関する夥しい業界誌の記事や書籍等

では着目されていないが、筆者は極めて重要な事項であると考え、本調査の意義の１つと自負している。何故ならば、契約締結の相手方が利用の当事者であることを意味し、利用に起因する事故等の責任を負うことになるからである。他社製の商用ソフトウェアに関して、回答の限りでは、事業者側では大半が事業者は当事者にはならず、「他社と利用企業が締結している」のに対して、利用企業側では「事業者が他社と締結」と「利用企業が他社と締結」が半々ぐらいだったのは（事業者側と非対称なのは標本数の少なさによる偏りであると解せられる)、筆者にとって意外である。実態的にその通りなのか、もしくはクラウド事業者のサービスとして契約締結の手続きを代行しているだけなのかは、クラウド利用の契約内容の詳細を開示してもらっていないので、未確認であり、疑念として残る。オープンソースに関しては、商用ソフトウェアに比べ、事業者が締結する傾向が多少増しているが、利用が無償であることが影響し、賠償責任を問われることも少ないからではないかと推測される。仮想化ソフトウェアに関することを含めると、事業者が管理・監視する関与度合とおおよそ対応していると解することができる。プライベート・クラウド環境内のソフトウェアに関しては、事業者は形式的な運用を管理・監視はするが、例え技術的、手段的には介入は可能だとしても、セキュリティ上の制限を含め、内容的な関与は許されない立場にある。従って、仮に不正利用等があったとしても、事業者の責任は間接的であり、直接的には利用企業が責任を負うのが合理的である。そういう意味で、利用企業側の回答が実態を反映しているとするならば、事業者が過大な責任を負っていることになり、疑問が残る。敢えてリスク・テイクによる事業戦略であれば、１つの立場の選択ではあるが、果たしてどうであろうか。いずれにせよ、これだけで決定的な要件とまでは言い切れないが、重要な要件なので、より広範な実態把握が求められる。また、進化途上にあるクラウド事業にとっては、責任主体を意味するライセンス契約の締結の有り様を十分に踏まえることが望まれる。

④　プライベート・クラウドの利用企業における会計処理に関しては、現状ではほとんどが費用処理を行なっている。勘定科目に関しては、パブリック・クラウドと同様な問題を指摘することができる。会計処理に関する意見的なことでは、事業者側では２社が「利用企業が資産計上」することが適切と考えているのに対して（但し１社はオンプレミスに限定)、利用企業側では「利用企業は費用処理」が回答の限りでは多数意見であった。

（4）調査結果を踏まえた考察

以上の考察を踏まえて、プライベート・クラウドに係る会計処理を考える。まず、留意事項に触れておく。１つは、記述の煩雑さを避けるために、主として利用企業の会計処理として提示する。事業者側は概ね対称的な取扱いとなる。２つには、クラウドの構成要素は、大別すると、ハードウェアとソフトウェアと構築並びに運用のサービスがあるが､本章の主題はソフトウェアであるから、ソフトウェアの会計処理に限定することである。同じＩＴの構成要素でも、クラウドに限らず、ハードウェアはこれまでも有形固定資産等で独立的に別個の会計処理が行なわれてきたのであり、そ

れは変わらない。また、構築並びに運用のサービスは、取り敢えずソフトウェアの資産性とは関係のないことであるから、これまで通り費用処理することに特に問題はない[416]。従って、あくまで取り上げるのはソフトウェアに関することに限定する。

第１に、次の３つの要件の何れかに該当する場合には、その対象範囲は利用企業が資産計上することが妥当である、と考える。認識時点は、ソフトウェアの場合には検収時点とするのが適当である。理由は、ソフトウェアの取引の一般的な完了時点だからである。

① クラウド利用の契約が解約不能なこと[417]。

② ソフトウェアが利用企業用のオーダーメイドであること[418]。

③ クラウド事業者もしくは他社製等のソフトウェアに関して、相手方としては利用企業がライセンス契約を締結するものであること。

理由は、次の通りである。①の場合は、調査の限りでも、契約形態には幾通りかあったが、解約不能の契約とすることは､事業者にとっても当該利用のリソースが利用企業向けの特有性が濃厚で、他の利用企業への転用が困難なものであるからと推測される。これは、実質的には当該利用企業の保有資産であると見做すことが適合的である。②の場合は、①と同様ないしそれ以上に、オーダーメイドのソフトウェアを他企業用に転用できることはほぼないから（部分的な再利用はないとは言えないが)、当該企業で使い切りとなるものであり、例え期間契約であったとしても、利用企業の資産と見做すのが合理的である。③の場合は、契約の当事者が事業者ではなく利用企業であり、法的にまさに当該利用企業の所有資源であることを示すものであり、賠償責任等を負うことにもなるので、これまでのソフトウェアの外部購入の場合ないし自社保有の場合と全同であり、会計処理としても整合的である。

第２に、契約形態として包括契約となっている場合のソフトウェアの価格（料金）の取扱いに関することであるが、システム要件（リソースの種類と数量等）により価格（料金）が設定されるのであるから、内訳は明らかなはずである。従って、個々のソフトウェアのうち、上記第１で挙げた要件に該当するソフトウェアを抽出し、その価格（料金）を算定することは可能である。例え値引き等々を理由に内訳が提示されない場合でも、一般的な自社保有の場合の価格は比較的容易に知り得るから、利用企業側で金額を算定することが困難となることはまずないであろう。

第３に、毎月等の単純な費用処理に比べれば、上記のような取扱いは会計実務上多少の処理増大とはなろうが、元々購入ないし自社保有をしていた場合に行なっていたことと変わりないのであるから、それを理由に忌避されるほどのことではない。プライベート・クラウドの特性に適合的な会計処理を第一義的に考えるべきである。

---

416 但し、運用改善はその限りではないが、それに関しては前章で取り上げているので、参照。
417 「リース取引に関する会計基準の適用指針」5.(1)、6.の解約不能要件が参考になる。
418 「リース取引に関する会計基準の適用指針」10.(3)の特別仕様規準が参考になる。

クラウド・コンピューティングが惹起する会計的問題を２つ考察してきた。１つは、クラウド事業者が同一のソフトウェアをパッケージ販売とクラウド事業でのサービス提供の両方で取り扱う場合に、現行のソフトウェア会計基準では制作目的別分類に基因して、実態にそぐわない恣意的な会計処理をせざるを得ないという問題である。それに対しては、収益の源泉としての直間性による分類並びに統一的な会計処理への改訂を提案した。

もう１つは、プライベート・クラウドの利用に関して、それまでの自社保有の資産計上に替えて、費用処理とする傾向が多く見受けられるが、筆者はそれが実質にそぐわないと考える問題である。但し、注意を要するのは、プライベート・クラウドと総称される事象・事態には多様な形態があり、それを適切に識別しなければならないということである。また、限られた調査ではあったが（得られた回答数を含め）、調査の結果、利用契約も一律ではないので、その内容に分け入り精査しなければ、適合的な会計処理が如何なるものか、明確化できないということであった。それに対して、リース会計を筆者なりに参考にして、３つの要件を挙げ、それに該当する場合には利用企業側が資産計上すべきことを提案した。なお、今後の課題として、クラウドの契約内容に立ち入り（契約書の閲覧が必要となる）、あるいは原価の具体的な金額を把握する、更に詳細な調査を行ない、より実態に迫る研究を行なうことが挙げられる。「企業機密の壁」があって容易なことではないが、今回の調査を通じてご協力を頂いた企業に更なるご協力を仰ぐ等により、アプローチすることを考えたい。

未だ進化の途上にあるクラウドに関して、時期尚早と受け取られるきらいはあるであろうし、多少の調整は必要になるかもしれないが、考察の大筋は揺るがないものと考える。今後一層普及拡大していくと予測されるクラウドが健全に発展していくためにも、会計的にコミットしていくことを使命と考え、考察を行なった。賛否共々、議論を誘発できればと願っている。

補遺Ⅰ　調査票

クラウド・コンピューティング調査票（事業者向け）

2013. 2. 1

明治大学大学院経営学研究科

長田芙悠子

＊選択回答の場合、該当の○を●にして下さい。

＊記述式の回答の場合、量を気にせず、自由にご回答下さい。

（社名：　　）

1. 自社で制作したソフトウェアを販売していますか。

○している　○していない

1-1. 貴社は、会計処理に関してどの会計基準を適用していますか。

○日本基準　○アメリカ基準　○ＩＦＲＳ　○その他（　　）

（なお、以下の会計処理に係る質問並びに回答は日本基準をベースにしています。従って、もし日本基準以外を適用されている場合は、実態に即した補訂等をお願い致します。）

1-2. 販売している場合、そのソフトウェアに関して、どのような会計処理をしていますか。

○資産計上　○費用処理　○その他（ケースによる等）（具体的には：　　）

1-2-1. 資産計上している場合、資産計上額は開発費総額のうち、どの程度ですか。

（　　）

2. 自社で制作したソフトウェアをサービス提供していますか（ＳａａＳ、ＰａａＳ、ＡＳＰ等）。

○している　○していない

2-1. サービス提供している場合、そのソフトウェアに関して、どのような会計処理をしていますか。

○資産計上　○費用処理　○その他（ケースによる等）（具体的には：　　）

2-1-1. 資産計上している場合、資産計上額は開発費総額のうち、どの程度ですか

（　　）

3. 自社で制作した同一のソフトウェアを、販売並びにサービス提供の両方で取り扱っていることはありますか。

○ある　○ない

3-1. 両方で取り扱っている場合、そのソフトウェアに関して、どのような会計処理をしていますか。

○市場販売目的の会計処理　○自社利用目的（サービス提供）の会計処理

○その他（ケースによる等）（具体的には：　　）

3-1-1. 市場販売目的と自社利用目的（サービス提供）とでは会計処理が異なりますが、どのような判断基準で処理をしていますか（例えば、売上が多い方の目的別処理をしている、監査法人と相談して決めている等）。

（　　）

3-1-2. 同一のソフトウェアに関して、異なる会計処理となることに不都合あるいは困ることはありませんか。

○不都合あるいは困る　○不都合はないあるいは困らない　○その他（　　）

3-1-3. 統一的な処理を望みますか、あるいは具体的な指針を望みますか。

○統一的な処理を望む（会計基準の改訂）　○具体的な指針を望む（実務指針の改訂、判断基準の具体化）　○今のままでよい　○その他（　　）

3-2. これまでは取り扱っていないが、今後両方の取り扱いを予定している場合、どのような会計処理を想定していますか。

○市場販売目的の会計処理　○自社利用目的（サービス提供）の会計処理

○その他（ケースによる等）（具体的には：　　）

3-2-1. 市場販売目的と自社利用目的（サービス提供）とでは会計処理が異なりますが、どのような判断基準で処理をすることをお考えですか（例えば、売上が多い方の目的別処理をするつもりである、監査法人と相談して決める等）。

（　　）

3-2-2. 同一のソフトウェアに関して、異なる会計処理となることに不都合あるいは困ることに

はなりませんか。

〇不都合あるいは困る　〇不都合はないあるいは困らない　〇その他（　　）

3-2-3. 統一的な処理を望みますか、あるいは具体的な指針を望みますか。

〇統一的な処理を望む（会計基準の改訂）　〇具体的な指針を望む（実務指針の改訂、判断基準の具体化）　〇今のままでよい　〇その他（　　）

4. プライベート・クラウド事業は行なっていますか。

〇行なっている　〇行なっていない

4-1. プライベート・クラウド事業を行なっている場合、プライベート・クラウドは多様な形態がありますが、どのような形態ですか（複数あれば、各々お答え下さい）。

4-1-1. 利用企業にとっての形態　〇オンプレミス　〇オフプレミス　〇両方

4-1-2. オフプレミスの場合の物理リソース等　〇共有型（共用型）　〇専有型（専用型）　〇両方

4-1-3. サービス範囲　〇システムの設計&構築有　〇監視&運用サービス有　〇その他（具体的には：　　）

4-1-4. サービスの種類　〇レディメイドのみ　〇オーダーメイド可　〇その他（具体的には：　　）

（上記以外に：　　）

4-2. プライベート・クラウドの価格体系（料金体系）は、どのように設定していますか（差し支えない範囲で具体的にお答え下さい）。

（　　）

4-3. プライベート・クラウドの場合、パブリック・クラウドと異なる契約上の規定（例えば解約期限の制約等）はありますか（ある場合、差し支えない範囲で具体的にお答え下さい）。

〇ない　〇ある（具体的には：　　）

4-4. 利用企業用の仮想環境（専用部分）で、他社製の商用ソフトウェアを搭載・利用可とする場合、他社とのライセンス契約はどのように締結しますか（例えばこれまでのアウトソーシングであれば、利用企業が契約締結する場合が一般的だったはずです）。

〇他社と貴社が契約締結　〇他社と利用企業が契約締結　〇その他（ケースによる等）（具体的には：　　）

4-5. 利用企業用の仮想環境（専用部分）で、オープンソースを搭載・利用可とする場合、無償でもライセンス契約は締結しますが、それはどのように締結しますか。

〇オープンソース財団等と貴社が契約締結　〇オープンソース財団等と利用企業が契約締結　〇その他（ケースによる等）（具体的には：　　）

4-6. 専用の仮想化ソフトウェア（利用企業固有のシステム環境等）は、どうですか。

〇貴社が契約締結　〇利用企業が契約締結　〇その他（ケースによる等）（具体的には：　　）

4-7. 共用の仮想化ソフトウェア（貴社のシステム環境内で個々の利用企業用の環境を構築・設定する等）は、どうですか。

〇貴社が契約締結　〇利用企業が契約締結　〇その他（ケースによる等）（具体的には：　　）

4-8. 共用の仮想化ソフトウェア(自社製又は他社製の商用ソフトウェアで有償の場合)に関して、価格体系(料金体系)はどのように設定していますか(差し支えない範囲で具体的にお答え下さい)。

（　　）

4-9. 仮想化ソフトウェアで、ロードバランシング、ライブマイグレーション等を行なっている場合（サーバ、ストレージ等)、価格体系（料金体系）にはどのように反映させていますか（差し支えない範囲で具体的にお答え下さい)。

（　　）

4-10. プライベート・クラウドとパブリック・クラウドの複合的な形態であるハイブリッド・クラウドのサービスを提供していますか。

〇提供している　〇提供していない　〇その他（　　）

4-10-1. ハイブリッド・クラウドのサービスを提供している場合、価格体系（料金体系）はどのように設定していますか（差し支えない範囲で具体的にお答え下さい)。

〇各々の合算　〇それ以外（ケースによる等）（具体的には：　　）

4-11. プライベート・クラウドの形態にもよりますが、利用企業にとって、これまで自社保有していたＩＴ資産からサービス利用（従って費用の支払）へと変わりますが、それはリースに似てい

るように思われます。ところで、リース会計は2007年の改訂でファイナンス・リースに関しては利用企業が資産計上することになりましたが（オペレーティング・リースは従来通り費用処理）、プライベート・クラウドも形態によっては同様になることが考えられます。事業者側の立場で、どのような会計処理が適切とお考えですか。

〇事業者が資産計上（利用企業は費用処理）　〇利用企業が資産計上（事業者は費用処理）
〇考えていない　〇その他（ケースによる等）（具体的には：　　）

以上です。ご回答、ありがとうございました。

補遺Ⅱ　調査票

クラウド・コンピューティング調査票（利用企業向け）

2013. 2. 1

明治大学大学院経営学研究科

長田芙悠子

＊選択回答の場合、該当の○を●にして下さい。

＊記述式の回答の場合、量を気にせず、自由にご回答下さい。

（社名：　　）

1. クラウド・コンピューティングを利用していますか。

○している　○していない

2. パブリック・クラウドを利用していますか。

○している　○していない

2-1. パブリック・クラウドを利用している場合、料金体系はどのようなものですか。

○従量課金制（従量の具体的な取り扱いは：　　）

○月額定額制（定額で利用できる範囲は：　　）

○その他（具体的には：　　）

2-2. 貴社は、会計処理に関してどの会計基準を適用していますか。

○日本基準　○アメリカ基準　○ＩＦＲＳ　○その他（　　）

（なお、以下の会計処理に係る質問並びに回答は日本基準をベースにしています。従って、もし日本基準以外を適用されている場合は、実態に即した補訂等をお願い致します。）

2-3. パブリック・クラウドを利用している場合、どのような会計処理をしていますか。

○費用処理（科目は：　　）

○その他（具体的には：　　）

3. プライベート・クラウドを利用していますか。

○している　○していない

3-1. プライベート・クラウドを利用している場合、どのような形態のものですか（複数の形態を利用している場合には複数お答え下さい）。

○オンプレミス（具体的な内容は：　　）

○オフプレミス（具体的な内容は：　　）

オフプレミスの場合の物理リソース等　○共有型（共用型）　○専有型（専用型）

（具体的な内容は：　　）

○システムの設計&構築～監視&運用サービスまで享受

（具体的な内容は：　　）

○レディメイド　又は　○オーダーメイド（具体的な内容は：　　）

○これまでのアウトソーシング／ハウジング／ホスティングと類似のもの

（具体的な内容は：　　）

○その他（具体的には：　　）

3-2. プライベート・クラウドを利用している場合、料金体系はどのようなものですか。

○従量課金制（従量の具体的な取り扱いは：　　）

○月額定額制又は年額定額制（定額で利用できる範囲は：　　）

○その他（具体的には：　　）

3-3. プライベート・クラウドの場合、パブリック・クラウドと異なる契約上の規定（例えば解約期限の制約等）はありますか。

○ない　○ある（具体的には：　　）

3-4. クラウド事業者のデータセンターにおける貴社の仮想環境（貴社の専用部分）で、クラウド事業者製の商用ソフトウェアを搭載・利用する場合、ライセンス契約はどのように締結していますか。

○クラウド利用の包括的な契約に含まれている　○当該ソフトウェアに関するライセンス契約を締結（包括契約内で特定条項含む）　○その他（具体的には：　　）

3-5. クラウド事業者のデータセンターにおける貴社の仮想環境（貴社の専用部分）で、クラウド事業者以外の他社製の商用ソフトウェアを搭載・利用する場合、他社とのライセンス契約はどのように締結していますか。

○貴社が他社と契約締結　○クラウド事業者が他社と契約締結　○その他（具体的には：　　）

3-6. クラウド事業者のデータセンターにおける貴社の仮想環境（貴社の専用部分）で、オープンソースを搭載・利用する場合、無償でもライセンス契約は締結しますが、それはどのように締結していますか。

〇オープンソース財団等と貴社が契約締結　〇オープンソース財団等とクラウド事業者が契約締結　〇その他（具体的には：　　）

3-7. 専用の仮想化ソフトウェア（貴社固有のシステム環境等）は、どうですか。

〇貴社が契約締結　〇クラウド事業者が契約締結　〇その他（具体的には：　　）

3-8. 共用の仮想化ソフトウェア（クラウド事業者のシステム環境内で個々の利用企業用の環境を構築・設定する等）は、どうですか。

〇クラウド事業者が契約締結　〇利用企業が契約締結　〇その他(具体的には:　　)

3-9. 共用の仮想化ソフトウェア(クラウド事業者製又は他社製の商用ソフトウェアで有償の場合)に関して、料金体系ではどのような取り扱いになっていますか。

（　　）

3-10. 仮想化ソフトウェアで、ロードバランシング、ライブマイグレーション等を行なっている場合（サーバ、ストレージ等)、料金体系にはどのように反映されていますか。

（　　）

4. プライベート・クラウドとパブリック・クラウドの複合的な形態であるハイブリッド・クラウドを利用していますか。

〇利用している　〇利用していない　〇その他（　　）

4-1. ハイブリッド・クラウドを利用している場合、料金体系はどのように設定されていますか。

〇各々の合算　〇それ以外（具体的には：　　）

5. パブリック・クラウドを利用している場合、どのような会計処理をしていますか。

〇費用処理（科目は：　　）

〇その他（具体的には：　　）

6. プライベート・クラウドを利用している場合、どのような会計処理をしていますか（複数の形態

を利用している場合は各々お答え下さい)。

〇費用処理（科目は：　　）

〇資産計上（具体的には、どういう場合ですか：　　）

〇その他（具体的には：　　）

6-1. プライベート・クラウドの形態にもよりますが、これまで自社保有していたＩＴ資産からサービス利用（従って費用の支払）へと変わることになりますが、それはリースに似ているように思われます。ところで、リース会計は2007年の改訂でファイナンス・リースに関しては利用企業が資産計上することになりましたが（オペレーティング・リースは従来通り費用処理)、プライベート・クラウドも形態によっては同様になることが考えられます。利用者側の立場で、どのような会計処理が適切とお考えですか。

〇事業者が資産計上（利用企業は費用処理）　〇利用企業が資産計上（事業者は費用処理）　〇考えていない　〇その他（ケースによる等）（具体的には：　　）

以上です。ご回答、ありがとうございました。

本論

第９章　ソフトウェア保守

１．ソフトウェア保守の概観

（1）統計的に見たソフトウェア保守

（2）ソフトウェア保守の様々な分類

（2-1）　国際標準ＪＩＳ規格におけるソフトウェア保守

（2-2）ＦＰ法におけるソフトウェア保守

２．会計におけるソフトウェア保守の取り扱いと問題点

（1）会計におけるソフトウェア保守の取り扱い

（1-1）現行の日本のソフトウェアに係る会計基準における取り扱い

（1-2）アメリカのソフトウェア会計基準等における取り扱い

（2）ソフトウェア保守に適合的な分類

（3）会計におけるソフトウェア保守の取り扱いの問題点

（3-1）消費税率変更対応

（3-2）消費税創設対応

（3-3）国際会計基準対応

３．ソフトウェア保守に適合的な会計処理

図表１-９-1　ソフトウェア保守分類一覧

図表１-９-2　ソフトウェア保守分類と具体例（消費税率変更対応）

図表１-９-3　ソフトウェア保守分類と具体例（消費税創設対応）

図表１-９-4　ソフトウェア保守分類と具体例（国際会計基準対応）

図表１-９-5　ソフトウェア保守分類と会計処理一覧

第９章　ソフトウェア保守

本論の第２章～第６章のソフトウェア開発に関する考察に引き続き、ソフトウェア・ライフサイクルに沿って、本章はソフトウェアの保守（maintenance）[419]を主題的に取り上げる。背景的要因は２つある。１つは、ソフトウェア保守は「地味」な分野ではあるが[420]、重要である。ソフトウェア保守の重要性について、ケーパーズ・ジョーンズ（Capers Jones）は近著で次のように述べている。「ソフトウェアアプリケーションの平均寿命は 10 年から 30 年以上に及ぶために、真の工学分野としては開発プロセスのみに注目するだけでは不十分であり、保守（欠陥修復）と機能拡張（新機能の追加）も含める必要がある」[421]。また、「ソフトウェアアプリケーションの運用開始後の規模拡大率は約８％／年であるから、20～30 年後には当初規模の２倍以上に膨らんでいると思われる」[422]。つまり、保守の期間は相当長期に亘り、且つ累積保守コストは初期開発コストを上回る場合も少なくないのであり、特有の工学分野として注目がなされてしかるべきなのである。

もう１つは、ソフトウェア保守は、（新規）開発と共通する点も多いが、相違する点があることである。ソフトウェア開発と保守との違いは、アクティビティ（作業）の観点から具体的に捉えることができる。ＩＰＡ(2007)は、保守を大きく三つのフェーズに分けており、①「調査・分析」と②「機能実現」と③「テスト」としている[423]。①「調査・分析」と③「テスト」において、母体となる既存システムを考慮したアクティビティが加わり、例え同規模でも開発に比べ工数・コストが増加する[424]、としている。このような保守の重要性並びに開発との相違から、保守を会計上独立的に考察する意義があると考えている。

そして、保守を会計的に捉える場合、何よりも問題になるのは、ソフトウェア会計基準における保守の取り扱いであり、特に分類の仕方である。１つは、ソフトウェア会計基準にはネガティブな不具合の対応とは異なるポジティブな維持対応の規定が欠落しているので、不具合（バグ、欠陥）対応と改良との間に隙間が生じてしまっている。もう１つは、業務上不要になった機能等を部分的に削除する規定が欠落している。筆者の知る限り、これらは問題視されず、放置されている。それ故、本章はこれらの欠落・遺漏（隙間）を匡正するために、ソフトウェア保守の実態に適合的な分類を考案し、それに基づく適合的な会計処理を導き出すことを目的とする。但し、測定に関しては

---

419 ソフトウェア実務では「保守」という用語が広く定着しているにも関わらず、それとは異なる用語を使う動向が見られる。「保守開発」「改良開発」といった開発との合成語としたり、「改良」という価値付与的な用語への選好が見られたりするが、筆者は実務に即した「保守」を用いる。

420 地味というのは、①開発に関する書籍が膨大にあるのに対し、保守に関する書籍は(翻訳書、絶版書を含めても)10 指に満たないこと、②保守固有の開発技法はリファクタリング(refactoring)程度しかないこと、③技術者が保守業務に従事したがらないこと、などを含意させている。

421 Jones(2010)邦訳 p. 202

422 Jones(2010)邦訳 p. 202

423 ＩＰＡ(2007)p. 12

424 同書 p. 13

示唆的な言及に留める。

本章の進め方は、１．でソフトウェア保守の特性を明確にするために、ソフトウェア・ライフサイクルに占める期間とコストの面から保守を概観する。また、比較的よく知られている保守の分類を取り上げ、問題点を浮彫りにする。２．で会計の現状における保守の取り扱いに関して、日本のソフトウェア会計基準に留まらず、アメリカ基準等をも取り上げ、特に分類とそれに基づく会計処理の仕方を確認する。次に、核心部分であるが、各種の保守分類を比較評価し、問題点を一層明確化した上で、ソフトウェア保守に最も適合的な筆者の分類案を提示する。更に、各種分類を具体的な保守事例に適用すると、途端に曖昧さにより非実効的となり、それと対比的に、筆者案が実効的であることを検証する。３．で、その分類に適合的な会計処理案を提示する。

１．ソフトウェア保守の概観

（1）統計的に見たソフトウェア保守

保守を的確に捉えるために、統計的・数値的な様相を一瞥したい。まず、ソフトウェア・ライフサイクルの一般的な期間と、その中に占める保守の期間を捉える。ソフトウェア・メインテナンス研究会(2007)は、「開発費の投資回収を考えて開発期間の何倍もの期間使い続けるなら、計画的に開発されたソフトウェアの寿命が10年以上に達することは十分考えられます」[425]と説明している。また、ケーパーズ・ジョーンズに依れば、平均寿命は、ＭＩＳ（Management Information System：経営情報システム、自社利用ソフトウェアの相当数を占める）が10.13年、市販（市場販売目的ソフトウェア）が6.50年、システム（基本ソフト等、市販と併せ市場販売目的ソフトウェアに相当する）が12.00年[426]、である。更に、ＪＵＡＳに依れば、ジョーンズのＭＩＳに相当する「主な基幹業務システムのライフサイクル」の単純平均は、14.6年（開発後8.5年と今後の利用予定6.1年の期間合計）であり、2007年度調査と比較すると「ライフサイクル全体では0.8年ほど延びている」のである[427]。

次に、ソフトウェア・ライフサイクルに占める保守のコスト比率を捉える。ロバート・Ｌ・グラス（Robert L. Glass）は、「プログラム開発に要するコストと時間は、平均で全体の20～60％である。残りの40～80％は、保守にかかる（本書で後ほど展開する議論上、保守は、ソフトウェア・ライフサイクル全体の約60％を占めるとここでは述べておく）。」とし、コストと時間の点で、保守が中心的なフェーズとして重要である、と捉えている[428]。そして「後ほど展開する議論」で、その「全体の約60％」の「保守費用の60％は、機能変更用、すなわち、ソフトウェアの効率を上げるための

---

425 ソフトウェア・メインテナンス研究会(2007)p. 4

426 Jones(2008)邦訳 pp. 244-246

427 ＪＵＡＳ(2012)p. 232

428 Glass(2003)邦訳 pp. 184-185。なお、文中の「プログラム」はソフトウェアと解すればよい。また、数値には根拠が示されていないので大まかな傾向性と受止めておくのが無難であろう。

費用」[429]であり、「保守フェーズで不良修正にかけるコストは、保守費用全体の 17%と、非常に低く」[430]、残りの「18%は環境適合のために使う。すなわち、異なるシステム環境下で、プログラムを稼働させるための保守作業である。ハードウェアが新しくなったり、ＯＳが変わったり、新パッケージ製品の新インターフェースに合わせたり、新しいデバイスを支援するなどの環境変化への対応である」。「保守の残り 5%」は「いわゆるその他」である。概算値として参考になる。

具体的な調査結果として、ＪＵＡＳ(2010)は「自社開発の稼働後の保守費用」を掲示している。「自社開発の初期開発投資」に対する保守費用の割合(%)の平均値として、初年度 7.7%、２年目 7.7%、３年目 9.1%、４年目 8.9%、５年目 10.9%、６年目以降 10.7%が挙げられている[431]。それを集計すると、開発費用に対して、初年度から６年目以降（が何年目までかは不明だが）までの保守費用は 55.0%ということになり、大凡半分強である。これから言えることは、開発費用が一時費用としては多額であるのに対して、保守費用は個々の案件毎には些少であるが、ソフトウェアのライフサイクルが長期間に及ぶ場合には、総額としては開発費用を上回ることもあり得るということである。

（2）ソフトウェア保守の様々な分類

比較的よく知られているソフトウェア保守に関する国際標準並びにＦＰ法の分類により、ソフトウェア保守の特性を更に明確化する。

（2-1） 国際標準ＪＩＳ規格におけるソフトウェア保守

まずは、ソフトウェア・メインテナンス研究会の一書がサブタイトルにも冠して全般的に依拠し、ＩＰＡ(2007)でも定義で依拠している[432]、国際標準且つＪＩＳ規格である「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」（Software Engineering－software Life Cycle Processes－Maintenance、ISO/IEC 14764:2006、JIS X 0161:2008）（以降では「国際標準 JIS 規格」と略記）[433]における、保守の分類を取り上げる。保守の定義は、「ソフトウェアシステムへの費用対効果の高い効率的な支援の提供を要求するあらゆる活動。活動は、引渡し後と同様に引渡し前にも実施される」とし、注記で「引渡し前の活動としては、引渡し後の運用、支援性及び配送決定についての計画策定が挙げられる。引渡し後の活動には、ソフトウェア修正、訓練及びヘルプデスクの

[429] 同書 p. 187。なお、「機能変更」は「機能拡張」とも言い換えており、そのほうが趣意には相応しい。
[430] 同書 p. 188。なお、続く引用も同ページからである。また、同書の「環境適合」の例示の中には保守の範疇を超えていることがあるので、要注意である。
[431] ＪＵＡＳ(2010) p. 145
[432] ＩＰＡ(2007) pp. 9-10
[433] ＪＳＡ(2011) pp. 259-292

運営が挙げられる」としている[434]。

保守を「訂正」と「改良」に大別し、前者を①「是正保守」と②「予防保守」、後者を③「適応保守」と④「完全化保守」に分類している[435]。

①「是正保守」(corrective maintenance) は、「ソフトウェア製品の引渡し後に発見された問題を訂正するために行う受身の修正 (reactive modification)」で、注記に「この修正によって、要求事項を満たすようにソフトウェア製品を修復する」とある[436]。なお、その系として「緊急保守」(emergency maintenance) があり、これは「是正保守実施までシステム運用を確保するための、計画外で一時的な修正」であり、注記で「緊急保守は、是正保守の一部である」としている[437]。

②「予防保守」(preventive maintenance) は、「引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」[438]である。

大別分類の「改良」は、「改良保守」(maintenance enhancement) と称呼し、「新しい要求を満たすための既存のソフトウェア製品への修正」で、注記で「改良保守には、適応保守及び完全化保守の二つのタイプがある。改良保守は、ソフトウェアの訂正ではない」としている[439]。

③「適応保守」(adaptive maintenance) は、「引渡し後、変化した又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」で、注記でやや長く「適応保守は、必す(須)運用ソフトウェア製品の運用環境変化に順応するために必要な改良を提供する。これらの変更は、環境の変化に歩調を合わせて実施する必要がある。例えば、オペレーティングシステムの更新が必要になったとき、新オペレーティングシステムに適応するためには、幾つかの変更が必要かもしれない」としている[440]。

④「完全化保守」(perfective maintenance) は、「引渡し後のソフトウェア製品の潜在的障害が、故障として現れる前に、検出し訂正するための修正」で、注記で「完全化保守は、利用者のための改良、プログラム文書の改善を提供し、ソフトウェアの性能強化、保守性などのソフトウェア属性の改善に向けて記録を提供する」としている[441]。

これらには、大きく３つの不備がある。１つは、保守をバグ対応に限定される「訂正」と「改良」に２大分類しており、維持的な性格の保守を逸している。２つには、大分類の「改良」の③が不明確なことである。「環境変化」が、専ら技術的なものか、業務的な環境変化への適応を含めたものか、両様の解釈があり得る。定義の文言と例示の限りでは、技術的な変化に限定しているようだが、規

---

434 同書 p. 262。注記の「運用」や「ヘルプデスクの運営」は、ソフトウェアの運用に関することなので、保守の定義に含めることは誤りである。
435 同書 p. 262。なお、①等は筆者が付番。
436 同書 p. 261
437 同書 p. 261
438 同書 p. 262
439 同書 p. 261
440 同書 p. 261。なお、オペレーティングシステムは引用以外ではＯＳと略称する。
441 同書 p. 262

程の全篇を通して明確化していない。業務的な環境変化への適応をも明示的に含めるべきである。３つには、仕様変更における部分的な削除が、いずれの分類にも該当しないのである。機能（仕様）が不要となり削除するので、バグの除去とは異なり、独立的に分類に加えなければならない。

（2-2）ＦＰ法におけるソフトウェア保守

ＩＦＰＵＧのＦＰ法は、ソフトウェアの機能規模を計測する手法である[442]。開発費用の見積りあるいは実績の測定において広く利用されている。

「計測種別」には、「新規開発プロジェクト」、「機能改良プロジェクト」、「アプリケーション」の３つがある。「アプリケーション」という種別は、最新の機能規模全体の計測であり、「機能改良プロジェクト」が遂行されると、「新規開発プロジェクト」とその「機能改良プロジェクト」を総合して最新の計測を行なうのである。従って、ここで取り上げるのは「機能改良プロジェクト」と「アプリケーション」である。「機能改良プロジェクト」が保守に相当する[443]。

「機能改良プロジェクト」[444]は、規模の計測として、性格の異なる機能の追加・変更・削除を的確に識別した計測となっており、機能追加と機能変更と移行機能の規模の合計に、機能削除の規模を加算する計算である。なお、機能削除を別の括りにしているのは、機能追加や機能変更は新たな作業として今回の計測を反映させるが、機能削除にはそれは不要で旧の計測のままとすればよい、という意味である。また、削除する機能規模を加算するのは、削除することも作業を行なうという意味で作業量の計測に繋げていくためであり、保守作業に対応する機能規模を計測する趣旨である。

そして、最新の機能規模全体の計測である「アプリケーション」[445]では、機能変更に関して「機能改良プロジェクト」の前後の差分を計測する「補正」をし、機能削除は減算する補正をするのである。この「補正」によって、「機能改良プロジェクト」後の全体規模が正しく計測される。

幾つかの問題点を指摘する。第１に、機能改良プロジェクトの計測に「移行機能」を加算することは不適切である。「移行機能」の主要作業はデータ移行であり、データはソフトウェアではないこと、また移行用のプログラムはデータ移行にのみ使用する使い捨てプログラムで、これ以降の運用には使用しないので、本番運用対象のソフトウェアの規模に加算することは不適切である。つまり、同じプロジェクトで作業を行なった場合でも、別の取り扱いとするのが適切である。会計上は、費用処理と資産計上の違いとなる。第２に、機能変更に関して、「アプリケーション」では正しく前後の差分を計測するが、機能改良プロジェクトの計測では変更後だけを計測していて、整合的ではないことである。機能改良プロジェクトの計測でも、差分を計測するのが整合的である。第３に、機能改良プロジェクトでの計測を「アプリケーション」でそのまま全体的に使用して再調整をするこ

442 ＦＰに関しては、ＩＦＰＵＧ(2005)を参照した。
443 「機能改良」には価値的な含意があるとは言えるが、バグ対応等も対象範囲としている。
444 本章では計測数値には立ち入らないので、計算式は省略する。ＩＦＰＵＧ(2005) pp. 9-12 参照。
445 計算式は省略する。同書 pp. 9-18 参照。

とは不適切である。部分的な「機能改良」の影響を無造作に全体に波及させる再調整だからである。これは会計的には、資産額を機能改良の都度再調達価額に置き換えることを意味する。もちろんFPが即座に資産額を示すものではないが、FPがベースとなり、コストの計算、ひいては計上される資産額へと繋がっていくからである。この再調整を取り止めて、増分を追加計上するだけとすべきである。第４に、FPを用いた会計上の取り扱いとしては、機能削除は資産対象から除外され「除却処理」対象となるので、１つの計算式に含めず、機能追加や変更とは区別して取り扱う必要がある。

以上のような問題点はあるが、維持か改良かといった価値的な判断を伴うことによる範囲・境界の問題は生じず、保守を既存のソフトウェア・プロダクトに対する追加・変更・削除という明解な違いによって分類し、それに即応した規模計測を行なうことはFP法の優れた特長であると言える。

２．会計におけるソフトウェア保守の取り扱いと問題点

（１）会計におけるソフトウェア保守の取り扱い

（1-1）現行の日本のソフトウェアに係る会計基準における取り扱い

ソフトウェア会計基準における保守の取り扱いには大きな問題がある。まずは会計基準での保守の取り扱いを整理する。実務指針Ⅱの 33.と 34.では市場販売目的ソフトウェアに関して、保守を「機能維持」と「改良（著しいものを除く。）及び強化」（以降「著しくない改良」と略記）と「著しい改良」に区分している。これに対して、自社利用ソフトウェアに関しては保守に関する規定が全く欠落している。実務指針Ⅰの 15.の規定は、再構築[446]又はパッケージソフトウェアの初期的なカスタマイズであり、保守ではない。「機能維持」等の規定は、厳密には市場販売目的ソフトウェアに関する、且つそれに限定した規定である。従って、それだけでも重大な欠落・遺漏だが、市場販売目的の規定を自社利用にも拡大適用・準用する流儀で、以降の議論を進めることにする。

「機能維持」は実務指針Ⅱの 34.で「バグ取り、ウィルス防止等の修繕・維持・保全」と規定している。「著しくない改良」は同Ⅱの34.でごく簡略に「ソフトウェアの操作性の向上等」としか規定していない。これに対して、「著しい改良」は同Ⅱの33.で少々詳しく、「研究及び開発の要素を含む大幅な改良を指しており、完成に向けて相当程度以上の技術的な困難が伴うものである」とし、「具体的な例として、機能の改良・強化を行うために主要なプログラムの過半部分を再制作する場合、ソフトウェアが動作する環境（オペレーションシステム、言語、プラットフォームなど）を変更・追加するために大幅な修正が必要になる場合などが挙げられる」と規定している。なお、「オペレーションシステム」は全く別の普通名詞であり、この文脈では正しくは「オペレーティングシステム（OS：Operating System）」でなければならない。以降ではOSと読み替える[447]。

---

[446] ソフトウェアの再利用(再構築を含む)に関しては、本論第４章で詳細に取り上げている。

[447] ＩＴの分野では同義語が多いので、改めて丹念に調べたが、大部の専門用語辞典である情報処理

これらの取り扱いには、問題が大きく２つある。１つは、「機能維持」と「改良」の区分の問題である。もう１つは、「改良」を「著しくない改良」と「著しい改良」に区分している問題である。まず、順序は逆だが、後者を取り上げる。「著しくない改良」を資産性の増加と認めて資産計上としながら、「著しい改良」は通常ならば更に資産性が顕著に増加するのであるから、同様に資産計上とするのが一貫性のある合理的な判断であるにも関わらず、「研究開発」と見做し、大半を費用処理とする不合理な取り扱いをしている[448]。

更に、「ソフトウェアが動作する環境」の例を３つ挙げているが、ＯＳとプラットフォームを除き、言語がなぜ、「ソフトウェアが動作する環境」なのか。ソフトウェアが具体的なプログラムとして体現される技術そのものであり、「環境」ではない。また、それらを「変更・追加するため」とあるが、変更と追加では大きく異なることが考慮されていない。追加であれば、部分的な影響に留まる可能性があり「改良」と見做しても差し支えないが、全面的な変更であれば、全面的な再構築つまり新規開発となる可能性が大きいのである。

次に、「機能維持」と「改良」の区分の問題であるが、「機能維持」とは、規定にある「修繕・維持・保全」を一般的に捉えると、より広い事象への対応が対象となり得るが、例示の「バグ取り、ウィルス防止等」と連接されると、範囲は極めて限られてくる。例え「等」を付して２例に限るものではないとしても、例示による限定は避けられず、素直に解釈すれば、ネガティブな事象への対応しか含意していないと見做さざるを得ない。バグ取りは不具合の除去、ウィルス防止はセキュリティ上の脆弱性の解消だからである。

用語大事典編集委員会編(1992)『情報処理用語大事典』オーム社、相磯秀夫監修、牧野武則・斉藤剛・中村克彦・オーム社編(2001)『情報技術用語大事典』オーム社、更にはＯＳに関する最も浩瀚なテキストである Silberschatz, Abraham, Galvin ,Peter Baer, Gagne, Greg, *Operating System Concepts 7th Edition*, Wiley.(2004)(アブラハム・シルバーシャッツ、ピーター・ベア・ギャルビン、グレッグ・ガニュ(土居範久・大谷真・加藤和彦・光来健一・清水謙多郎・高田眞吾・高田広章・千葉滋・野口健一郎訳)(2010)『オペレーティングシステムの概念』共立出版)では、「オペレーションシステム」は全く記載がなく、本文・索引ともオペレーティングシステムの「同義語」参照といった扱いすらされていない。JIS 情報処理用語解説編集委員会編(1985)『JIS 準拠コンピューティングデータ通信用語解説集』日本理工出版会でも、情報処理技術者試験受験者用の日本情報処理開発協会監修、日本システムアナリスト協会・上級システムアドミニストレータ連絡会編(2003)『情報処理技術者用語辞典』日経 BP でも全く同様である。そもそも世界初の商業化されたＯＳは 1964 年に IBM の System/360 に搭載された OS/360 であり、それは紛れもなく「オペレーティングシステム」と呼称され、それが用語としても一般化されたのである。これらからすると、年代や専門性のレベルによるブレは全くない。ただ１つ、筆者の検索し得た限りでは、ウィキペディアで、「オペレーションシステム」(URL:http://ja.wikipedia.org/wiki/ (アクセス 2013/08/21,17:59))とは、まず「様々な情報を一元化して管理するための仕組み。航空、物流などで使われる」という説明が挙げられており(普通名詞)、次いで「オペレーティングシステムの誤用」とある。確かに、インターネット上で検索すると、「オペレーションシステム」の名で UNIX や Windows に言及するサイトが幾つもあり、これで得心がいったと言える。学術的には認知されていないウィキペディアですら「誤用」と明記される程、初歩的なミスであり、会計における指針のような権威あるものがこのようなミスをすることは、お粗末である。こういった誤りを、ソフトウェアではバグという。

[448] この問題に関しては、本論第２章で主題的に取り上げている。

だが、一般的に「修繕・維持・保全」を捉えるならば、受動的ではあるがポジティブと言える事象への対応を含めて然るべきである。これが有形物との違いと言える。有形物であれば、それ自体の経年劣化や使用に伴う磨耗や汚染等が生じ、「修繕・維持・保全」で部品等の交換や研磨や洗浄等を行なうことで、近似的に「原状回復」するが、新品への完全な回復にはなり得ない。それに対して、無形のソフトウェアにはそれ自体の経年劣化や使用による磨耗等は一切起こらない。「修繕・維持・保全」であり得るのは、顕在的であれ潜在的であれ、埋め込まれていた不具合（バグ、欠陥）を除去するか、又は何らかの変化に対する受動的ではあるがポジティブな対応である。従って、「不具合（バグ、欠陥）対応」と、改良ではない「維持的な対応」と、「改良」という３区分が必要なのであるが、会計基準では、不具合（バグ、欠陥）対応である「機能維持」と「改良」（「著しくない」又は「著しい」改良）しか規定しておらず、不具合対応ではなく且つ改良でもない維持的な対応を欠落させ、隙間を作ってしまっている。これに関しては、後ほど例を挙げて更に明確にしよう。

（1-2）アメリカのソフトウェア会計基準等における取り扱い

ソフトウェア会計に関する先行研究には、ソフトウェア保守を主題的に取り上げているものは筆者の知る限りごく僅かしかない。そこで、ソフトウェアに係る国際的な会計基準等における取り扱いも含めて取り上げる。それらは①櫻井通晴（1993）、②ＦＡＳＢ（1985）、③ＡＩＣＰＡ（1998）である。

①櫻井通晴（1993）は、1991 年に当時の通産省所管で櫻井通晴教授が委員長として立ち上げられた「ソフトウェア資産性委員会」から公表された「ソフトウェア会計実務指針[案]」で、拘束力・強制力を持つ会計基準ではないが、ソフトウェア会計基準が制定される以前から、明確にソフトウェアの会計上の取り扱いを包括的に規定した指針案なので取り上げる。

同指針[案]の「７　機能の改良、強化と保守費用」では、まず「１．機能の改良、強化コストの会計処理」[449]は、「客先がついていない既存ソフトウェアの機能を改良、強化するための定期的・継続的作業のためにかかったコストは、研究開発目的を除き、製造原価とする」としている。注２で「既存製品に新たに機能を付加する際、付加する機能の新奇性ならびに開発リスクが高いものもある。」そのように、「新規開発と考えられるべき既存製品への改良、強化のためにかかった費用は、研究開発費として処理するのが妥当である」としている。続いて注３で、「ソフトウェアの価値は、時とともに急速に陳腐化する。そこで、当該ソフトウェアの現在の価値を維持するためには、どうしてもバージョン・アップが必要になる。かかる必要から行われるバージョン・アップのための費用を資本的支出と解するのは妥当ではない。」従って、既存製品の機能の改良、強化に限定して製造原価とすべきであり、既存製品の機能維持の費用は期間費用として処理すべきであるとしている。

次に「２．保守費用の会計処理」[450]では、「ソフトウェアが顧客に引き渡されてから保守と顧客支

[449] 櫻井(1993)pp. 59-60
[450] 櫻井(1993)pp. 60-61

援のためにかかった費用は、すべて発生時の費用である」としている。注１で「保守には、仕様、設計、コーディングなどに現れたソフトウェアの誤りを修正したり、簡単なユーザーからの要求、法律の小改正への対応、運用状況の変化などに対応するため、ソフトウェアを定型的に改訂することである。保守と機能改良、強化との区別は、現実には理論でいうほど明確ではない」としている。注２で「顧客に引き渡されてからの保守と顧客支援のための費用は収益的支出であり、資本的支出ではない。したがって、これらのコストは、発生時の費用で処理するのが妥当である」としている。更に、「保守とは、ソフトウェアが顧客へ引き渡された後、誤りを訂正したり、現在の情報にもとづいて当該製品を更新することをいいます。一方、バージョンアップとは、製品マスターの耐用年数を延長させ、販売可能性を著しく改善することを意図した既存製品に対する改良・強化のことをいいます」[451]とし、この両者の区別には「客先がついている」か否かが重要であり、ついていなければ、あくまで製造原価とすべきだとしている。

この取り扱いには、大きく２つの問題がある。１つは、「保守」は「顧客へ引き渡された後」の作業、「機能の改良、強化」は「客先がついていない」作業であるとして、両者を区別する規準を、客先の有無としていることである。同実務指針[案]は、販売用だけでなく、受注制作や自社利用のソフトウェアをも包括的に規定しているにも関わらず、販売用と受注制作のソフトウェアに囚われていると言わざるを得ない。既に運用している自社利用ソフトウェアに機能の改良、強化を施すのは、ユーザ企業主導の自社体制で外部者を参加させる場合と、外部委託する場合とがある。前者の場合には、客先などは存在しないので、客先という概念の適用自体が不適切である。後者の場合には、当該ソフトウェアを所有しているユーザ企業が客先であることは所与なので、「保守」に該当する。しかし、前者を「機能の改良、強化」と判断するならば、委託と自社で行なう場合とで、同様の事象に対する分類、ひいては会計処理が異なるのは不合理である。いずれにしても、曖昧で解釈の余地があり、不適切な判断規準である。整合性のある包括的に適用可能な規準としなければならない。２つに、「保守」あるいは「機能の改良、強化」ないしは「バージョン・アップ」の定義が曖昧なことである。保守の定義に挙げられている事象がどの範囲までを指すのか、機能の改良、強化の範囲とも密接に関わり、その分類如何により会計処理は大きく異なる。バージョン・アップにしても、「機能維持」のためとしながら、定義の内容からは、維持では済まず、機能の改良、強化と区別がつきにくく、解釈的たらざるを得ない規定であり、会計処理にバラつきが生ずることは否めない。

②ＦＡＳＢ（1985）は、1985年に公表されたＦＡＳＢ（Financial Accounting Standards Board）のＳＦＡＳ（Statement of Financial Accounting Standards）第86号「販売、リースその他の方法で市場に出されたコンピュータ・ソフトウェアの原価の会計処理（Accounting for the Cost of Computer Software to Be Sold, Leased or Otherwise Marketed）」で、現在の「FASB ACS 985-20-25-2 - Costs of Software to be Sold, Leased, or Marketed」（以降「ＳＦＡＳ86」と略記）で、世

[451] 櫻井(1993)p. 75

界で最初のソフトウェアに関する会計基準として、①や日本基準の参照枠であるので取り上げる。

par52.で「保守(Maintenance)」とは、「製品が顧客へ一般に出荷可能となった後、誤謬を改正し、又は製品を現在の情報に基づき更新するために実施される活動。当該活動には、定型的な変更及び追加が含まれる」とし、par6.で「関連する収益が認識される時点、若しくは、当該原価が発生した時点のいずれか早い時点において費用処理しなければならない」としている。

また、par52.で「製品機能強化（Product enhancement)」とは、「当初の製品の寿命を延長させ、又は市場性を大幅に改良することを意図した現存製品の改良。機能強化は通常、製品設計を必要とし、現存の製品のすべて又は一部の再設計を必要とすることがある」とし、par2.及びpar4.で一定の要件を満たせば資産計上しなければならないとしている。

③ＡＩＣＰＡ（1998）は、1998 年に公表されたＡＩＣＰＡ（American Institute of Certified Public Accountants）のＳＯＰ（Statement of Position）98－１「社内利用のために開発又は購入されるコンピュータ・ソフトウェアの原価の会計処理（Accounting for the Costs of Computer Software Developed or Obtained for Internal Use)」で、現在の「FASB ACS 350-40 -Internal-Use Software」（以降「ＳＯＰ98－１」と略記）で、②より少し遅れて自社利用のソフトウェアを対象として公表されたものであるので取り上げる。

par17.でソフトウェア開発のプロセスを３つに大別した上で、「完成後／運用ステージ(Post-Implementation/Operation Stage)」で発生する原価に「トレーニング（Training)」と「アプリケーションメンテナンス（Application maintenance)」を挙げている。par23.で、「内部及び外部で発生したトレーニング及びアプリケーションメンテナンスの原価は、発生時に費用処理する」としている。

Par24.では、「アップグレード（Upgrades）及び改良強化（Enhancements)」とは、「機能追加を伴う既存のソフトウェアに対する変更・修正であり、それは以前には実行不可能であったタスクを実行可能とするものである。これらは、新たな仕様や設計、既存の仕様や設計の一部又は全部の変更が求められるものである。」としている。これに関して、par73.では、「機能を追加しない単なるソフトウェアの耐用年数の延長は、資産計上すべき原価の作業というより、メンテナンス作業である。」

Par25.と Par26.では、アップグレード及び改良強化の内部及び外部（契約）で発生する原価は、par20-23.に基づいて資産あるいは費用として処理する。また、メンテナンスとマイナーアップグレードと改良強化、あるいはメンテナンスとアップグレード及び改良強化が契約により一体となっている場合には、合理的な基礎に基づき各要素に割り当て、メンテナンスの原価は契約期間に亘って費用処理すべきとする。更にpar72.で、これらが分離できない場合には、すべて発生時に費用処理すると結論付けている。

ＳＦＡＳ86 は、「保守」と「製品機能強化」を定義している。それに対してＳＯＰ98－１は、まず「アップグレード及び改良強化」を定義して、それに該当するもの以外を「メンテナンス」としているので、保守の明確な定義はなく、不分明である。そこで、両者の区別には機能追加の有無が、

重要な判断規準となる。ＳＯＰ98-１には保守の明確な定義はなくとも、この判断規準により、両者を機械的に判断可能なのに対して、ＳＦＡＳ86 は両者を定義してはいるが、定義が曖昧であり、いずれに該当するかは判然としないと言える。また、耐用年数の延長について、機能追加を伴わない場合には、いずれの会計基準を適用するかにより、分類並びに会計処理が異なる。つまりは、ソフトウェアの目的が異なるだけで、同様の事象に対して会計上の取り扱いが異なることは、整合的でなく不適切である。

（2）ソフトウェア保守に適合的な分類

ソフトウェア会計基準における保守の取り扱いには、大きな問題があることは明白である。それにはまず、ソフトウェア保守に関する適切な分類を行なうのでなければならないが、如何なる分類が適切か。現在ある比較的妥当な分類規準は、大別すれば、国際標準ＪＩＳ規格の性格分類か、又はＦＰ法で採用されているテクニカルな分類である。テクニカルな分類は何よりも実務に馴染んでおり、分類規準が明確で客観的であり、適用に迷うことがない点では優位性がある。それに対して、国際標準ＪＩＳ規格は適用に少々覚束ないところがあるが、価値的な分類の意義は逸せられないものがある。なお、総じて言えることは、両者の分類とも会計処理を意識したものではないことである。

従って、性格分類やテクニカルな分類が如何なる会計処理と対応するか、あるいは対応させるのが適合的か、相即的には導き出せないのである。そこで、筆者案を提示せざるを得ない。これまでの考察を整理するために、一覧的な図表を掲げる。それを説明する形で、考察を進めることにする。

図表１-９-1　ソフトウェア保守分類一覧

<table>
<tr><td>分類規準</td><td>国際標準
ＪＩＳ規格</td><td colspan="2">ＦＰ法</td><td>筆者案</td><td>ソフトウェア会計基準</td><td colspan="2">櫻井通晴教授</td><td colspan="2">ＳＦＡＳ86</td><td colspan="2">ＳＯＰ98-1</td></tr>
<tr><td rowspan="9">分類内容</td><td>是正保守</td><td rowspan="9">機能追加・機能変更・機能削除</td><td>機能追加</td><td rowspan="3">バグ対応</td><td rowspan="3">機能維持</td><td rowspan="3">保守</td><td rowspan="3">誤りの修正
更新
定型的な改訂</td><td rowspan="6">メンテナンス</td><td rowspan="3">誤謬の改正
更新
定型的な変更及び追加</td><td rowspan="6">アプリケーション　メンテナンス</td><td rowspan="3">機能追加のない耐用年数の延長
マイナーアップグレード</td></tr>
<tr><td>予防保守</td><td>機能変更</td></tr>
<tr><td>該当なし（但し適応保守を改釈すれば可）</td><td>機能削除</td></tr>
<tr><td>適応保守</td><td>機能追加</td><td rowspan="3">改良</td><td>著しくない改良</td><td rowspan="3">機能の改良強化</td><td rowspan="3">製品機能強化</td><td rowspan="3">機能追加を伴うアップグレード改良強化</td></tr>
<tr><td>完全化保守</td><td>機能変更</td><td rowspan="2">著しい改良</td></tr>
<tr><td>該当なし</td><td>機能削除</td></tr>
<tr><td colspan="2">該当なし</td><td>機能削除</td><td>部分的削除</td><td>該当なし</td><td colspan="2">該当なし</td><td colspan="2">該当なし</td><td colspan="2">該当なし</td></tr>
</table>

（出典：筆者作成）

前提的なことを２つ述べると、１つはＦＰ法の「機能変更」に関しては、形式的なことにだけ着

目するのではなく、内容的なことを含めて捉えなければならないことである。設計書の記述様式（要領）やプログラム標準（コーディング規約）では変更履歴を残す意味で、「修正」に関して、実際には現行仕様（旧仕様）を論理的に削除して（設計書では当該箇所に抹消線を引く、プログラムではコメント行扱いする等）、変更仕様（新仕様）を追加する場合があるが、内容から判断して、それは「機能変更」と見做すのであり、削除と追加とは見做さないのである。

２つには、１つの保守案件が複合的な内容の案件である場合があるが、作業は一纏まりで行なうとしても、それらを分割・分解して、個々の個別案件として捉えるのでなければならないことである。そうでないと、維持なのか改良なのかといった分類は混濁してしまうからである。

以上のことを前提として、ソフトウェア保守の筆者分類案を説明する。言うまでもないが、これはソフトウェアの制作目的や利用目的に関わらず、いずれのソフトウェアに対しても適用可能な分類である。

①「バグ対応」は、最低限のネガティブな維持対応として欠陥、不具合の除去を意味する。除去といっても、削除するとは限らず、誤った仕様を修正したり、漏れていた仕様を追加したりすることもある。障害を含め、誤処理を起こさないようにすることである。規模の増減はいずれもあり得るが、例え増加しても、価値の増加を意味するのではなく、正常に復したと見做すのが妥当である。

②「維持」は、何らかの環境変化に対する受動的且つポジティブな維持対応を意味する。これは、会計基準が規定を欠落させ隙間を作ってしまっている区分であり、その隙間を埋めるために設けるものである。環境変化は、業務に関わる主として外部的な変化であるが、それに起因して生ずる企業内部の変化、あるいはソフトウェアに関わる変化も含まれるが、いずれも基本的には既存の事象・事態の変更に留まるものに限定される。それらに対して、仕様を変更して対応することが維持である。

ソフトウェア会計基準では「バグ取り」と「ウィルス防止」を例示として同列に扱っているが、開発時点で既成のウイルスに対応していなければ脆弱性のバグであるが、新たに出現したウイルスを「防止」できないことはバグではなく、受動的ではあるがポジティブに堅牢とする維持である。転義的に使っている用語の元の分野に照らせば、明白なことではないだろうか。医療分野で、新種（新型）のウイルスへの対応を潜在的な医療ミスあるいは脆弱な身体への対処と看做すことなどありえないであろう。

また、「ソフトウェアが動作する環境」の変更を「著しい改良」の例示としているが、一概に改良となるわけではなく、受動的な対応で維持となる場合もある。例えば、ＯＳやプラットフォームが製造停止ないしサポート・サービス停止で利用を継続できなくなり、別のものにリプレースする場合が該当する。実際にこの例は少なくない。但し、プログラミング言語がそれ自体でサポート・サービス停止となることは滅多にないが、ＯＳやプラットフォームが当該言語をサポートしていないので、ＯＳやプラットフォームをリプレースする際に連動して言語も変更しなければならなくなることはあるので、そうした場合と見做しておく。しかも、リプレースするものが互換的であるか、

その程度如何によって、過半のコンポーネントを作り直すのであれば、保守における維持ではなく、新たな開発となるであろう。しかも、既存のコンポーネントの過半を廃棄するのであるから、既存のソフトウェアは「除却処分」を行なうのが適切で、例え既存のコンポーネントを一部残すとしても、それは新たなソフトウェアにおける再利用[452]と見做すのが妥当である。受動的な対応で、互換性の程度が多く、過半の作り直しには到らない場合は維持である。

③「改良」は、何らかの環境的もしくは自発的な、新たな事態・動向に対する、能動的な対応を意味する。主として新たな仕様を追加する場合であるが、付随的な修正等もあり得る。ソフトウェア会計基準の「ソフトウェアが動作する環境」の変更・追加に関して、能動的な対応は当然に改良である。但し、既存の技術との互換性が少ないか、それ以外の事由を含め、過半のコンポーネントを作り直すのであれば、保守の改良ではなく、新たな開発となる。これは、ソフトウェア会計基準の「著しい改良」の具体例で挙げられていた、「主要なプログラムの過半部分を再制作する」場合の程度により、保守ではなく、開発の範疇となることを指摘したことに関する応えでもある。

総じて、①②がネガティブであれポジティブであれ、あくまで維持対応であるのに対し、維持対応以外の新たな追加的対応等の全てが該当する区分である。他の基準等では、②と③の区分が曖昧で解釈的たらざるを得ない規定となっているので、明確に区別できることを意図している。

更に、④「部分的な削除」は、当該ソフトウェア全体の廃棄ではなく、機能等の部分的な削除を意味している。これはＦＰ法の「機能追加」「機能変更」「機能削除」のそれぞれの削除に該当し、纏めて１つの区分として設けたものである。会計基準には該当する規定が全くない事象である。バグ対応の除去とは異なるものであり、業務的に不要となったので取り除くのである。例えば、制度対応としては時限立法の期限切れや法律の廃止、あるいは事業における或る事業分野からの撤退、或る製品や商品の取り扱いの中止、といった場合である。これらに対応するソフトウェアが独立的であれば、全体的な廃棄となるが、基幹系や広範囲の業務系のソフトウェアの場合には、部分的な削除となる。コンポーネント単体内の部分的な削除、またはソフトウェア内のコンポーネント毎の削除で、複数もあり得るが、部分的なものである。独立的に発生する場合もあるが、②③と併行的に発生する場合もあり、ソフトウェア実務における保守案件としては一纏まりであっても、識別し、会計上は別の処理をしなければならない。これらは後述するように、会計処理が異なることを考慮し、且つ実務での適用において解釈的な違いを生じさせないための区分とすることが筆者の意図である。

（3）会計におけるソフトウェア保守の取り扱いの問題点

これまで様々な保守の分類を見てきたが、文言や定義ではなく、実際の保守案件がどの分類に該当するか、実務への適用という次元で判然としないという一層大きな問題に関して、取り上げてき

[452] ソフトウェア再利用に関しては、前述したが、本論第４章で取り扱っている。

た各分類の相違と併せて、具体的な例を挙げ、それへの適用の妥当性を問う形で考察する[453]。

（3-1）消費税率変更対応

例えば、旧聞に属するが、「消費税率の変更」を取り上げてみよう。1989 年に消費税が創設され、施行当初は税率３％であった。ところが、1997 年４月１日施行で、税率が５％に変更された。その時の対応である[454] (2014 年４月に 8%への新たな税率変更がなされたので、全くの旧聞ではないし、今回も若干のシステム・トラブルがマスメディアで報道されたように、完全にルーチン化し切れていない事象である)。

図表１-９-2　ソフトウェア保守分類と具体例（消費税率変更対応）

保守の分類規準	分類内容
国際標準ＪＩＳ規格	適応保守(但し、「運用環境変化」が制度変更という業務環境変化を包含しているとすれば、である。)
ＦＰ法	機能変更
筆者案	維持（ポジティブな維持的対応）
ソフトウェア会計基準	**該当なし**
櫻井通晴教授	保守（法律の小改正への対応）
ＳＦＡＳ86	メンテナンス（「定型的な変更及び追加」が制度変更を包含していれば、である。）
ＳＯＰ98－1	アプリケーションメンテナンス（「マイナーアップグレード」が制度変更を包含していれば、である。）

（出典：筆者作成）

図表を説明する形で議論を進めると、５％で計算するように変更することは、国際標準ＪＩＳ規格の分類では「予防保守」または「適応保守」のいずれが適当か。この場合、タイミングも影響し考慮しなければならないのである。もし税率変更の施行までに対応しなければ、施行日以降は誤計算となり「運用障害になる」ので、それから対応した場合は「緊急保守」ないし「是正保守」となる。その延長上で言えば、税率変更の制定日以降で施行日前は３％が「潜在的な障害」と言えなくはないので、その場合は税率変更対応が「予防保守」になるだろう。だが、制定日から施行日までの間の３％での計算は誤計算ではないから、税率変更施行前の３％を「潜在的な障害」と見做し、

[453] 上原(2000)の分類は筆者案と外見的に近いが、実質的な内容には相当の違いがある。同書 pp. 4-6 で保守を「システムの機能の維持」と「改善」とに大別し、前者を、①業務環境の変化への対応、②開発環境の変化への対応(開発言語の変更等)、③運用環境の変化への対応(ハードウェアをホストコンピュータから UNIX マシンへの変更等)に分けているが、余りにも拡大解釈に過ぎ、大半は後者とすべきである。また、同書 p. 5 で前者①の例で、1993 年[89 年が正しい:筆者]の消費税 3%導入、97 年の消費税 5%修正、98 年の郵便番号 7 桁化、昭和から平成への元号切り換え並びに西暦 2000 年問題を挙げており、それらを全く同列に置き機能維持としているが、余りに雑多な取扱いである。

[454] 予め断っておくが、消費税率をプログラムにおいてロジック又は内部テーブルで抱き込んでいる場合を想定した議論を行おうとしている。処理方式として、税率を外部テーブル化すれば、税率データの変更で済み、ソフトウェアの保守(変更)は必要ない。そうする場合も多いが、例えば、外部テーブル化すればディスクの I/0 が発生するので、処理性能を高く求める場合等は、プログラムに組込むこともある。外部テーブル化してもメモリー常駐させれば、ディスクの I/0 は回避できるが、アプリケーション・データをメモリー常駐させることは滅多にないので、プログラムに組込むことは非現実的な想定ではない。なお、上原(2000)でも処理方式が関係することに言及している。

それを５％に変更することを「予防保守」と見做すのは無理があろう。後知恵を外挿してはならず、リアルタイムで捉えなければならない。やはり「変化した又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」である「適応保守」が妥当だろう。但し、前述したように、注記にある「運用環境変化」が制度変更という業務環境変化を包含しているとすれば、である。ＦＰ法では「機能変更」が適切である。

ソフトウェア会計基準では「機能維持」または「著しくない改良」の何れに該当するか。「機能維持」が「バグ取り、ウィルス防止等の修繕・維持・保全」の限りでは、税率変更対応を「バグ取り、ウィルス防止」と同列に扱うのは不適切であろう。しかし、税率変更対応は「ソフトウェアの操作性の向上等」ではないだろうから、「著しくない改良」も適当とは言えない。従って、該当する分類がないと言わざるを得ない。しかし、会計処理はせざるを得ないから、会計実務ではいずれかで処理しているというのが実状であろう。

櫻井通晴教授の分類では、「保守」に定義されている「法律の小改正への対応」が適切である。ＳＦＡＳ86 では、「メンテナンス」の定義にある「定型的な変更及び追加」、ＳＯＰ98－1 では「アプリケーションメンテナンス」の「マイナーアップグレード」、が制度変更を包含しているとすれば、保守となる。しかし、規定がどの範囲までを含むのかは具体性がなく、明確でない。

総じて、ＦＰ法、櫻井通晴教授の分類を除いて、定義や規定が解釈的たらざるを得ず、具体的な保守案件がどの分類に該当するか判然としない。従って、筆者の分類では、何らかの変化に対する受動的且つポジティブな対応として「維持」の区分に該当するものと適用する。

（3-2）消費税創設対応

遡って、消費税創設時の「消費税対応」はどうだったであろうか。

図表１－９-3　ソフトウェア保守分類と具体例（消費税創設対応）

保守の分類規準	分類内容
国際標準ＪＩＳ規格	適応保守（制度創設という業務環境変化を包含していればである。）
ＦＰ法	機能追加と機能変更(消費税項目と計算仕様等の追加と既存の表示や集計等の変更)
筆者案	改良
ソフトウェア会計基準	**該当なし**（強いて拡大解釈すれば著しくない改良）
櫻井通晴教授	保守あるいは機能の改良、強化
ＳＦＡＳ86	保守あるいは製品機能強化
ＳＯＰ98－1	アップグレード及び改良強化

（出典：筆者作成）

図表に沿って説明すると、国際標準ＪＩＳ規格の分類では、「適応保守」であろう。繰り返すが、制度創設という業務環境変化を包含していればである。この場合、「新しい要求を満たすための」「改良」でも、余り抵抗はない。ＦＰ法では、「機能追加」と「機能変更」である。消費税項目と計算仕様等の追加と、既存の表示や集計等の変更となる。

ソフトウェア会計基準の分類では、「著しくない改良」であろうか。「著しい改良」ではないが、

「改良」と見做しても強い抵抗はない。しかし、「操作性の向上」といった性格ではなく、制度創設という業務環境変化への「適応」が妥当であろう。つまり会計基準には適切な規定がないのである。

櫻井通晴教授の分類では、「保守」あるいは「機能の改良、強化」のいずれか。「保守」の「法律の小改正への対応」には該当せず、ソフトウェアの「現在の価値を維持するため」の「機能維持」の「バージョン・アップ」にも該当しないであろうから、消去法的に「機能の改良、強化」となる。しかしながら、保守と機能の改良、強化の区分には「客先がついている」か否かを重要な判断規準としているが、この事象には、いずれの場合にも対応が必要であるので、どちらにも該当すると考えられる。なお、具体的事象に適用させようとすると、逆に判断規準によって迷いが生じ、不適切な判断規準であることが判明する。

同様にＳＦＡＳ86 でも、「保守」あるいは「製品機能強化」のいずれに該当するかは、定義が曖昧で判断がつかない。ＳＯＰ98－1 では、「アップグレード及び改良強化」であろう。機能追加を伴う変更・修正であり、「以前には実行不可能であったタスクを実行可能とする」ものとなるからである。

総じて、ＦＰ法、ＳＯＰ98－1 の分類を除いては、実際の保守案件への適用には、定義の範囲の曖昧さ故に解釈の余地があり、判然としない。また、上原三八（2000）の分類も妥当ではない。従って、筆者の分類では、何らかの環境的もしくは自発的な、新たな事態・動向に対して、能動的に新たな仕様を追加する対応として「改良」の区分に該当するものと適用する。

（3-3）国際会計基準対応

それならば、今後予定されている「国際会計基準対応」はいずれに該当するであろうか[455]。

図表１-９-4　ソフトウェア保守分類と具体例（国際会計基準対応）

保守の分類規準	分類内容
国際標準ＪＩＳ規格	・「完全化保守」 ・開発並びに関連ソフトウェアの「適応保守」又は「完全化保守」（会計ソフトウェアの再構築という保守の範疇ではない）
ＦＰ法	・機能追加と機能変更と機能削除 ・「新規開発プロジェクト」並び関連ソフトウェアの「機能改良プロジェクト」における機能追加と機能変更と機能削除
筆者案	改良
ソフトウェア会計基準	・著しい改良 ・開発並びに関連ソフトウェアの著しくない改良又は著しい改良
櫻井通晴教授	保守あるいは機能の改良、強化
ＳＦＡＳ86	保守あるいは製品機能強化
ＳＯＰ98－1	アップグレード及び改良強化

（出典：筆者作成）

図表に沿って説明すると、国際標準ＪＩＳ規格の分類では、「完全化保守」か、あるいは会計ソフトウェアの再構築というソフトウェア保守の範疇ではない開発並びに関連ソフトウェアの「適応保

[455] これは国際会計基準対応を全く想定せずに開発し今後対応する場合の想定であり、ＳＡＰ等は対象としていない。現在、実際に運用されている会計ソフトウェアの大半は該当するであろう。

守」または「完全化保守」であろう。ＦＰ法では、「機能追加」と「機能変更」と「機能削除」か、あるいは会計ソフトウェアの再構築という「新規開発プロジェクト」並び関連ソフトウェアの「機能改良プロジェクト」における「機能追加」と「機能変更」と「機能削除」であろう。

ソフトウェア会計基準では、「著しい改良」か、あるいは会計ソフトウェアの再構築という「開発」並び関連ソフトウェアの「著しくない改良」又は「著しい改良」であろう。この場合、国際会計基準対応という一連の対応において、「開発」や「著しい改良」は研究開発扱いをしなければならず、「著しくない改良」はそうしないことになる。あるいは包括的に研究開発扱いになってしまうのであろうか。コンバージェンスによって近いものとなってはいるが、これまでの日本基準とは相当に異なるので、容易いとは言えないが、少なくとも原則規定としては明確な国際会計基準のソフトウェアへの仕様化並びに実装が研究開発として扱われるのは妥当とは思えない。

櫻井通晴教授の分類では、「保守」あるいは「機能の改良、強化」のいずれか。定義では、「保守」には該当せず、「機能維持」の「バージョン・アップ」にも該当しないので、「機能の改良、強化」となる。しかし、前述のように、保守と機能の改良、強化を区別する客先の有無の判断規準を勘案すると、逆に判断がつかず、判断規準が不適切であるように思う。

同様にＳＦＡＳ86 でも、「保守」あるいは「製品機能強化」のいずれに該当するかは、定義が曖昧で判断がつかない。ＳＯＰ98－1 では、「アップグレード及び改良強化」が適切であろう。

総じて、ＦＰ法、ＳＯＰ98－1 の分類を除いては、やはり定義の範囲の曖昧さ故に解釈の余地があり、判然としない。従って、筆者の分類では、前述の消費税制度創設と同類の事象と捉え、「改良」の区分に該当するものと適用する（消費税創設対応といった個別案件に比べれば、はるかに規模の大きな改良になるが、その区別を行なう必要はない）。但し、全面再構築とする場合は、保守ではなく、開発である。

３．ソフトウェア保守に適合的な会計処理

これまでの保守の分類に関する考察を踏まえて、個々の区分に如何なる会計処理が適合的かを考察する。但し、国際標準ＪＩＳ規格とＦＰ法に関しては、会計処理の一覧には掲げていない。これらは、ソフトウェア分野において保守を取り扱うものであり、言うまでもなく、会計処理を取り扱ってはいないからである。

図表の筆者案に沿って、会計処理を説明する。①「バグ対応（欠陥、不具合の除去）」は、規模の増減に関わらず、体化していると見做されていたが（資産計上済み）実際には毀損していた価値の回復（実現）であるから、資産価値は変わらないので、費用処理が妥当である。これは、ソフトウェア会計基準やその他の基準等と、取り敢えず同様である。但し、留意しなければならないことがある。委託開発した場合、委託契約にはほぼ必ず瑕疵担保責任条項があり、規定の期限内に顕現した瑕疵すなわちバグに関しては、委託先が無償で対応することになる。従って、その場合は委託元

では費用が発生せず、費用処理は行なわないのである（なおこの点は、図表には記載していない）。これに関しては、いずれの基準等も全く留意していない。この点は、筆者案が異なるところである。

図表１−９−5　ソフトウェア保守分類と会計処理一覧

<table>
<tr><th colspan="2">筆者案</th><th colspan="2">ソフトウェア
会計基準</th><th colspan="3">櫻井通晴教授</th><th colspan="2">ＳＦＡＳ86</th><th colspan="2">ＳＯＰ98−1</th></tr>
<tr><th>分類</th><th>会計
処理</th><th>分類</th><th>会計
処理</th><th colspan="2">分類</th><th>会計
処理</th><th>分類</th><th>会計
処理</th><th>分類</th><th>会計
処理</th></tr>
<tr><td>バグ対応
（欠陥、不具合）
の除去</td><td>費用
処理</td><td>機能維持</td><td>費用
処理</td><td>保守</td><td>誤謬の
修正

更新

定型的な
改訂</td><td>費用
処理</td><td>メンテナンス</td><td>誤謬の
改正

更新

定型的な
変更及び
追加</td><td>費用
処理</td><td>アプリケーションメンテナンス</td><td>機能追加のない耐用年数の延長

マイナーアップグレード</td><td>費用
処理</td></tr>
<tr><td>維持
（ポジティブな維持）</td><td>資産
計上</td><td colspan="2">該当なし</td><td colspan="2">機能維持</td><td></td><td colspan="2"></td><td colspan="2"></td><td></td></tr>
<tr><td rowspan="2">改良</td><td rowspan="2">資産
計上</td><td>著しくない
改良</td><td>資産計上
（要件充足）</td><td colspan="2" rowspan="2">機能の
改良強化</td><td rowspan="2">製造
原価</td><td colspan="2" rowspan="2">製品
機能強化</td><td rowspan="2">資産
計上
（要件
充足）</td><td colspan="2" rowspan="2">機能追加
を伴う
アップ
グレード
改良強化</td><td rowspan="2">資産
計上
（要件
充足）</td></tr>
<tr><td>著しい
改良</td><td>大半費用処理、一部資産計上</td></tr>
<tr><td>部分的
削除</td><td>除却処理
（資産の
削減）</td><td colspan="2">該当なし</td><td colspan="3">該当なし</td><td colspan="2">該当なし</td><td colspan="2">該当なし</td></tr>
</table>

（出典：筆者作成）

②「維持（ポジティブな維持）」は、ソフトウェア会計基準には該当する規定が欠落しているものである。バグを混在させた「修繕・維持・保全」とは似て非なる規定であり、且つ「改良」とは別に独自に設定するのが妥当なものである。有形資産の会計基準における取り扱いとは異なり（そもそもソフトウェアには物理的な経年劣化等はない）、既存の自ら体現している仕様の変更による環境への適応である。また、瑕疵へのネガティブな対応ではなく、あくまでポジティブな維持であるという性格からすれば、資産計上が妥当である、と考える。他の基準等では、費用処理する規定となっているが、経済的（業務的）な陳腐化を回避し、適応的になることで、且つそれにより経済的な耐用年数が延長することで資産価値の増加をもたらす事象であるので、資産計上とする。

③「改良」は、維持対応ではない追加的対応なので資産価値の増加であることは確かだが、ソフトウェア会計基準のように「著しくない」と「著しい」に区分せず、一纏まりの事象として捉え、全てを同じ取り扱いで資産計上とすることが妥当であると考える。

④「部分的な削除」は、業務的に不要となったので取り除く、機能等の部分的な削除である。従って、ソフトウェア資産から削減する資産価値の減少なので、「除却処理」が妥当であり、資産の減額（未償却残高がある場合）と所要費用の費用処理をすることになる[456]。

[456] 特別損失ではなく、費用処理とすることに関しては、後続の第 10 章で取り上げる。

本章は、ソフトウェアにとって重要な保守を主題的に取り上げ、ソフトウェア会計基準における保守の取り扱いを問題視し、保守に適合的な会計処理に関する考察ないし提案を行なった。まず、ソフトウェア保守とは何かを明確にするために、ソフトウェア・ライフサイクルに占める期間とコストの面から保守を概観した。また、保守に関する分類を取り上げ、保守の特性を更に明確化した。次に、会計の現状における保守の取り扱いに関して、日本基準だけでなく、アメリカ基準等をも併せて、分類とそれに基づく会計処理を取り上げることにより、明らかにした。これらに対して、ソフトウェア保守には如何なる分類が適合的か、取り上げた分類を比較評価し、問題点を明確にすることを通して、優位的な筆者案を提示した。更に、その分類に適合的な会計処理案を提示した。

筆者の分類案は、ソフトウェア会計基準が規定を欠落させ、隙間を作ってしまっている「維持」という区分を設けたことが大きな特徴である。これは、何らかの環境変化に対する受動的且つポジティブな維持対応を意味する。この区分と、最低限の維持対応である欠陥、不具合の除去を意味する「バグ対応」の区分とは、截然と異なったものである。また、「改良」の区分は、上記２区分がネガティブであれポジティブであれ、維持対応であるのに対し、維持対応以外の新たな追加的対応の全てが該当する区分とした。これらの区分のいずれに該当するかについて、他の会計基準等では規定が曖昧で解釈的たらざるを得ないので、適否を明確に区別できるものとした。しかも、会計処理に直結し、実際の適用に解釈の余地を残さないような分類を提示した。更に、長期間に亘る保守では当然に起こり得る機能等の部分的な削除を意味する「部分的な削除」の区分を設けたことも大きな特徴である。これは、会計分野における保守の取り扱いでは全く欠落していたものであり、不可欠な補充である、と考える。会計処理として、ポジティブな「維持」を資産計上とすることは、それ自体の経年劣化等が生じないソフトウェアの特性と環境変化への適応的な維持、そしてそれにより利用（寿命）が継続延長できることを考慮すれば、資産計上とすることが妥当である、と考える。

なお、測定に関しては立ち入らなかったが、それには理由がある。開発と保守で測定が大きく異なるのは、見積りである。保守の場合には、既存の母体を考慮し、また保守特有のアクティビティを考慮して見積りを行なわなければならない。一般的には、同じ規模であれば、保守の方が生産性は低くなる。しかし、会計処理が対象とするのは、見積りではなく、実績である。そして、実績の測定に関しては、保守は開発と異ならない。従って、本論第３章で既に開発の測定を取り扱っているので、本章では立ち入らなかったのである。会計実務として具体的に考慮すべきことがあるとすれば、保守案件を性格分類別の個別案件に分けて測定を行なうこと、年間保守契約等の場合でも個々の案件内容が判明している限り、それに即した工数等に基づく按分を行なうことにすればよい、と考える。

本論

第10章　ソフトウェア廃棄

１．ソフトウェア廃棄の概観

（1）ＪＩＰＤＥＣ「システム管理基準」における廃棄

（2）ＩＰＡ「共通フレーム」における廃棄

（3）ソフトウェア廃棄における各種ドキュメント

（3-1）廃棄計画書

（3-2）廃棄手順書

（3-3）廃棄結果報告書

（3-4）廃棄後の管理資料

２．ソフトウェア廃棄に係る会計処理

（1）現行会計基準における会計処理

（2）櫻井通晴教授の会計処理案

（3）ソフトウェア廃棄に適合的な会計処理

３．ソフトウェア廃棄に伴う情報開示

（1）ＩＴ投資の事後評価を巡る現況

（2）ソフトウェア廃棄に伴う情報開示

（3）ソフトウェア・ライフサイクルの総括情報

第10章　ソフトウェア廃棄

ソフトウェア・ライフサイクルにおいて、大フェーズとしては最後に当たるソフトウェア廃棄を取り上げる。ソフトウェア・ライフサイクル会計研究は、これによって一応完結することになる。

ソフトウェア廃棄は、１．で概観するように、比較的短期間の作業であり、内容的にも比較的僅かな作業があるだけである。そうした意味で、会計的にも捕捉しやすい対象であり、現行の会計処理にも特段の問題があるわけではない。

しかし、筆者からすれば、これまでほとんど全くと言っていいほど着目されてこなかったことであるが、ソフトウェア廃棄に伴い行なうべき情報開示があると考えている。これをもって、ソフトウェア・ライフサイクルは会計的には完結するのである。まずはソフトウェア廃棄を概観し、その中でソフトウェア廃棄において作成するドキュメントをも取り上げる。次にソフトウェア廃棄に係る現行の会計処理を確認し、遡って櫻井通晴の先行研究を取り上げ、それらを踏まえてソフトウェア廃棄に適合的な会計処理を再確認する。そのあと、情報開示の前提的ないし背景的な状況としてＩＴ投資の事後評価を巡る現況を瞥見する。それを踏まえて、どのような情報開示を行なうべきかを提言し、更にその開示すべき情報を詳細に提示し、筆者の情報開示案を具体化する。

１．ソフトウェア廃棄の概観

ソフトウェア廃棄に関して、まずＪＩＰＤＥＣ「システム管理基準」並びにＩＰＡ「共通フレーム」という各標準においてどのように取り扱われているかを取り上げる。次に、それを踏まえて、ソフトウェア廃棄で作成するドキュメントを補強するものを含めて整理する。

（1）ＪＩＰＤＥＣ「システム管理基準」における廃棄

最初に取り上げるのは、ＪＩＰＤＥＣ「システム管理基準」であり、その中で廃棄がどのように取り扱われているか、確認する。

「情報システムの廃棄」は、それ自体が「保守業務」における中項目であり、２つの小項目で構成されているだけである[457]。内容的には、廃棄計画の策定（責任者の承認含む）と、不正防止及び機密保護対策の考慮である。両者を通じて廃棄に伴うリスク管理が強調されている。リスクとしては、例示だが、「個人情報、機密情報等の漏えい」[458]が想定されている。

ソフトウェア廃棄が独立的に取り扱われていないこと、仮にいずれかに含めるならば、保守ではなく、運用の方が次善であることは、ソフトウェア基礎論第２章「ソフトウェア・ライフサイクル」で論及したので繰り返さない。

---

[457] 経済産業省商務情報政策局(2005)pp. 12, 384-385

[458] 同書 p. 384

なお、廃棄において作成するドキュメントとして想定しているのは、明示的ではないがリスク評価[459]のワーキング・ドキュメントと、「廃棄計画」[460]書である。

（2）ＩＰＡ「共通フレーム」における廃棄

次に取り上げるのは、ＩＰＡ「共通フレーム」であり、その中で廃棄がどのように取り扱われているか、確認する。

最初の 1994 年版「共通フレーム」では、「旧システムの廃棄」は、「6. 保守プロセス」における最後の１つのアクティビティとして位置付けられ、5 つのタスクで構成されている[461]。タスクは、「6.6.1　廃棄計画の策定（業務運用者も参加）」、「6.6.2　廃棄計画およびアクティビティの業務運用者への通告」、「6.6.3　新システムと旧システムの並行運用期間の設定と業務運用者の訓練」[462]、「6.6.4　廃棄時の業務運用者およびサポート要員への通告」、「6.6.5　廃棄データの安全性確保」である[463]。廃棄しても、「システムおよび付随文書の保管」、「開発時の文書、ログ、コードは保管する」[464]としている。また、「ソフトウェアを廃棄しても残るサポート責任の明確化」とあるが、それ以上の説明がないので、どういう「責任」かは詳らかでない。「安全性確保」ということでは、「廃棄されたソフトウェアで使われていたデータは、データ保護やデータに対する監査のための組織的な要件に従う」[465]、とある。

なお、廃棄において作成するドキュメントとして想定しているのは、「廃棄計画」[466]である。

次の 1998 年版「共通フレーム」では、「システム又はソフトウェア廃棄」は、「1.6　保守プロセス」における最後の１つのアクティビティとして位置付けられ、同じく 5 つのタスクで構成されている[467]。タスクは、「1.6.6.1　廃棄計画の立案」、「1.6.6.2　廃棄計画等の利用者への通知」、「1.6.6.3 新旧ソフトウェア製品の並行運用と利用者の教育訓練」、「1.6.6.4　関係者全員への廃棄の通知」、「1.6.6.5　廃棄関連データの保持と安全性確保」である[468]。文言等に多少の異同はあるが、1994 年版と実質的にはほぼ同じものである。従って、1994 年版で指摘した問題点も解消されていない。

---

459 同書 p. 384
460 同書 p. 384
461 共通フレーム検討委員会編(1994) pp. 72-73
462 ソフトウェアの廃棄が、必ず新システムへのリプレースを伴うと想定しているが、そうしたケースが多いとはいえ、必ずセットになっているとは限らないし(6.6.1 の d)では「該当すれば」(p. 73)という条件付きにはなっているが、6.6.3 では条件付きとはなっていない、並行運用を必ず行なうわけではないし、「業務運用者の訓練」は新システムに関することで、廃棄のアクティビティではないことなど、特定の想定に基づいた混濁した内容で、とても標準の体を成していない記述である。
463 同書 p. 73
464 同書 p. 73
465 同書 p. 73
466 同書 p. 73
467 SLCP-JCF98 委員会編(1998) pp. 174-175
468 同書 pp. 174-175

なお、廃棄において作成するドキュメントとして想定しているのは、「廃棄計画」[469]である。

３番目の 2007 年版「共通フレーム」では、「システム又はソフトウェア廃棄」は、「1.8　保守プロセス」における最後の１つのアクティビティとして位置付けられ、同じく５つのタスクで構成されている[470]。タスクも、1998 年版と項番が変更になっているだけなので、掲示は省略する。1994 年版並びに 1998 年版と実質的にはほぼ同じものである。従って、1994 年版で指摘した問題点も解消されていない。ほぼ唯一、「廃棄の分析では、次のことを判断することが望ましい」として、「・時代遅れの技術の保持」等の 6 項目のチェック項目が増補[471]されていることが目立った改訂と言える[472]。

なお、廃棄において作成するドキュメントとして想定しているのは、「廃棄計画」[473]である。

最新版の 2013 年版「共通フレーム」では、「廃棄プロセス」は、1994 年版～2007 年版と異なり、「３．運用・サービスプロセス」の小プロセスというように位置付けが変更されている（保守→運用)。「3.2.1　システム又はソフトウェア廃棄計画」、「3.2.2　廃棄の実行」という２つのアクティビティで構成され、各々のアクティビティは 1、5 タスクで構成されている[474]。但し、位置付けや構成は変更されているが、実質的な内容は 2007 年版と大差ないのである。それでも、「この共通フレームの廃棄プロセスは、ISO/IEC 15288 の廃棄プロセスの特化である」[475]とのことで、廃棄に関してだけは準拠標準を異にしており、その説明を最初に追加している。また、廃棄計画に３項目の「スケジュール、行動及び資源を定義する」との注記[476]が追加されている。

なお、廃棄において作成するドキュメントとして想定しているのは、「廃棄戦略」[477]と「廃棄計画」[478]である。

以上を纏めて一覧化すると、次の図表のようになる。新システムの「並行」運用は廃棄のアクティビティないしタスクではなく、新システムの新たなアクティビティないしタスクと位置付けるべきであろう。また、廃棄後のことを配慮して、「不正防止及び機密保護対策」又は「安全性確保」を

---

469 同書 p. 174

470 ＩＰＡ編(2009) pp. 154-155

471 同書 p. 155。なお、「廃棄の分析」と言いながら、「新しいソフトウェア製品」の在り様を取り上げた項目が大半を占め、新しいソフトウェアの企画に関することとが混在しており、廃棄に関することでは参考になることはないので、立ち入らない。

472 正確を期せば、もう１つ、「システム又はソフトウェア廃棄アクティビティを実施するための入力と出力」という説明記述が追加されている(p. 155)。

473 同書 p. 154

474 ＩＰＡ監修(2013) pp. 192-194

475 ISO/IEC 15288 はシステムライフサイクルプロセスの標準であり、同フレームは全般的には ISO/IEC 12207:2008(JIS X 0160:2012)というソフトウェアライフサイクルプロセスの標準に準拠しているので、特に断り書きをしているのである。

476 同書 p. 193。３項目とは、「ソフトウェアサービスの提供の終了」、「社会的及び物理的に受入れ可能な状態へのシステム形態の変換又はシステムの保持」、廃棄活動に関わる「物理的資材及び情報の長期的状態に適用できる健康、安全、セキュリティ及びプライバシの考慮」(p. 193)である。

477 「廃棄計画」とは別に、独立的に、「廃棄戦略を定義し、文書化する」とされている(同書 p. 193)。

478 同書 p. 193

アクティビティないしタスクとしていることは特徴的である。それ以外のことは、廃棄に関して特段の予備知識がなくとも、容易に理解できるアクティビティないしタスクであろう。なお、欄外に保守又は運用と表記したのは、廃棄がいずれの大フェーズに位置付けられているかを示したものである。

図表 1-10-1　廃棄アクテイビティ（タスク）並びにドキュメント一覧

	システム管理基準	共通フレーム			
		1994年版	1998年版	2007年版	2013年版
アクティビティ又はタスク	廃棄計画策定 不正防止及び機密保護対策の考慮	廃棄計画の策定 廃棄計画およびアクティビティの業務運用者への通告 新システムと旧システムの並行運用期間の設定と業務運用者の訓練 廃棄時の業務運用者およびサポート要員への通告 廃棄データの安全性確保	廃棄計画の立案 廃棄計画等の利用者への通知 新旧ソフトウェア製品の並行運用と利用者の教育訓練 関係者全員への廃棄の通知 廃棄関連データの保持と安全性確保	同左	システム又はソフトウェア廃棄計画 廃棄の実行
ドキュメント	リスク評価のワーキング・ドキュメント 廃棄計画	廃棄計画	廃棄計画	廃棄計画	廃棄戦略 廃棄計画
	←	保守		→	← 運用 →

（出典：『システム管理基準』並びに『共通フレーム』各版から項目抽出、表化筆者）

（3）ソフトウェア廃棄における各種ドキュメント

ソフトウェア廃棄に関するドキュメントは、（1）と（2）で挙示したようにそれほど多くない。以下に、若干補足する形で再整理する。

（3-1）廃棄計画書

ワーキング・ドキュメントとしてリスク評価に関する資料や、それとは性格を異にする「廃棄戦略」に関するドキュメントを作成することが（1）と（2）で挙げた標準では推奨されているが、一般的にはそうした内容を含めるかあるいは含めずに、廃棄計画書が作成される。内容的には、対象システム又はソフトウェア、その詳細な各種資源、廃棄スケジュール、廃棄体制、廃棄後の保管資源並びに保管方法等である。新システムとの並行運用を行なう場合には、それを追加する場合もあり得るが、それはどちらかと言えば新システムの移行計画書の方で旧システムとの並行運用として取り扱われる場合が多いであろう。

（3-2）廃棄手順書

廃棄手順書は、廃棄の具体的な手順、方法を記述するものである。何らかの意図せざる事情で廃

棄を延期し、一定期間再び運用を続けるために、一旦廃棄したシステム又はソフトウェアを利用可能な状態に復元させる手順をも慎重を期して記述する場合も少なくない。

なお、廃棄手順書は独立的に作成する場合と、廃棄計画書に廃棄手順内容を盛り込み、独立的には作成しない場合がある（前に挙げた標準の「廃棄計画」は廃棄手順内容を含めたものと解せられる)。いずれにせよ、廃棄手順の明確化は廃棄に関して必須のものである。

（3-3）廃棄結果報告書

実際に遂行した廃棄作業を纏めた報告書であり、形式は様々であれ、作成するものである。廃棄計画書に実績を記載するスペース（欄）を用意して、そこに記載する様式の場合もある（前に挙げた標準では明示的ではないが、この様式と解せられる)。特に、全てを消去するのではなく、データ等の資源を残し保管する場合には、その内容（対象資源の明細、保管場所、保管方法・媒体等）の記載が重要である。但し、それに関しては別途保管に関する独立的なドキュメントとして作成する場合もある。

（3-4）廃棄後の管理資料

それが、この括りの意味するものであり、保管資源一覧ないし台帳といった形で作成する。また、廃棄後も何らのサポートを行なう場合には、それへの対処方法等に関する資料を作成する（前に挙げた標準では明示的にはないが、「不正防止及び機密保護対策」又は「廃棄データの安全性確保」を配慮し、廃棄後も何らかのサポートを行なうことを考えれば、必要なものであろう)。

２．ソフトウェア廃棄に係る会計処理

１．の概観を踏まえて、ソフトウェアに係る会計処理を取り上げる。まずは現行の会計処理がどのようになっているかを取り扱うが、その際現行のソフトウェア会計基準は独立的に廃棄に関する規定を設けておらず、それに準拠するだけでは実際の処理は行なえないので、企業会計原則ないし法人税法に準拠して処理を行なっているはずである。そうした会計処理を取り上げることにする。次に、ソフトウェア会計基準設定前であるが、櫻井通晴教授が廃棄に係る会計処理を取り扱っており、筆者の調査の限りでは、それがほぼ唯一の先行研究と言えるものなので、取り上げる。その後、それらを踏まえて、ソフトウェア廃棄に適合的な会計処理を整理し、確認する。

（1）現行会計基準における会計処理

ソフトウェア資産の廃棄に係る処理[479]は、大別すると、未償却残高がある場合と、償却済の場合

[479] ソフトウェアを当初費用処理したのであれば、廃棄に関しては、ソフトウェア自体の会計処理としては明示的には行なわず、廃棄に係る作業等に費用が掛かれば、それも費用処理をすることにな

とで分かれることになる。未償却残高がある場合には、廃棄に関して費用が発生すれば、それを含めて除却損（特別損失）として処理する。償却済の場合には、ソフトウェア資産自体に係る会計処理は明示的には行なわず、廃棄に関する費用が発生すれば、それのみの処理となるが、金額がそれほど高額になることはないので、処理の簡便さもあって、運用業務全般において発生する費用の一部として、それに含めて処理されることになるであろう。

なお、未償却残高がある場合、上記のように処理することは廃棄を「非経常的な性質」と解した場合のことであるが、多くのソフトウェアを制作（又は購入）し運用している場合には個々のソフトウェアは新たに追加されたり途中で廃棄されたりするという流れの中にある１つであり、廃棄を「経常的に生じている場合」と解せられるのであり、その場合には除却損（特別損失）ではなく、営業外損失として処理することになる[480]。

更に、税務会計からの逆規定的な処理として、法人税法基本通達に準拠して会計処理を行なっていることが少なくないと解せられる。ソフトウェアの廃棄に係る規定は、次の通りである。「ソフトウエアにつき物理的な除却、廃棄、消滅等がない場合であっても、次に掲げるように当該ソフトウエアを今後事業の用に供しないことが明らかな事実があるときは、当該ソフトウエアの帳簿価額（処分見込価額がある場合には、これを控除した残額）を当該事実が生じた日の属する事業年度の損金の額に算入することができる。（平 12 年課法 2－19「九」により追加）」（7－7－2 の 2）。掲示されている例示は２つある。１つは、「(1)　自社利用のソフトウエアについて、そのソフトウエアによるデータ処理の対象となる業務が廃止され、当該ソフトウエアを利用しなくなったことが明らかな場合、又はハードウエアやオペレーティングシステムの変更等によって他のソフトウエアを利用することになり、従来のソフトウエアを利用しなくなったことが明らかな場合」である。もう１つは、「(2)　複写して販売するための原本となるソフトウエアについて、新製品の出現、バージョンアップ等により、今後、販売を行わないことが社内りん議書、販売流通業者への通知文書等で明らかな場合」である。これによって、ソフトウェア廃棄は、例え未償却残高がある場合でも、損金算入（費用処理）を行なうことができる。この場合も、「経常的に生じている場合」と同様、廃棄に係る費用を含めて、営業外損失として処理することになろう。なお、これに関しては、基本通達の例示にもある通り、ソフトウェア会計基準の分類で言えば、「市場販売目的のソフトウェア」と「自社利用のソフトウェア」のいずれであっても、全く同様の処理をすることになる。

蛇足に近い補足をすると、ソフトウェアの「常識」に属することなので、これまで特に触れなかったが、無用の疑義が生ずるといけないので一応触れておくと、ソフトウェアは廃棄する場合に売却することはまずない。従って、売却に係る会計処理を考慮する必要はない。また、廃棄後保管をしておいた資源を再利用するということは全くないとは言えないが、それに関しては第４章「ソフ

る（但し、後述するように、その場合でも単独で取り扱うのではなく、運用に係る一連の費用に含めて処理しているであろう）。

480 桜井(2013)pp. 188-189 参照

トウェア再利用」で取り扱っているので、廃棄では考慮する必要はない。

（2）櫻井通晴教授の会計処理案

櫻井通晴教授は、ソフトウェア会計基準設定前ではあるが、管見の限りほぼ唯一と言ってよいが、廃棄に係る会計処理を考察している。「ソフトウェアの技術革新はきわめて早い。それゆえ、ソフトウェアは不良資産になる可能性が高いから、資産計上を控えるべきだとの意見がある。確かにそのとおりである。しかし、ソフトウェアの除却が制度化されるならば、不良資産の不安はなくなるはずである。そこで、ソフトウェア会計基準としては、資産計上を許容するならば、それと同時に、使用または販売に供しえないことが客観的証拠によって認定されたソフトウェアは、会計上、速やかに除却されうるような除却の基準を明文化する必要がある」[481]。櫻井の論述構成は、ソフトウェアの資産計上に消極的な意見（技術革新→旧技術の不良資産化を理由とする）に対して、資産計上と除却をセットにすることで、不良資産化を回避可能とし、ソフトウェアの資産計上を「許容」させようという戦術的な意見の提示となっている。そして、周知の通り、ソフトウェア会計基準は資産計上を「許容」したが、「除却の基準を明文化」することはしなかった。

続いて櫻井は、「客観的な証拠によって販売に適さないと認定されたソフトウェア」のケースを４つ（「その他」を含めれば５つ）列挙し[482]、「客観的な証拠によって使用に供しえなくなったソフトウェア」のケースを５つ（「その他」を含めれば６つ）列挙している[483]。同書の「ソフトウェア会計実務指針〔案〕」に対するＱ＆Ａという別の箇所でも、同じ内容を要約的に再掲している。「早期除却」の例示としては、今日的にもほぼ妥当なものと言えるが、特にその内容の当否が重要とも思えないので、引用は控える。そして、櫻井は「ソフトウェア会計実務指針〔案〕」として、「除却の会計処理」は、「使用または販売に供しえないことが客観的な証拠によって認定されたソフトウェアは、会計上、速やかに除却しなければならない。」[484]、とする。眼目とするところは、次のようなことである。当時の法人税基本通達では、損金算入できる除却のケースが限定されているので[485]、櫻井が列挙したケースが該当するとは限らないので、「ソフトウェアの性質に鑑みて、除却のケースが資産の滅失と契約の解約に限定されるわけではないことはいうまでもない。不要な混乱を避けるためにも、除却のケースを経営実態に合わせて拡張する必要がある」[486]、ということである。つまりは、

---

[481] ソフトウェアの資産性に関する検討委員会・委員長櫻井通晴、櫻井編著(1993)p. 20
[482] 同書 pp. 20-21
[483] 同書 p. 21
[484] 同書 p. 61
[485] 同書の「注１」で、次のように典拠を含めて取り上げられている、「現在の法人税基本通達では、「繰延資産とされた費用の支出の対象となった固定資産又は契約について滅失又は解約等があった場合」（基通８－３－６)についてのみ、「当該繰延資産の未償却残高を損金の額に算入する。」（基通８－３－６)ことができるケースとして例示列挙されているにすぎない」(p. 61)。なお、同基本通達がその後改正され、現在では(１)で引用したような内容になっている。
[486] 同書 p. 61

ソフトウェアの「早期除却」に関して、「客観的な証拠」がある場合には、特別損失としてではなく、費用処理（損金算入）とすることを提案しているのである（この限りでは、その後の基本通達の改正により、櫻井の主張はほぼ達成されたと言えるであろう）。

櫻井の問題設定に即する限り、それほどの異論はないのだが、専ら「早期除却」だけを問題とする偏った論述構成を、筆者はそもそも支持できない。その前提となるソフトウェアの技術革新による「早期」の陳腐化という想定自体が一面的である。櫻井は、「現在の技術水準を正当に評価すれば、販売用ソフトウェアの償却期間は3年、社内利用ソフトウェアの償却期間は5年が概ね妥当である」[487]としており、これは現行のソフトウェア会計基準に継承されている（償却期間に関しては税法に準拠していると言うべきだが）。更に、「基本ソフトウェアや販売用のアプリケーション・ソフトウェアなど技術的変化の激しいソフトウェアについては、現在よりも大幅な耐用年数の短縮を図るという方向を取らざるを得ないかもしれない」[488]、とまで言っている。しかし、ソフトウェアの一部には確かにそうした短命なものがあることは事実だが、ソフトウェアは全域的には非常に長命で30年を超えて使用されているものも実際にあり、相当の期間的な幅があるとするのが穏当な偏りなく捉えた実態である[489]。専ら短命と見るのは著しく偏った見方である[490]。従って、「早期除却」を含め、いわば寿命を全うした場合、あるいは更に想定外に延命した場合をも包含した、ソフトウェア廃棄に係る会計処理を総合的に考慮しなければならない、というのが櫻井の見解に対する筆者の対抗的な見解である。

（3）ソフトウェア廃棄に適合的な会計処理

まず予め言及しておきたいことは、ソフトウェア廃棄に係る会計処理を考える場合に、ソフトウェアのライフサイクルを全域的に想定しているわけではない現行のソフトウェア会計基準や、個々

---

487 同書 p. 61

488 同書 p. 63

489 昨今の身近な周知の事例を取り上げると、格好の例として Windows XP が挙げられる。2001 年に出現し、ついに 2014 年 4 月にサポート停止となったが、10 年を遥かに超える期間、企業を中心に広範に利用されてきた。リプレースの予算等の理由で、サポート停止後も引き続き利用する企業等が少なくないことは、サポート停止前後のマスメディア報道でも話題になったくらいである。クライアント系に限っても、後継ＯＳとして、Windows Vista、Windows 7、 Windows 8 が出現したが、特にリプレースの必要を感じず(メジャー・バージョンアップはオーバー・スペック気味である)、Windows XP は利用され続けた、と言える。近時のサポート停止にしても、セキュリティ対応の困難さを理由に挙げているが、主として後継ＯＳのための営業政策的な性格が濃厚であると思えてならない。比較的最近の良く知られた事例なので取り上げたが、このように長期に利用されることに関して、Windows XP が特に例外的というわけではない。まして、競争的な性格が遥かに少ない自社システムの寿命は意外に長いのである。

490 そういう意味で、筆者は現行のソフトウェア会計基準の減価償却の期間に関しても、改訂が必要であると考えているが、それを本格的に取り扱うには、産業政策的な意図や財政事情とも密接に関係する税法をも視野に入れなければならず、本論文の枠組みとは少々異なる論述構成を必要とするので、別途独立的に主題化して取り上げることにしたい。今後の課題とすることで、ここでは立ち入らない。

のソフトウェアを必ずしも総合的に捕捉しているわけではない会計諸規範（税法等を含む）に準拠するのではなく、これまでソフトウェア・ライフサイクルに沿って考察し、提言してきたことの延長上で、ソフトウェア廃棄に係る会計処理を考えるということである。

開発時に関しては、本論第６章で既に仕損処理を提言した。これによって、（２）で取り上げた櫻井が「不良資産の不安」という事態は、少なくとも開発終了時点（運用開始時点）では対処済なのである。保守・運用期間に関しては、減損会計によって対処可能である。あるいは、企業会計基準第 24 号「会計上の変更及び誤謬の訂正に関する会計基準」における減価償却の取り扱い(20.，57.，61.～62.）で対処することもできる。更に、本論第７章システム運用改善の会計処理で取り上げたような運用改善を行ない、それに係る会計処理を行なってきたとする。そうであれば、「不良資産」であったり、杜撰な取り扱いにより突発的に廃棄することは限られてくるであろう。未償却残高がある場合であっても、社内外のそれ相当の理由により廃棄することになるとするのが穏当である。

未償却残高がある場合と償却済の場合、いずれに関しても、「除却費」として費用処理を行なうことが適合的であると筆者は考える。未償却残高がある場合は、その金額と廃棄に係るそれ以外の費用（廃棄計画並びに実施に関する人件費あるいは外部委託費その他、廃棄に関する企画業務を行なったのであれば、それを含む）を含める。償却済であれば、廃棄に係る費用が該当する。「除却費」と言っても、特別損失ではなく、営業外費用とする（廃棄により利用を止めるのであるから、販管費といった営業費用とするのは適当ではない)。従来、運用の一環として位置付けられてきたが（保守の一環とするのは明らかに誤りである)、ソフトウェア・ライフサイクルとして、独立的な大フェーズとする位置付けからは、運用費用の一部として処理することは適当ではなく、廃棄ということが明示的な処理をすることが妥当である。

３．ソフトウェア廃棄に伴う情報開示

ソフトウェア廃棄というライフサイクルの終了時点で、当該ソフトウェアがどのような生涯を辿ってきたのかを総括し、その情報を開示する必要があると、筆者は考えている。まずは、ＩＴ投資の事後評価を巡る現況を瞥見する。何故そうするかと言えば、開発完了以降且つ廃棄以前に、どの程度の情報が捕捉されているかを知るためである。それを踏まえた上で、どのような情報開示を行なうべきかを考察し提言する。更にその開示すべき情報を詳細に挙示し、筆者の情報開示案を一層具体化する。

（１）ＩＴ投資の事後評価を巡る現況

ソフトウェア廃棄を締め括る、ソフトウェア**総括情報**の開示の意義等を述べる前に、少々迂回して、ＩＴ投資価値評価に関する調査研究を取り上げたい。ＪＵＡＳが「企業におけるＩＴ投資の利

活用が適正に行われるための環境調査事業」[491]として行なった調査を取り纏めたものである。経営管理的な視点で行なわれた調査であり、事前評価をも取り扱っているが、筆者の問題意識や本章の対象とする範囲とは異なるので、主題に沿って限定的に取り扱うことにする。

第１に、ＩＴ投資評価の実施状況であるが、「事後評価を実施していると回答した企業も、「実施している」「一部実施している」を合わせて 53%となっており、2006 年度初めて過半数を超えた。こちらも、ここ数年の推移を見ると、足踏み状態／やや後退した感もあったが、2006 年度は増加傾向に転じている」[492]、とのことである。売上高規模別に見ると[493]、大規模企業の方が実施率が高いという傾向にあり、「2005 年度は 10 億円未満の企業はすべての企業（10 社）が事後評価を実施していないと回答した。今回調査では 4 割の企業（5 社）が「実施している」「一部実施している」と回答している」[494]、とのことである。驚くことではないのかもしれないが、半数ほどの企業が事後評価を全く実施していないというのが、限られた調査なので一般化は慎むべきではあるが、企業の現況のようである。なお、事後評価をいつ実施するのかということだが、大凡の想定は「稼働後半年、あるいは 1 年後」[495]のようである。その理由は、「当プロジェクトの効果をもっと上げるためと、次のプロジェクトのためのノウハウ蓄積のために行われる」とのことであるが、そうした目的も結構ではあるが、ライフサイクルを通観するという視点はないようである（問題点は後述する）。

第２に、ＩＴ投資評価の実施基準であるが、「事後評価については、「一定金額以上の案件は実施」と回答した企業が半数を超えたが、「その他基準」と回答した企業が 26%であった」[496]、とのことである。「金額以外の基準として、利用ユーザー数の多い案件、年度重点投資上位 xx 位までの案件、年度計画案件、金銭的な効果／定量効果のある案件、基幹系など主要機能を支援する案件、などが上がっている」[497]ようである。「一定金額以上の案件は実施」という具体的な金額規模に関しては、「企業の売上規模により評価対象基準額が異なっている」が、大凡 1,000 万円を基準としている企業が多い（「売上高 1 兆円以上の企業（15 社）では、5,000 万円を事後評価する基準としている企業が最も多く」なっている）[498]、とのことである。なお、事前評価に関することだが、「大企業ほど対売上高比率でみて少額の案件まで評価対象としていることになり、ＩＴ投資に対する評価がよりき

---

[491] ＪＵＡＳ(2008)の副題に掲げている語句である。なお、同調査は、基本的に使用するデータはＪＵＡＳが毎年行なっている「企業ＩＴ動向調査」によるものなので(同書 p. 13)、それの近年のもので数値をアップデートすることは可能であるが、経年的変化の精細を確認することが目的ではなく、ここで確認しようとしている大凡の傾向が変わるほどのことではないので、その労は取らない。従って、数値等が若干古いものではあるが、支障はないと思う。

[492] ＪＵＡＳ(2008) p. 18。ちなみに、事前評価の同数値は 61%であった(同書 p. 18)。

[493] 売上高規模を、10 億円未満(n=13)／10 億～100 億円未満(n=157)／100～1000 億円未満(n=435)／1000 億～1 兆円未満(n=138)／1 兆円以上(n=31)、という 5 区分に分けている(カッコ内の n は回答企業数である)(同書 p. 19)。

[494] 同書 p. 18

[495] 同書 p. 26

[496] 同書 p. 20

[497] 同書 p. 20

[498] 同書 p. 21

め細かく実施されていると言える」[499]というコメントが加えられているが、事後評価に関しても同様であろう。

第３に、評価手法であるが、事前評価／事後評価とも「「システムのユーザーの満足度」が評価手法として圧倒的に多くの企業に利用されて」おり、「事後評価では「システムオーナーの満足度」、「システムのユーザーの満足度」、「顧客の満足度」がより多く利用されている」[500]、とのことである。同調査では、投資カテゴリーとして「インフラ型」／「業務効率型」／「戦略型」に分けており、各々の事後評価の評価手法としては、「インフラ型」では「評価を実施しているほぼすべての企業が「システムのユーザーの満足度」となって」おり、「業務効率型」では「「KPI」を確認している企業が 32%、「ROI」を確認している企業が 29%となっており」、「戦略型」では「「KPI」を確認している企業が 29%、「ROI」を確認している企業が 25%となっている」[501]、とのことである。なお、評価手法としては、ROI（投下資本利益率）、KPI（システム化対象業務上の指標）、システムオーナーの満足度、システムのユーザーの満足度、顧客の満足度、他社との比較・ベンチマーク、その他、が挙げられている[502]。但し、その各々の詳細内容は明らかにされていないので、適否・当否の判断は行なえない[503]。

第４に、事後評価が「全体としては半数近くの企業が実施していない」ということに関して、「理由」が分析されているが、①時間差、②負荷、③組織文化の問題を挙げている。①時間差ということでは、「システムの企画時とシステム利用時の時間差の影響で、環境がかなり変化している場合もあり、実績を出しにくい」[504]、としている。②負荷に関しては、「効果測定には負荷がかかるため、「後ろ向きなデータを集めるくらいなら新しいシステム開発をする方が企業の役に立つ」と、次の新規開発プロジェクトに走りたがる人が多い」[505]、としている。③組織文化の問題に関しては、「「上手く行っていないじゃないか。だからＩＴ部門はダメなんだ」と経営者がひと言、発した途端に、ＩＴ投資評価は停滞することになりかねない」[506]、としている。いずれも皮相なことが言われているだけである。なお、「システムライフサイクルコストは計画通りか」ということを取り上げ、「既

499 同書 p. 21
500 同書 p. 23
501 同書 p. 23
502 同書 p. 24
503 本論文はＩＴ投資評価に立ち入らないが、筆者は軽々しく ROI 等をまるで確かな定量的評価として挙示していることには常々疑問を持っている。例えば、収益の直接的源泉である「市場販売目的のソフトウェア」にしても、それを基に複製の商品・製品を販売することは営業活動ないし事業活動として総合的に行なうことであり、資産としてのソフトウェアだけが収益獲得に貢献しているわけではない。会計的に言えば、のれんも寄与しているのであり、それぞれの貢献を識別し測定し得なければ、ソフトウェアに関する ROI は精確には算出できないはずである。まして、収益の間接的源泉である「自社利用（社内利用）のソフトウェア」であれば、労働代替的な性格が極端に強いもの以外は費用削減等にどれほど貢献しているかの測定は容易なことではないはずである。
504 同書 p. 81
505 同書 p. 81
506 同書 p. 82

に運用段階に入っているので、システムライフサイクルコストは、高い精度で計算することが可能である」[507]と言っているが、「稼働後半年、あるいは１年後」の事後評価で、そのようなことが可能であるなどということは筆者には到底認め難いことである。視野が余りにも「近視眼的」である。最後に、１つ参考になることを挙げると、「利用頻度、効果測定機能をシステムの中に組み込まないと効果測定は難しい」[508]とし、測定機能の組み込みを薦めているが、内部統制や監査の機能をシステムに組み込むことによる実効性の担保と同様、機能を具体化し、組み込むことで負荷を軽減しつつ測定していくことは有意義なことであると思う。

以上で、ＪＵＡＳのＩＴ投資価値評価における事後評価を概観してきたが、事後評価の実施状況が芳しくないこと、評価内容が「満足度」といった漠然としたものが主たるもので、詳細な情報に基づく地道な評価ではないことが確認できた。その程度の評価でも行なわないよりはましかもしれないが、ソフトウェアに関することでありながら、堅牢で高い硬度のデータに基づく評価でなければ、今後に活かす確かな評価にはなり得ないと言うべきである。また、「システムライフサイクルコスト」などと言いながら、真にライフサイクルをトータルに捕捉するスタンスは全くない。そもそも事前評価／事後評価という設定自体が、開発の前／後という、開発を中心とした視点に囚われた（あるいは固執・固着した）ものであり、且つそのことに些かの疑念すら抱いていないのでは、ライフサイクルを本格的・全域的に捕捉することは難しいであろう。筆者の見据えていることとは、懸隔が甚だしいと言わざるを得ない。しかし、そうした現況を承知の上で、在りうべき情報開示を提言することにしたい。また、開示情報の評価は経営管理的には意義のあることであっても、その評価情報は開示情報としては代行的な予断であり、余計なものである。利害関係者が評価を行なえるような客観的な実態に即した開示情報とすべきである。

（2）ソフトウェア廃棄に伴う情報開示

後述の（3）で挙示するような10の纏まった情報の総称がソフトウェア**総括情報**であるが、説明の際に「測定」ということを言うが、言うまでもなく廃棄の際に各年度分を測定するわけではなく、ライフサイクルにおいてその都度測定し、記録をしておくのであり、廃棄に際しては廃棄に当たる最終年度分のみを別途記録してある資料から集計し記載するのである。従って、「管理簿」的なものがあれば、情報開示をするための作業は、比較的容易である。コストも対して掛からない。しかし、ソフトウェアは寿命が長いものであれば30年を超えるものもあり、10～20年のものは数多くある。同一の管理担当者が同じ１つのソフトウェアのライフサイクルを通じて担当者であり続けることはごく限られており、大半は交代すると言える。従って、組織的な取り組みをしていなければ、ライフサイクルを通した情報の捕捉はまず無理である。ソフトウェア**総括情報**を開示するには、これが第１の要諦となる。逆に言えば、これがクリアできれば、開示の作業負荷はそれほどないと言える。

[507] 同書 p. 86
[508] 同書 p. 87

但し、（1）で概観したように、「事後評価」の負荷が問題になっている現況では、この程度のことが負荷なのかもしれないが、このような取り組みなくしてソフトウェアを使いこなしていくことは覚束ないことであると言うべきである。

作業負荷は大したことはないけれども、このような情報を企業が果たして進んで開示するものであろうか。そもそも他の資産に関して、このような情報開示をすることはないであろう。それをソフトウェアだけ何故開示する必要があるのか。これが最大の問題であろう。端的に言えば、他の資産一般には、それほどのブラックボックスはないからである。特に広範囲に利用される汎用的な技術に関しては、形成・普及の途上では種々の問題が発生していたとしても、大方はクリアされ、問題は消滅しているのである[509]。それに対して、ソフトウェアは広範囲に普及し利用されているにも関わらず、今でも制作段階ないし運用段階で問題を出し続けているのである。一方では、労働代替手段として、あるいは業務を支援する手段として多大な便益を供しているが、他方では多大な問題含みの対象なのである。この途方もないギャップによく対処し得ていない、と筆者は考えている。しかも、社会インフラ、企業インフラという性格を有しながら、他のインフラと異なり、特定の限られた供給源からの供給ではなく、膨大な供給源が個々に形成され利用に供されているのである。そのような汎用的な技術は、ソフトウェア以外にはまずないと言える。また、「犯罪」といった限られた事象以外には、規制しようもないことであるし、規制することがよいとも言えない。自主的努力と、その情報開示による社会的な相互的関与こそが望ましい在り方であろう。

セキュリティに関しては、個人情報保護法を始めとする法的な根拠をもって、あるいは社会的な要請も強く、強化の方向にあり、それはそれとして更に取り組んでいけばよいが、筆者にはそればかりが着目される偏りがあるように思えてならない。あるいは、社会的な影響が大きいトラブルが起きると散発的にマスメディア等で報道され、話題にはなる。だがそれらはアドホックな関与でしかない。リスク管理が構造的に組み込まれ、位置付けられ、間断なく継続的に取り組まれなければならないことではないのか。そして、共時態的な対応としては、システム監査といったことを実効あるものにしていくことが必要である。しかし、特定時点の管理・監査だけでは限界がある。それに対して通時態的な対応として、ソフトウェア・ライフサイクルを通して適切な取り組みを行ない、その情報を開示することが、企業当事者にとっても、また多様な利害関係者にとっても必要であり、共時態的な対応と併せて、必要十分な対応になるのではないか。

これが、ソフトウェア**総括情報**の開示を必要とする理由である。そして、開示場所としては、**基礎情報**と同様に、内部統制報告書とすることが今のところよいであろう（将来的には独立的な報告書を作成することが望まれるとしても）。ソフトウェア廃棄に際しては、会計処理を行なうと共に、ソフトウェア**総括情報**を取り纏め、開示の段取りを整えることが、ライフサイクルを締め括ること

[509] だからこそ、例えば「安全性神話」が喧伝されてきた原発が、実際に事故を起こすと、大きな問題になるのである。そして、それまで特段に求められていなかった事象に関しても、環境問題を始めとして、種々の報告書による情報開示が求められるようになる、という事象・事態は少なくない。

であると提言したい。

なお、昨今、処分に係ることで、ＩＦＲＳ第５号「売却目的で保有する非流動資産及び廃止事業」に関する新たな議論が行なわれているが、同基準の表示・開示はほぼ金額的な情報に限定されており、ソフトウェア廃棄に際して非財務情報を含めたソフトウェア**総括情報**の開示という筆者の提言とは直接的に対象のカバレッジが合致するものではないが、交差することは少なくないので、国際的な動向としてその帰趨を注視していきたい。

また、分野が多少異なるが、ＣＳＲ（Corporate Social Responsibility：企業の社会的責任）の活動の一環としてＣＳＲ報告書が大企業を中心として開示されるようになってきている。ＣＳＲ報告書の準拠枠（の主要な１つ）である「ＧＲＩガイドライン G3」では、「網羅性」として「重要なテーマおよび指標の網羅および報告書のバウンダリー[510]の特定は、経済的、環境的および社会的な著しい影響を十分に反映し、ステークホルダーが報告組織の報告期間内のパフォーマンスを評価できるように行うべきである」[511]としており、「パフォーマンス指標」は「比較可能であり、経年の変化を示す、組織と関連する成果または結果に関する定性的または定量的情報のこと」[512]とされている。それを具体化したものとして、例えば食品業界ではライフサイクルアセスメント（Life Cycle Assessment:LCA、ISO14040 準拠）[513]、化学業界ではレスポンシブル・ケア（Responsible Care）活動[514]という、いずれも全ライフサイクルを対象とした取り組みを行ない、それをＣＳＲ報告書で報告するようになっているのである[515]。リスクの性格が異なるから、直ちに同様なことを行なうべきだということにはならないが、参考になる良い事例であると考える。

（3）ソフトウェア・ライフサイクルの総括情報

ソフトウェア・ライフサイクルの**総括情報**は、本論第３章で提示したソフトウェア**基礎情報**に対応するものである。**基礎情報**が開発終了時点の情報であるのに対して、**総括情報**は廃棄というライフサイクルの終了時点の情報である。大規模開発の場合には開発に数年を要するが、開発後の保守・運用は余程見込み違いで拡張性等に欠け、あるいは想定外の状況変化により、ごく短命に終わる場

510 「サステナビリティ報告のバウンダリーとは、組織のサステナビリティ報告書でパフォーマンスがカバーされている事業体の領域のことを指す」（ＧＲＩ(2006)p.39)。

511 ＧＲＩ(2006)p.12

512 ＧＲＩ(2006)p.39

513 ライフサイクルアセスメントとは、ISO 14040：2006(JIS Q 14040:2010)によれば、「製品システムのライフサイクルの全体を通したインプット，アウトプット及び潜在的な環境影響のまとめ，並びに評価」(3.2)のことである。

514 レスポンシブル・ケアとは、日本化学工業協会(2002)によれば、「化学物質を製造し、または取り扱う事業者が、自己決定・自己責任の原則に基づき、化学物質の開発から製造、流通、使用、最終消費を経て廃棄に至る全ライフサイクルにわたって、環境・安全・健康を確保することを経営方針において公約し、環境・安全・健康面の対策を実施し、改善を図っていく自主管理活動」(p.2)のことである。

515 新日本有限責任監査法人(2009)pp.148,151 等参照

合を除き、少なくとも開発期間の数倍の長さとなるものであり、様々な変遷を辿るのである。それらの経過を通時的に捉えた情報が**総括情報**である。**基礎情報**と対応する個々の**総括情報**を順次取り上げていくことにする。

図表 1-10-2　ソフトウェア総括情報（②機能の変遷）

＜①システム名＞

No.	サブシステム	機能	小機能	t_1年度	t_2年度	…	t_n年度
	Sub_1 ：	F_1 ：	f_1 ：				

（出典：筆者作成）

第１に挙げるのは、②機能（要件）の変遷である（①はシステム名なので、表題等に使用する。以下の図表では繰り返しになるので省略する）。大規模なソフトウェアを想定して、サブシステム＞機能＞小機能に分けたが、機能単位の程度に適合した分け方をすればよい。最初に挙示するのは、開発時に実装した機能である。それに対して、保守に入り廃棄に到るまでの各年度（t_1，t_2，…，t_n）において、機能の変更がなされた場合には、それを漏れなく記載するのである。開発時には実装していない機能の追加、実装していた機能の修正、実装していた機能の削除を各々記載する。同一年度で、同一機能に対して複数の変更を行なった場合には、複数行に分けて記載する。これらによって、生涯の機能の変遷を一望できることになる。機能の変更をコンパクトな語句で要約的に記載する。詳細情報としては、これに対応したプログラム（モジュール）レベルの精細な情報を一覧形式で作成する。

図表 1-10-3　ソフトウェア総括情報（③非機能要件の変遷）

No.	非機能要件	t_1年度	t_2年度	…	t_n年度
	$n-f_1$ ：				

（出典：筆者作成）

第２に挙げるのは、③非機能要件の変遷である。記載の仕方は、機能（要件）の場合と同様である。機能（要件）に比べれば、非機能要件の変更は遥かに少ないであろう。利用が拡大し、処理量・データ量が増大し、性能が低下したための性能向上とか、余りセキュリティのことが問題にならなかった時代に作られたソフトウェアのセキュリティ強化等、限られるかもしれない。しかし、その限られた変遷には、利用を巡る環境の変化等が如実に現れることになる。実現の技術的手段（方法）を含めたコンパクトな記載を行なう。詳細情報としては、これに対応したプログラム（モジュール）レベルの精細な情報を一覧形式で作成する。

図表 1-10-4　ソフトウェア総括情報（④アーキテクチャの変遷）

No.	アーキテクチャ	t_1年度	t_2年度	…	t_n年度
	Arc_1 ：				

（出典：筆者作成）

第３に挙げるのは、④アーキテクチャの変遷である。記載の仕方は、機能（要件）並びに非機能要件の場合と同様である。非機能要件と概ね同様、アーキテクチャの変更はそれほど多くないであろう。他のあるいは全体的な技術との親和性や移植性の関係で、部分的なアーキテクチャの変更はそれほど容易ではないからである。特定の機能の追加に関して、特定の新たなあるいは異質の技術要素で実現するといったことはしばしばあり得るが、機能横断的なものとなると、そうはいなかいのである。それでも、長期間の間には、基本ソフトないしミドル・ソフト、あるいはハードウェアの変更はサポート停止等の理由で起こり得るし、それに関連したアーキテクチャの変更はある。これらの変遷には、ソフトウェアを取り巻く環境の変化等が如実に現れることになる。実現の具体的な手段（方法）を含めたコンパクトな記載を行なう。詳細情報としては、これに対応したプログラム（モジュール）レベルの精細な情報を一覧形式で作成する。

なお、⑤開発方式、⑥開発技法、⑧プロジェクト管理は、更に変更は少なく、年度単位で時系列に追跡するほどではないので、もし生じた場合にはこの表に追記すればよいと考える。また、⑦開発ツールは、技術的な関連性も強いので、この表の項目に加えるのがよい。

図表１-10-5　ソフトウェア総括情報（⑩規模の推移）

No.	規模		初期開発	t_1年度	t_2年度	…	t_n年度
1	追加						
2-1	修正	対象規模					
2-2		実質増減					
3	削除						
4	増減計						
5	保守累計						
6	総合計						
7	総合累計						

（出典：筆者作成）

第４に挙げるのは、⑩規模の推移である。これまでは定性的情報を取り上げてきたが、これからは定量的情報を取り上げる（⑨開発期間という情報は、⑩以降の「初期開発」欄の参考情報となる）。まず初期開発終了時点の実績規模を記載する。この時点では、総合計と総合累計は同値である。また、初期開発ではまだ保守に関することは発生してない（それを灰色で表している、以下逐一断らないが、灰色の表示は同様の意味である）。それ以降の各年度で、保守作業が発生するが、保守として一括りにするのではなく、追加・修正・削除別に実績規模を測定し記載する。修正の場合は、修正対象の規模と実質的な増減を併記する。増減計は「追加＋修正の増減－削除」（実質的な増減）とする。保守累計は「追加＋修正＋削除」（修正作業の対象規模）とする。実質的な規模の増減と作業量の多寡は必ずしも相関的なわけではないので、作業量に対応する規模をも区別して捕捉する必要があるのである。総合計は「t_{k-1}の総合計＋t_kの増減計」とする（当該時点の全体規模）。総合累計は「t_{k-1}の総合累計＋t_kの保守累計」とする（当該時点までの開発・保守作業で対象とした累計規模）。総合計（全体規模）にそれほど増減がなくとも、保守の頻度が必ずしも少ないとは限ら

ないが、総合累計によって規模という観点からの保守の作業量を捕捉することが可能となる（直接的にわかるのは、工数においてであるが）。また、追加・修正・削除の各規模並びに増減計と、機能の変遷等と合わせてみることにより、保守の傾向性（追加による拡張性、制度改訂等による改訂高頻度、等）を察知することができる。いずれにせよ、規模という観点から、初期開発のあと、比較的変更が少なく定型的に利用されたのか、あるいは相当の変貌を遂げながら利用されてきたのか、といったことを捕捉することができる。

図表１-10-6　ソフトウェア総括情報（⑪工数の投入の推移）

No.	工数	初期開発	t_1年度	t_2年度	…	t_n年度
1	開発/保守					
2	開発/保守累計					
3	運用					
4	運用累計					
5	総合計					
6	総合累計					

（出典：筆者作成）

第５に挙げるのは、⑪工数の投入の推移である。まず初期開発終了時点の実績工数を記載する。この時点では、開発/保守と開発/保守累計、総合計と総合累計は同値である。それ以降の各年度で、保守並びに運用の作業が発生するので、各々の実績工数を測定し記載する。開発/保守累計は「t_{k-1}の開発/保守累計＋t_kの開発/保守」とする。運用累計は「t_{k-1}の運用累計＋t_kの運用」とする。総合計は「t_kの開発/保守＋t_kの運用」とする（当該年度の全体工数）。総合累計は「t_{k-1}の総合累計＋t_kの総合計」とする（当該時点までの開発/保守並びに運用の累計工数）。これらにより、各年度の保守と運用あるいはそれらの合計の工数並びに工数比率、開発/保守と運用あるいはそれらの合計の生涯工数並びに工数比率、といったことを捕捉することができる。

図表１-10-7　ソフトウェア総括情報（保守分類別の推移）

No.	保守	t_1年度			……	t_n年度		
		件数	規模	工数		件数	規模	工数
1	バグ対応							
2	維持							
3	改良							
4	部分的削除							
5	保守計							

（出典：筆者作成）

第６に挙げるのは、保守分類別の推移である。保守に関するやや立ち入った捕捉だが、初期開発終了時点の**基礎情報**には当然ない情報である。本論第９章「ソフトウェア保守」で提案した保守の分類に基づいて捕捉することになる。第４に挙げた規模の推移では、追加・修正・削除という形式面に着目した区分で捕捉したが、ここでは保守の内容面に着目している。件数は独自の項目である。規模に関しては、第４の規模の推移における保守累計と保守計は同値となる。工数に関しては、第５の工数の推移における開発/保守と保守計は同値となる。つまり、規模と工数の保守計は独自の項目というより、クロスチェック項目である。保守の分類別の推移により、内容的にどのような保守がどの程度行なわれたのか、判然とする。品質が悪くバグ対応が多かったのか、維持が多く軽微な

調整をする程度で利用され続けたのか、改良が多く進化し続けたのか、業務の改廃があり部分的削除が少なくなかったのか。あるいは、通史的な趨勢だけではなく、単年度ないし一定期間で異なる特徴が顕現したのか、こうしたことを捕捉することができるのである。

図表 1-10-8　ソフトウェア総括情報（⑫生産性の推移）

No.	生産性	開発計画	初期開発	t_1年度	t_2年度	…	t_n年度
1	開発/保守						
2	同上累計平均						

（出典：筆者作成）

第７に挙げるのは、⑫生産性の推移である。まず初期開発の開発計画時の目標と終了時点の実績の生産性を記載する。この各時点では、開発/保守と同上累計平均は同値である。それ以降の各年度で、保守の作業が発生するので、各々の実績生産性を測定し記載する。規模でウェイト付けした平均値とする。規模別（大規模/中規模/小規模等のランク別）、追加・修正・削除別、工程別（基本設計/詳細設計/製造（単体テスト含む）/結合テスト/総合テスト等の工程別）といった区分別の詳細な情報を採取できることが望ましいが、それに見合った管理粒度で管理している場合に限られるであろう。それよりも、開発計画で生産性の目標値を設定したり、それによって開発見積りをすることは広く行なわれているが、実績値を精密に測定し、目標値にフィードバックする管理メカニズムが機能していることは実際には少ない。あるいは、保守を継続する場合に、ＳＬＡの１つとして、生産性向上に関して具体的な目標値を徐々に高めていく（習熟により可能なはずである、但し保守要員のローテーションによる習熟の一時的低下等は考慮しなければならない）といった取り組みはそれほど構造化されているとは言えない。生涯の推移をみることで、あるいは後続のソフトウェアの目標値をみることで、このような組織的な取り組みがどのように行なわれているか、あるいは行なわれていないか、といったことを捕捉することができる。

図表 1-10-9　ソフトウェア総括情報（⑭品質の推移）

No.	品質	開発計画	初期開発	t_1年度				……	t_n年度			
				開発時作り込み件数	保守時作り込み件数	運用に起因する件数	年度計		開発時作り込み件数	保守時作り込み件数	運用に起因する件数	年度計
1	対外的に重大な障害											
2	社内的に重大な障害											
3	軽微な障害											
4	障害計											
5	障害累計											
6	欠陥除去											
7	欠陥除去累計											
8	欠陥除去率											
9	残存欠陥											
10	残存欠陥率											

（出典：筆者作成）

第８に挙げるのは、⑭品質の推移である。第６で挙げた保守分類におけるバグ対応よりも対象範囲が広く、運用マターの障害を含め、実際に顕現した障害を捕捉するものである。まず初期開発の開発計画時の目標と終了時点の実績の品質を記載する。障害の分類は、影響の軽重で、「対外的に重大な影響を及ぼす障害」と「社内的に重大な影響を及ぼす障害」と「軽微な影響に留まる障害」の３分類としたが、判定が明確にできるものであれば、異なった分類でも差し支えない。障害計と障害累計は、開発計画時並びに初期開発終了時点では各々同値である。障害累計は「t_{k-1}の障害累計＋t_kの障害計」とする。欠陥除去は、発生した障害に対して、除去した件数である。欠陥除去累計は「t_{k-1}の欠陥除去累計＋t_kの欠陥除去」とする。欠陥除去率は「t_kの欠陥除去累計÷t_kの障害累計」とする。各年度の欠陥除去率を算出してもよいが、その場合は当該年度で発生した障害と前年度以前からの未除去だった障害の除去を区別した方が各年度の特徴を捕捉しやすいであろう。残存欠陥は、「実際に発生した障害－欠陥除去」という捉え方もあるが、初期開発終了時に推計した「残存欠陥」を初期値として加算する。従って、「t_0の残存欠陥＋t_kの障害累計－t_kの欠陥除去累計」とする。残存欠陥率は「t_k残存欠陥÷t_k規模総合計」とする。「開発時作り込み件数」は障害原因が開発時のバグ等の件数であり、「保守時作り込み件数」は障害原因が保守時のバグ等の件数であり、「運用に起因する件数」はソフトウェアのバグ等ではなく、利用者の操作ミスを含め環境設定の不具合やハードウェア障害等運用次元の問題により障害が発生した件数であり、「年度計」はそれらの合計である。これらの数値により、どの程度の障害がどの程度の件数で発生し、それがどの程度除去され、なお潜在的な欠陥をどれほど抱えながら、運用され利用されてきたか、ということを具体的な数値で捕捉することができるのである。

図表１-10-10　ソフトウェア総括情報（⑮⑯⑰開発費等の推移）

No.	費用	初期開発	t_1年度	t_2年度	…	t_n年度
1	開発等直接費					
2	同上累計					
3	その他費用					
4	同上累計					
5	開発費等総額					
6	同上累計					
7	運用費					
8	同上累計					
9	総費用合計					
10	同上累計					

（出典：筆者作成）

第９に挙げるのは、⑮⑯⑰開発費等の推移である。まず初期開発の終了時点の開発費の実績金額を記載する。直接費とその他費用に分け、それらの合計を「開発費等総額」とする。「同上累計」は、「t_{k-1}同上累計（初期開発含む）＋t_kの各費用」とする。初期開発時は、各々の費用と「同上累計」は同値であり、運用費はまだ発生していない。「総費用合計」は「開発費等総額＋運用費」とする。これらにより、ＴＣＯを文字通り総合的に、且つ各年度の推移、開発/保守/運用の内訳あるいは配分を含めて捕捉することができるのである。次第に縮小傾向にあったのか、廃棄に到るまでコ

ンスタントに費用を投じて利用し続けたのか、時々に纏まった費用を投じて内外の何らかの変化に対応したのか、といった歴史をも金額ベースで如実に捕捉することができる。

図表１-10-11　ソフトウェア総括情報（⑬単価の推移）

No.	単価	初期開発	t_1年度	t_2年度	…	t_n年度
1	開発/保守					
2	同上累計平均					
3	運用					
4	同上累計平均					
5	総合累計平均					

（出典：筆者作成）

第10に挙げるのは、⑬単価の推移である。まず初期開発の終了時点の実績の単価を記載する。単価は、第９の費用と第５の工数から算出する。「同上累計平均」はt_{k-1}の同上累計平均とt_kの開発/保守又は運用を工数でウェイト付けした平均とする。「総合累計平均」は開発/保守と運用の「同上累計平均」を工数でウェイト付けした平均とする。これらによって、第９の合算費用に留まらず、単価ベースでコスト効率的に開発/保守/運用を生涯を通して行なってきたのか、あるいはそうした努力をしなかったのか、あるいは単価水準を高めとすることで、業務遂行の水準を確保し、それによって生産性又は品質を高水準とすることで、投じた費用に見合う成果を出してきたのか、といったことも捕捉することができるのである。

個々の各情報の説明を行なってきたが、それらを一覧的に纏めると、次ページのような表となる（斜字体は該当内容の数値を記載する）。②機能（要件）、③非機能要件、④アーキテクチャは対象とするソフトウェアによって可変的であるが、それ以外は固定的な項目（行）であり、46行となる。年度（列）はライフサイクルの年数によるので、10年であれば10列、20年であれば20列となる。保守分類別の推移と⑭品質の推移は、各年度が更に3又は4列の内訳で構成される。従って、それほど小さな表（行・列）ではないが、これによってライフサイクルを通時的に一望することができる。膨大なソフトウェアを運用し利用していても、各年度で廃棄するものは限られているから、それほどの量になるとは思えない。しかも、内部管理的な意義もあるが、利害関係者にとっては当該企業がソフトウェアをどのように取り扱ってきたのか、その全貌を推知することができる。この意義は大きい。どうしてもブラックボックスとなりがちなソフトウェアを、確かな定量的な情報に裏付けられ、鮮明にホワイトボックス化することになる。ソフトウェアのライフサイクル終了時点における重要なイベントとして、ソフトウェア**総括情報**の開示を是非とも強く提唱したい。

図表 1-10-12　ソフトウェア総括情報（一覧）

No.	項目	明細項目			t_1年度			……	t_n年度		
	機能	サブシステム	機能	小機能							
		Sub_1 :	F_1 :	f_1 :							
	非機能要件	$n-f_1$:									
	アーキテクチャ	Arc_1 :									
	規模	追加									
		修正	対象規模								
			実質増減								
		削除									
		増減計									
		保守累計									
		総合計		*初期開発分*							
		総合累計		*初期開発分*							
	工数	開発/保守		*初期開発分*							
		開発/保守累計		*初期開発分*							
		運用									
		運用累計									
		総合計		*初期開発分*							
		総合累計		*初期開発分*							
					件数	規模	工数	……	件数	規模	工数
	保守	バグ対応									
		維持									
		改良									
		部分的削除									
		保守計									
	生産性	開発/保守	*開発計画分*	*初期開発分*							
		同上累計平均	*開発計画分*	*初期開発分*							
					開発時作り込み件数	保守時作り込み件数	運用に起因する件数	年度計	……	開発時作り込み件数	保守時作り込み件数	運用に起因する件数	年度計
	品質	対外的に重大な障害											
		社内的に重大な障害											
		軽微な障害											
		障害計											
		障害累計											
		欠陥除去											
		欠陥除去累計											
		欠陥除去率											
		残存欠陥											
		残存欠陥率											
	開発費等	開発等直接費		*初期開発分*							
		同上累計		*初期開発分*							
		その他費用		*初期開発分*							
		同上累計		*初期開発分*							
		開発費等総額		*初期開発分*							
		同上累計		*初期開発分*							
		運用費									
		同上累計									
		総費用合計		*初期開発分*							
		同上累計		*初期開発分*							
	単価	開発/保守		*初期開発分*							
		同上累計平均		*初期開発分*							
		運用									
		同上累計平均									
		総合累計平均		*初期開発分*							

（出典：筆者作成）

結論

ソフトウェア基礎論　第１章　ソフトウェアの定義

ソフトウェア会計基準等におけるソフトウェアの定義では、「各種環境定義類」の規定が欠落しており、且つドキュメントの規定がわずかな例示に留まり、ドキュメントの対象範囲が不明確である。それに対して、筆者改訂案のソフトウェアの定義は、次の通りである。

「ソフトウェアとは、コンピュータを機能させるように指令を組み合わせて表現したプログラム並びに各種環境定義類の構造体、副次的なものとして企画、開発、保守、運用、廃棄の各種関連ドキュメントをいう」（下線部が改訂箇所）。

ソフトウェアの定義をこのように改訂することによって、ソフトウェアの的確且つ精緻な捕捉が可能となる。併せて、会計的にはソフトウェア資産の範囲が拡大する。具体的には、ソフトウェアの企画(そのうちのソフトウェアの購入や開発に結実する企画)や運用改善が資産となるのである。

ソフトウェア基礎論　第２章　ソフトウェア・ライフサイクル

ソフトウェア会計基準にはソフトウェア・ライフサイクルという全域的な視座がなく、ソフトウェアを開発中心に限られた範囲でしか取り扱っていない。ＩＳＯ等の各種標準でも、企画はほぼ独立的であるが、廃棄は運用又は保守の一部と位置付けられ、独立的に取扱われていない。それに対して、実態的には、ソフトウェア・ライフサイクルは、企画、開発、運用、保守、廃棄という５つの独立的な大フェーズで構成されているのである。

この明確化したライフサイクルという枠組みが、本論の構成となっており、ひいては筆者の構想するソフトウェア会計の体系構成となっているのである。

ソフトウェア基礎論　第３章　ソフトウェアの分類

ソフトウェアの分類は、会計処理を具体化するために必要なことであるが、会計基準における分類は不適切なものである。特に、自社利用におけるサービス提供は市場販売目的のソフトウェアと共に、収益の直接的源泉であるにも関わらず、別々に分類され、異なった会計処理を行なう規定となっている。それに対し、まず研究開発と研究開発以外に大別し、次に研究開発以外のソフトウェアは、第一分類規準を収益の源泉（直接的／間接的）、第二分類規準を制作目的、第三分類規準を取得形態とすることで、会計的に適切な分類となる。

このようにソフトウェアを分類することにより、自社利用におけるサービス提供と市場販売目的のソフトウェアの会計処理における齟齬を解消することが可能となる。そして、例えば今日のクラウド・コンピューティングにおける同一のソフトウェアを販売し且つサービス提供の資源(資産)とするような場合にも、整合的な会計処理が行えるようになる。

本論　第１章　ソフトウェア企画

ソフトウェア企画は、ソフトウェア会計基準では取り扱われていない。結果的に、会計慣行としては全て費用処理となっている。それに対し、ソフトウェアの定義におけるドキュメントの規定の明確化（企画ドキュメントもソフトウェアの定義に含めること）によって、異なる取り扱いが可能となる。企画ドキュメントが、ソフトウェアの開発ないし購入に結実する場合、開発時のドキュメントのように同時的ではなく、継時的ではあるが、ソフトウェアとセットになる成果物として、ソフトウェアと捉えることができる。これにより、企画の成果物であるドキュメントはソフトウェア資産となるのである。

なお、ソフトウェアの定義におけるドキュメント規定を主要な根拠とするので、ソフトウェアに直結しないソフトウェア戦略や中長期ないし年間計画に関するドキュメント等はソフトウェア資産とは見做さない。それらを資産化するには、別の論理構成を必要とするので、本論文では取り扱わないことにする。

本論　第２章　ソフトウェアの研究開発と非研究開発

ソフトウェア会計基準における「研究開発看做し」（研究開発目的のソフトウェアだけではなく、研究開発目的ではないソフトウェアにも「研究開発要素」が含まれているという看做し）を、会計基準の中核的な誤謬（セントラル・ドグマ）と、筆者は捉えている。これによって、資産範囲を不当に狭め、また「著しい改良」と「著しくない改良」という不当な区別を派生させている（著しい改良のほうがより資産性が増すにも関わらず、費用処理という会計処理を強いている）。

そのような看做し規定はアメリカ基準に淵源するが、それを打破することが適正なソフトウェアに係る会計基準としていくための中枢的な取り組みである。その意味で、本論第２章は、本論全体における最も重要な章と言えるものである。しかも、看做し規定に疑義を唱える研究者等は多少いたが、適切な論理構成によって整然と論破し得た者は管見の限り誰もいなかったのである。

それに対し、経営学におけるイノベーションのライフサイクル研究を援用し、ソフトウェアのプロダクトの位相で、ライフサイクル座標（創生期、普及初期、普及拡大期、コモディティ期、レガシー期）を設定して、それにより研究開発と非研究開発を的確に測位可能であることを証示した。具体的には、ソフトウェアの機能又は技術のいずれかが創生期のものである場合が研究開発であり、それ以外は研究開発ではないのである。もう１つの側面であるプロセスに関しても、研究開発と非研究開発が形態的並びに定量的に明確に相違していることを示した。具体的には、プロダクトに比べて、新奇のプロセスは圧倒的に少ないのである（製法の研究開発が少ないということである）。また、試行錯誤のため、同じアクティビティ等の反復が構造的に組み込まれているのが研究開発であり、そうでなければ研究開発ではないのである。

これらによって、研究開発目的ではない通常の一般的なソフトウェア開発には、「研究開発要素」など含まれていないことが明白となった。従って、ソフトウェアにおける研究開発と非研究開発は

峻別しなければならないし、またそれが可能になるのである。それにより、各々に相応しい会計処理を行なうことが適正であることを提言した。研究開発と非研究開発を混濁させたソフトウェア会計基準の中核的な誤謬（セントラル・ドグマ）を周到な論証により打破し得た意義は実に大きいと言わなければならない。

本論　第３章　ソフトウェア開発

ソフトウェア開発を、特に機能と技術の立体的構造化として捕捉し、その上でソフトウェア資産の測定として、ソフトウェアの特性に即したソフトウェア測定の方法並びに項目を使って原価計算を行なうことを提案した。更に、より具体化するために、仮想的な事例を設定して、計算を行い、計算項目の説明を詳細に行った。また、ソフトウェア測定の項目である、規模（ＬＯＣ又はＦＰ）、工数、生産性、単価、開発費は、ソフトウェアの基礎情報と言えるものであり、これらによって開発の成否等の内実を精緻に捕捉可能とするのである。

それにより、企業内並びに企業間のソフトウェアの比較評価を客観的且つ定量的に行なうことが可能となる。開発費総額の金額的多寡が一意的にソフトウェアの規模や有用性を示すものではないのであり、現行の会計処理や情報開示では捕捉できないことである。

本論　第４章　ソフトウェア再利用

ソフトウェアの再利用に関しては、ソフトウェア会計基準における取り扱いには幾つか問題点があり、それらを剔抉している。1 つに費用あるいは資産に振り分けているだけであり、2 つに想定されている事態が相当稀ないし局部的な事態を一般化しており、3 つに自社利用のみの再利用が想定されていることである。

それに対し、再利用に適合的な会計処理を、ソフトウェアの分類に基づき、ケースを設定して具体化した。外部購入ソフトウェアの再利用、自社開発ソフトウェアの再利用、ソフトウェア再構築に大別し、ａ～ｋの全 11 ケースに関して、具体的に数値事例を挙示した。そして、費用負担の仕方や振替処理等の実態に適合した会計処理を提示したのである。

本論　第５章　オープンソース利用

オープンソースは、利用が拡大しているが、現行の会計処理では、一般的に無償取得であるためオフバランスになっている。しかし、オンバランス化すべきであり、そうでなければ財務諸表ではオープンソースの利用が全く捕捉不能のままであるので、それを可能とする会計制度的な根拠を探索した。企業会計原則における無償取得に係る規定並びにＩＡＳＢの「財務報告に関する概念フレームワーク」における資産概念である。

それらを根拠とし、具体的に「公正な評価額＝取得原価」を算定する算定モデルを考案した。まずオープンソース評価モデルで、オープンソースと「類似の商用ソフトウェア」等を評価し、点数

化する。「類似の商用ソフトウェア」がある場合は、その価格と各点数により、オープンソースの評価額を算定する。「類似の商用ソフトウェア」がない場合は「自社開発ソフトウェア」を想定し、開発見積りを行い、その開発費と評価点により、オープンソースの評価額を算定するのである。各々に関して、算定方法を仮想的な事例に沿って具体的に提示し、それをオンバランス化する場合の資産計上額とすることを提言した。

本論　第６章　ソフトウェア仕損

ソフトウェア開発プロジェクトの失敗は、10 年等のスパンでも顕著に減少していないが、そうした開発の失敗が会計処理には何ら反映されていない。むしろ、失敗により、例えば開発費が増額した場合、そのまま資産計上され、増嵩された資産となってしまう。開発の失敗は、ソフトウェアの仕損と捉えるべきではないのか。しかし、かつて櫻井通晴も同じ問題意識を有しながら、仕損処理を設定することは断念した経緯がある。筆者はそれを、櫻井が当時依拠したプロジェクト管理の粗い粒度（フェーズ・ベース）並びにソフトウェア測定の未成熟に基因していると総括した。

それに対し、開発プロセスを精細な粒度（タスク・ベース）で捉え、開発の成否をＱＣＤに関するソフトウェア測定の該当項目で定量的に捉えることで、ソフトウェアの仕損を捕捉可能とした。

その上で、減損会計の処理に類似した形で、ソフトウェアの仕損に係る会計処理を、具体的に、定義並びに適用対象、仕損の兆候の識別、仕損の認識、仕損の測定という順序で提示した。これらによって、ソフトウェア開発プロジェクトの失敗を、会計的に具体的に捕捉可能としたのである。

本論　第７章　システム運用改善

システム運用に係る日本並びにアメリカの会計基準は、いずれも運用全般を取り扱ってはおらず、ごく限定的であり、あるいは運用の範疇ではないものを運用と看做すなど、不適切である。しかも、運用改善を取り扱っていない。しかし、開発の延長である保守において改良があるように、ソフトウェアのうちプログラムの改変を伴わない、運用における改善も実際にあるのである。

それに対し、システム運用改善を正規に取り上げ、しかもソフトウェアの定義の改訂案に基づき（具体的には、「各種環境定義類」をソフトウェアの要素に含めること）、「各種環境定義類」を改良することによるシステム運用改善という事象を画定し、それに適合的な会計処理として資産計上を、運用改善の例示を含め提案したのである。プログラムの改良が資産となるのと、「各種環境定義類」の改良が資産となることは相同であるとするのは十分に合理的なことである。これによって、運用は会計慣行では全て費用処理になっていたが、運用改善に関しては資産計上が可能になるのである。

本論　第８章　クラウド・コンピューティング

クラウド・コンピューティングは、大別すると、パブリック・クラウドとプライベート・クラウドとそれらの複合形態であるハイブリッド・クラウドがある。今日、次第に利用が拡大してきてお

り、更に拡大する可能性がある。ソフトウェア会計基準は、その前身ないし一形態であるＡＳＰが出現した時期に設定されたので、クラウド・コンピューティングを取り扱っていない。それは時期的に致し方ないことであるが、その後の改訂でも一顧だにされていない。しかし、普及拡大により、現に問題が発生しているのである。

他の章とは異なり、調査方法として、筆者自らアンケート調査を、事業者側並びに利用企業側に対して行った。理由は、市販の書籍等だけでは、必要な情報が得られなかったからである。また、ＪＩＳＡ（情報サービス産業協会）の行った論点整理を参考にしたが、惜しむらくはソフトウェア会計基準の改訂にまでは踏み込まず、現行基準の枠内での整理に留まったからである。

それに対し、事業者側の会計処理の問題点としては、「事業者側におけるソフトウェアの制作目的別分類の適合性」（現行会計基準のソフトウェア分類には具体的に齟齬が生じていること）を明確にした。具体的には、同一のソフトウェアを販売し、且つパブリック・クラウドでサービス提供している場合に、会計基準のソフトウェアの分類の市場販売目的のソフトウェアと、自社利用のサービス提供のいずれかとせざるを得ず、それによって会計処理が大きく異なってしまうことである。これに関しては、筆者が提案しているソフトウェアの分類による以外に解決の方途はないのである。

利用企業側の会計処理の問題点としては、パブリック・クラウドの場合には純然たる利用として費用処理でよいが、プライベート・クラウドの場合には「利用企業側におけるプライベート・クラウドの資産性」（費用処理ではなく資産計上することが実態適合的であること）を考慮することが妥当であると判断した。調査でも大凡判明したように、利用するソフトウェアのライセンス契約は権利元と利用企業（事業者ではなく）が締結するのであるから、利用企業はライセンス契約の当事者であり、パブリック・クラウドの利用者とは明らかに性格が異なるのである。また、期間的に比較的長いこと、解約不能な契約があること、事業者側に起因する障害・事故発生に対する損害賠償の範囲が狭いことなどからも、専有的であり、資産計上することの妥当性が高いと言える。

本論　第９章　ソフトウェア保守

ソフトウェアの保守を多角的に捉え（統計、保守の分類、ＪＩＳ規格の規定、ファンクションポイント法における捕捉方法）、会計における保守の取り扱いとその問題点を日本並びにアメリカの会計基準に沿って明確にした。更に、問題点をより具体的に剔抉するために、幾つかの保守事例を設定し、どのような会計処理となるかを各々確認した。

それらを通じて明らかとなった問題点に対し、保守に適合的な会計処理とは如何なるものかを提示した。保守に適合的な分類は、バグ対応、維持（ポジティブな維持）、改良、部分的な削除である。そして、各々に対する会計処理としては、費用処理、資産計上、資産計上、除却処理（資産の削減）である。維持（ポジティブな維持）を資産計上することには異議があり得るかもしれないが、物理的な経年劣化のないソフトウェアに関しては、維持が資産の寿命の延長に繋がることであり、資本的支出であると解することが妥当である。

本論　第10章　ソフトウェア廃棄

ソフトウェア廃棄に係る会計処理に関しては、現行会計基準並びに櫻井通晴の会計処理案を点検した上で、廃棄に適合的な会計処理を提示した。未償却残高がある場合と償却済の場合、いずれに関しても、「除却費」として費用処理を行なうことが適合的である。未償却残高がある場合は、その金額と廃棄に係るそれ以外の費用(廃棄計画並びに実施に関する人件費あるいは外部委託費その他、廃棄に関する企画業務を行なったのであれば、それを含む）を含める。償却済であれば、廃棄に係る費用が該当する。但し、「除却費」と言っても、特別損失ではなく、営業外費用とするのが妥当である。

更に、ソフトウェア・ライフサイクルを締め括るに際して、それを総括し、総括情報を開示すべきことを提言した。企画に始まり、開発し（あるいは購入し)、一定期間運用並びに保守を続けてきたソフトウェアが、どのような変遷を辿ってきたのか、それらを総括し、情報を開示することは、内部関係者に留まらず、広く利害関係者に有用であるからである。

個々の章に即した結論は上記の通りであるが、最後に本論文全体を通した結論を述べることにする。ソフトウェアに係る会計処理の在り方としては、そのライフサイクル全域を対象としたものでなければならない。ところが、現行のソフトウェア会計基準は開発並びに保守等を部分的に対象としたものに留まっており、且つ基準設定以降のソフトウェア分野の動向に対応し得ていない。それに対して、本論文はライフサイクル全域を対象とし、近年のソフトウェア分野の動向までを視野に入れて、体系的な研究を行なったものである。現行基準に比べ、資産範囲が拡大し、開示情報も大幅に増加する会計処理を提言したが、それによりソフトウェアへの投資並びに利用が益々拡大・高度化してきている経済的実態により適合し、利害関係者への目的適合性を増すことになると考えている。そのような提言に到る過程では、ソフトウェアの市場動向・技術動向、企業の取り組み状況（会計実務を含む)、会計における概念フレームワークや無形資産の理論的研究を調査・検討し、それらを踏まえ、整合する考察を繰り返し行なうことにした。それ故、提言は十分な裏付けを持ったものであると考えている。そして、ソフトウェア会計の代表的な研究者である櫻井通晴教授の研究を継承し、更に前進させたものと考えている。

なお、ソフトウェア会計研究が全般的に低調であり、先行研究も少ないことから、反対意見あるいは異見との議論を通じて考察を深め、あるいは洗練させることが困難であることから、筆者の研究に多少の偏りがあるかもしれない。本論文の限界と言える。これに関しては、より幅広い、奥行きのある研究にしていくことは今後の課題として残される。また、ＩＡＳＢの概念フレームワークに関しては現段階では公開草案であり、未だ流動的であるが、それが確定した段階で、再度資産概念等の突合を行ない、会計理論的により整合的な内容に深めていくことは今後の課題である。更に、今後のソフトウェア分野の動向次第では、新たな会計処理の検討あるいは改訂の必要はしばしば生

じ得るので、それらに対しては継続的に対応していきたい。その意味では、本論文は現時点での体系的集成であり、意義は小さくないと考えるが、引き続き行なっていく研究の一里塚といった位置付けが相応であると考えている。

参考文献

・合崎堅二、能勢信子共編(1971)『企業会計と社会会計』森山書店

・秋葉賢一(2011)『エッセンシャルＩＦＲＳ』中央経済社

・あずさ監査法人編(2010)『内部統制報告書の記載事例分析』別冊商事法務 No. 338 商事法務

・あずさ監査法人ＩＴ監査部編著(2010a)『ＩＴ内部統制ケースブック　最新 50 の不備対応事例に学ぶ』東洋経済新報社

・東季彦監修(1996)『全訂二版　著作権法』学陽書房

・アールワークス編著(2012)『クラウド時代の正しいシステム運用　オープンソースを活用した次世代運用技術の実際』インプレスビジネスメディア

・アールワークス編著(2012)『クラウド時代の正しいシステム運用　オープンソースを活用した次世代運用技術の実際』インプレスビジネスメディア

・池田公司(2009)『知的資産の監査』　中央経済社

・井手吉成佳(2010)「ソフトウェア原価計算における機能的規模測定法の適用可能性」広島大学大学院、博士論文

・井上哲也(1997)「情報化関連産業の成長とその捕捉における問題について」『金融研究』第 16 巻第 4 号 P. 55-82、日本銀行金融研究所

・インプレスＲ＆Ｄ　インターネットメディア総合研究所編(2012)『インターネット白書 2012』インターネット協会監修、インプレスジャパン

・上原三八、Wei-Tek Tsai、佐野隆、馬場一弥(2000)『保守とリエンジニアリング』共立出版

・植松宏嘉(2000)『改訂新版　コンピュータプログラム著作権Ｑ＆Ａ』金融財政事情研究会

・鵜飼康東編著(2003)『銀行業情報システム投資の経済分析』多賀出版

・浦川卓也 (2010)『実践研究開発マネジメント』日刊工業新聞社

・江口純一(2013)『ソフトウェアの信頼性向上に向けた取組』コンピュータソフトウェア協会

・太田昭和監査法人、ビジネス・ブレイン太田昭和編(1992)『ソフトウェア開発の原価管理　改訂版』中央経済社(1987 年初版)

・大森徹(1998)「国民経済計算におけるコンピュータ・ソフトウェアの取り扱いに関する概念的整理」IMES Discussion Paper Series 98-J-30、日本銀行金融研究所

・大和田尚孝(2009)『システム統合の「正攻法」　世界最大のプロジェクト三菱東京ＵＦＪ銀行「Ｄａｙ２」に学ぶ』日経ＢＰ社

・岡田賢治(2011)『はじめての Java フレームワーク　第３版　Struts 2/Spring/Hibernate 対応』秀和システム

・岡本彬良(2005)『よくわかるプリント基板回路のできるまで―基板設計、解析、ＣＡＤからＤＦＭまで―』日刊工業新聞社

・奥本佳伸(2012)「新しい国民経済計算体系 2008SNA について」『千葉大学　経済研究』第 26 巻第 4 号 P. 125-138、千葉大学
・片山尚平(2006)『投資，成長と経済制作』晃洋書房
・加戸守行(1979)『三訂　著作権法逐条講義』著作権資料協会
・加戸守行(2013)『著作権法逐条講義　六訂新版』著作権情報センター
・加藤久明(2007)『現代リース会計論』中央経済社
・金井重彦、小倉秀夫編著(2000)『著作権法コンメンタール(上巻)』東京布井出版
・企業会計基準委員会(2006)「実務対応報告第 17 号　ソフトウェア取引の収益の会計処理に関する実務上の取扱い」
・企業会計基準委員会(2007a)「企業会計基準第 13 号　リース取引に関する会計基準」
・企業会計基準委員会(2007b)「企業会計基準適用指針第 16 号　リース取引に関する会計基準の適用指針」
・企業会計基準委員会(2010)「第 197 回企業会計基準委員会(2010 年 3 月 11 日)審議事項(５)－３」
・企業会計基準委員会(2012a)「第 242 回企業会計基準委員会(2012 年 4 月 19 日)審議事項(４)」
・企業会計基準委員会(2012b)「第 251 回企業会計基準委員会(2012 年 9 月 5 日)審議事項(２)」
・企業会計基準委員会(2013) ASBJ の意見募集 平成 25 年 8 月 12 日「IASB ディスカッション・ペーパー「財務報告に関する概念フレームワークの見直し」に関する意見の募集」
・企業会計基準委員会(2013)「IASB ディスカッション・ペーパー「財務報告に関する概念フレームワークの見直し」の和訳」
・企業会計審議会(1960)「企業会計原則と関係諸法令との調整に関する連続意見書」
・企業会計審議会(1962)「原価計算基準」
・企業会計審議会(1998)「研究開発費等に係る会計基準」
・企業会計審議会(1997)「研究開発費等に係る会計基準の設定に関する意見書〈公開草案〉」
・企業会計審議会(1998)「研究開発費等に係る会計基準の設定に関する意見書」
・企業会計制度対策調査会(1982)「企業会計原則」「企業会計原則注解」
・木村俊孝、木村早霧(2012)「自社開発ソフトウェアの推計方法について」『季刊国民経済計算』平成 24 年度第 2 号(№148)P. 109-114、内閣府経済社会総合研究所国民経済計算部
・共通フレーム検討委員会（SLCP-JCF98 委員会）編(1998)『共通フレーム 98―SLCP-JCF98―(1998 年版)　ソフトウェアを中心としたシステム開発および取引のための共通フレーム』通産資料調査会
・共通フレーム検討委員会編(1994)『システム開発取引の共通フレーム(1994 年版)―SLCP-JCF94―』通産資料調査会
・金融情報システムセンター(2014)『金融機関等のシステム監査指針　改訂第３版』金融情報システムセンター
・金融庁(2012)『金融検査マニュアル(預金等受入金融機関に係る検査マニュアル)』平成 24 年 6

月、金融庁
・金融庁(2012)「主要行等向けの総合的な監督指針」金融庁
・銀行システム研究会(2013)『銀行システム入門　銀行経営者とＣＩＯが知っておくべき銀行システムの基礎知識』キャリア教育出版
・熊坂有三・峰滝和典(2001)『ＩＴエコノミー　情報技術革新はアメリカ経済をどう変えたか』日本評論社
・倉西誠一編(2012)『成功するクラウド選び　2012 年度版』アスキー・メディアワークス
・倉林義正(1989)『SNA の成立と発展』岩波書店
・経済企画庁経済研究所(2000a)『93SNA 推計手法解説書（暫定版）』経済企画庁経済研究所
・経済企画庁経済研究所(2000b)『我が国の 93SNA への移行について（暫定版）』経済企画庁経済研究所
・経済産業省(2007)『システム管理基準　追補版（財務報告に係る IT 統制ガイダンス）』経済産業省
・経済産業省(2012)『平成２３年情報処理実態調査結果報告書』経済産業省
・経済産業省商務情報政策局(2005)『新版　システム監査基準／システム管理基準解説書』平成 16 年基準策定版　日本情報開発協会
・経済産業省商務情報政策局情報処理振興課(2010)「産業競争力を担う組込み技術の今後の展開について」経済産業省商務情報政策局情報処理振興課
・経済産業省ソフトウェアメトリクス高度化プロジェクト プロセスメトリクス WG(2011)『IT プロジェクトのベンチマーク供給者のためのガイドライン～組織内用，公開用ベンチマークの供給～』経済産業省
・古賀智敏(2012)『知的資産の会計　マネジメントと測定・開示　改訂増補版』千倉書房
・国民経済計算次回基準改定に関する研究会(2014)「研究開発（R&D）の資本化等」第７回資料１－１、国民経済計算次回基準改定に関する研究会
・古庄修(2012)『統合財務報告制度の形成』中央経済社
・作間逸雄編(2003)『SNA がわかる経済統計学』有斐閣
・櫻井通晴、松本文一郎、佐藤光司、東野敏雄(1992)『ソフトウェア原価計算〔増訂版〕』白桃書房(1987 年初版)
・櫻井通晴編著(1993)『ソフトウェア会計■ソフトウェア会計実務指針〔案〕の解説と実際例』中央経済社
・櫻井通晴(1999)「ソフトウェア会計の基準化は何をもたらすのか――日本企業に及ぼすインパクト」『経理情報』1999 年 7 月 1 日号
・櫻井通晴(2006)『ソフトウェア管理会計―ＩＴ戦略マネジメントの構築―　第２版』白桃書房(2001 年初版)

・櫻井通晴(2012)「ＩＦＲＳがソフトウェア開発費の会計処理に及ぼす影響――ソフトウェア開発費の理論的・実務的検討」『企業会計』Vol.64 No.８ 、中央経済社
・桜井久勝(2013)『財務会計講義　第14版』中央経済社
・桜井久勝(2014)『財務会計講義　第15版』中央経済社
・櫻本健(2008)「2008SNAの特徴と日本へ導入する際の注意点」『統計学』第95号P.32-35、経済統計学会
・佐藤信彦・角ヶ谷典幸編著(2009)『リース会計基準の論理』税務経理協会
・茂野正史(2012)「我が国の国民経済計算におけるＲ＆Ｄ資本化の導入に向けて」『季刊国民経済計算』№149P.83-99、メディアランド
・証券取引等監視委員会(2013)『金融商品取引業者等検査マニュアル』証券取引等監視委員会事務局
・情報処理学会編(1983)『ＪＩＳ情報処理用語解説』朝倉書店
・情報サービス産業協会（ＪＩＳＡ）(2003)『情報サービス産業における経理規程モデル』ＪＩＳＡ
・情報サービス産業協会（ＪＩＳＡ）(2010a)『近時の情報サービス業界の会計動向に関する調査報告』ＪＩＳＡ
・情報サービス産業協会（ＪＩＳＡ）(2010b)『平成22年度　情報サービス産業　取引及び価格に関する調査』ＪＩＳＡ
・情報処理推進機構（ＩＰＡ）(2006a)『ＩＴプロジェクトの「見える化」上流工程編』ＩＰＡ
・情報処理推進機構（ＩＰＡ）(2006b)『ＩＴプロジェクトの「見える化」中流工程編』ＩＰＡ
・情報処理推進機構（ＩＰＡ）(2006c)『ＩＴプロジェクトの「見える化」下流工程編』ＩＰＡ
・情報処理推進機構（ＩＰＡ）(2007)『共通フレーム2007　～経営者、業務部門が参画するシステム開発および取引のために～』ＩＰＡ
・情報処理推進機構（ＩＰＡ）(2008)『第29回　情報処理産業経営実態調査報告書(2007年度調査実施)』ＩＰＡ
・情報処理推進機構（ＩＰＡ）(2009)『共通フレーム2007　第2版　～経営者、業務部門が参画するシステム開発および取引のために～』ＩＰＡ
・情報処理推進機構（ＩＰＡ）(2013)『共通フレーム2013　～経営者、業務部門とともに取り組む「使える」システムの実現～』ＩＰＡ
・情報処理推進機構（ＩＰＡ）ソフトウェア・エンジニアリング・センター編(2006)『ソフトウェア開発見積りガイドブック　ＩＴユーザとベンダにおける定量的見積りの実現』オーム社
・情報処理推進機構（ＩＰＡ）オープンソースソフトウェア・センター編(2007)『ＩＴシステム導入虎の巻　オープンソースで構築！』オーム社
・情報処理推進機構（ＩＰＡ）ソフトウェア・エンジニアリング・センター編(2007)『ソフトウェ

ア改良開発見積りガイドブック～既存システムがある場合の開発～』オーム社
・実積寿也(2005)『ＩＴ投資効果メカニズムの経済分析　ＩＴ活用戦略とＩＴ化支援政策』九州大学出版会
・篠崎敏彦(2003)『情報技術革新の経済効果――日米経済の明暗と逆転』日本評論社
・新日本有限責任監査法人(2009)『ＣＳＲ報告書の読み方・作り方』中央経済社
・総務省統計局(1990)『産業連関表―総合解説編―』総務省統計局(リプリント)
・総務省統計局(1995)『産業連関表―総合解説編―』総務省統計局(リプリント)
・総務省統計局(2000)『産業連関表―総合解説編―』総務省統計局(リプリント)
・総務省統計局(2005)『産業連関表―総合解説編―』総務省統計局
・ソフトウェア・メインテナンス研究会編、増井和也・弘中茂樹・馬場辰男・松永真著(2007)『ソフトウェア保守開発～ISO14764 による～』ソフト・リサーチ・センター
・武田隆二(2008)『会計学一般教程　第７版』中央経済社
・武野秀樹(2001)『国民経済計算入門』有斐閣
・竹村敏彦(2008)『情報通信技術の経済分析』多賀出版
・通商産業省編(1992)『産業科学技術の動向と課題　地球規模での技術的共生に向けて』通商産業調査会
・天明茂(1996)「汎用ソフトウェアの会計処理に関する一考察――製品マスターの資産性と棚卸資産表示の妥当性吟味」『企業会計』Vol.48 No.５ 、中央経済社
・電子情報技術産業協会（ＪＥＩＴＡ）(2012)『民間向けＩＴシステムのＳＬＡガイドライン　第四版』日経ＢＰ社
・特定サービス産業動態統計調査：(アクセス 2014/04/25,12:25)
(http://www.meti.go.jp/statistics/tyo/tokusabido/gaiyo.html)
・富永新(2009)『我が国金融機関への期待　ＩＴリスク管理と事業継続の未来を拓く』生産性出版
・内閣府経済社会総合研究所(2007)『SNA 推計手法解説書(平成 19 年改訂版)』内閣府経済社会総合研究所
・内閣府経済社会総合研究所(2011a)『国民経済計算の作成方法』内閣府経済社会総合研究所
・内閣府経済社会総合研究所(2011b)「「平成 16 年度国民経済計算確報及び平成 12 年基準改定結果」利用上の注意」内閣府経済社会総合研究所
・内閣府経済社会総合研究所(2011c)「「平成 22 年度国民経済計算確報（平成 17 年基準改定値」に係る利用上の注意について」内閣府経済社会総合研究所
・内閣府経済社会総合研究所国民経済計算部(2009)「第２回国民経済計算部会ストック専門委員会資料２－２　自社開発ソフトウェア推計方法について」内閣府経済社会総合研究所国民経済計算部
・内閣府経済社会総合研究所国民経済計算部(2010)「第９回国民経済計算部会ストック専門委員会資料２　自社開発ソフトウェアについて」内閣府経済社会総合研究所国民経済計算部

・内閣府経済社会総合研究所国民経済計算部(2011)「国民経済計算における平成 17 年基準改定の概要」内閣府経済社会総合研究所国民経済計算部
・内閣府経済社会総合研究所国民経済計算部編(2013a)『平成 25 年版　国民経済計算年報』メディアランド
・内閣府経済社会総合研究所国民経済計算部(2013b)「2008SNA について」資料３、内閣府経済社会総合研究所国民経済計算部
・内部統制監査研究会編(2012)『内部統制・内部統制監査の研究』商事法務
・永井昭弘(2005)『ＲＦＰ＆提案書 完全マニュアル』日経ＢＰ社
・永井昭弘(2011)『事例で学ぶＲＦＰ作成術 実践マニュアル』日経ＢＰ社
・中村恒彦(2003)「アメリカにおけるソフトウェア会計の経路依存」『桃山学院大学経済経営論集』第 45 巻第 2 号、桃山学院大学総合研究所
・中村直人(2011)『判例に見る会社法の内部統制の水準』商事法務
・中村洋一(2010)『新しい SNA　2008SNA の導入に向けて』日本統計協会
・西澤脩(1991)「米国におけるソフトウェア費の会計処理基準」『早稲田商学』通号 349
・日経コンピュータ(2008)「第２回システム開発プロジェクトの実態調査(2008 年実施)」『日経コンピュータ』2008 年 12 月 01 日号、日経ＢＰ社
・日経コンピュータ(2012)「みずほの次期システムはマルチベンダー発注先決定、預金系を除きオープン化」『日経コンピュータ』2012 年 11 月 22 日号、日経ＢＰ社
・日経コンピュータ(2013)「みずほ、ＩＴ統合はこれからが正念場　総額 3000 億円、前例なき全面刷新に挑む」『日経コンピュータ』2013 年７月 11 日号、日経ＢＰ社
・日経コンピュータ編(2011)『システム障害はなぜ二度起きたか　みずほ､12 年の教訓』日経ＢＰ社
・日経コンピュータ他編(2012)『すべてわかる仮想化大全 2013　クラウドを強化する製品・技術』日経ＢＰ社
・日経ＢＰシステム運用ナレッジ編著(2013)『システム運用実態調査 2013 報告書』日経ＢＰ社
・日経ＢＰ社出版局編(2010)『クラウド大全　第２版――サービス詳細から基盤技術まで』日経ＢＰ社
・日本化学工業協会(2002)「レスポンシブル・ケア　コード」ＪＲＣＣ－ＲＬ－２、日本化学工業協会レスポンシブル・ケア委員会
・日本規格協会（ＪＳＡ）(1987)『情報処理用語（基本用語)』JIS X 0001:1987　日本規格協会
・日本規格協会（ＪＳＡ）(1994)『情報処理用語－基本用語』JIS X 0001:1994　日本規格協会
・日本規格協会(ＪＳＡ)(2010)「ライフサイクルアセスメント」ISO 14040:2006(JIS Q 14040:2010)、日本規格協会
・日本規格協会（ＪＳＡ）(2011)『ＪＩＳハンドブック　66-1　ソフトウェア』日本規格協会

・日本規格協会（ＪＳＡ）編(2013)『JIS ハンドブック 66-1 ソフトウェア』日本規格協会
・日本規格協会（ＪＳＡ）(2013a)『ＪＩＳハンドブック　64　情報基本』日本規格協会
・日本規格協会（ＪＳＡ）(2013b)『ＪＩＳハンドブック　66-1　ソフトウェア』日本規格協会
・日本工業標準調査会（ＪＩＳ）情報処理用語解説編集委員会編(1990)『ＪＩＳ準拠コンピュータ／データ通信用語解説集　新版』日本理工出版会
・日本工業標準調査会審議(2009)『JIS X 0141(ISO/IEC 15939)システム及びソフトウェア技術－測定プロセス』日本規格協会
・日本公認会計士協会(1999)「会計制度委員会報告第 12 号　研究開発費及びソフトウェアの会計処理に関する実務指針」
・日本 JBoss ユーザ・グループ(2008)『ＪＢｏｓｓ徹底活用ガイド　Java・オープンソース・Jboss Seam・Jboss AS』技術評論社
・日本情報処理開発センター（ＪＩＰＤＥＣ）(2005)『新版　システム監査基準／システム管理基準解説書』(平成 16 年基準改訂版)ＪＩＰＤＥＣ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2008a)『ＩＴ投資価値評価に関する調査研究　ＩＴ投資価値評価ガイドライン(案)について』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2008b)『検収フェーズのモデル取引・整備報告書ＵＶＣ(User Vender Collaboration)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」』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2010)『2009 年度版「ユーザー企業　ソフトウェアメトリックス調査 2010」報告書』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2011a)『2010 年度版「ユーザー企業 ソフトウェアメトリックス調査 2011」報告書』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2011b)『2010 年度版　企業ＩＴ動向調査 2011』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2012a)『2012 年度版「ユーザー企業 ソフトウェアメトリックス調査 2012」報告書』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2012b)『ユーザー企業　ソフトウェアメトリックス調査 2012 年版　ソフトウェアの開発・保守・運用の実績プロジェクトデータを元に分析』ＪＵＡＳ
・日本情報システム・ユーザー協会（ＪＵＡＳ）(2012c)『企業ＩＴ動向調査報告書 2012(2011 年度調査)』日経ＢＰ社
・日本内部監査協会編(2012)『ＩＴ監査とＩＴ統制　基礎から事業継続・ネットワーク・クラウドまで』同文館出版
・野村総合研究所(2011)『ＩＴ運用管理 攻めのツール活用術　ＮＲＩのデータセンター運用から生

まれた「Ｓｅｎｊｕ　Ｆａｍｉｌｙ」』日経ＢＰ社
・初田賢司(2012)『システム開発のためのＷＢＳの作り方　プロジェクト成功の道しるべ』日経ＢＰ社
・羽深修・志田隆弘・田中智文(2011)『Eucalyptus ではじめるプライベートクラウド構築』インプレスジャパン
・浜田浩児(2001)『93SNA の基礎　国民経済計算の新体系』東洋経済新報社
・林謙三・村上浩貴・鈴木祐介(2013)『実践ＷＢＳ　プロジェクト指向経営の基礎』オーム社
・林雅之(2012)『オープンクラウド入門　CloudStack，OpenStack，OpenFlow，激化するクラウドの覇権争い』インプレスＲ＆Ｄ
・原陽一郎(2009)「第３世代の技術経営（ＭＯＴ）　…組織を超えたマネジメント・システム」『長岡大学　研究論叢』第７号　長岡大学
・東季彦監修(1996)『全訂二版　著作権法』学陽書房
・日立製作所監修(2009)『ＪＰ１による業務システム運用管理の実践』技術評論社
・平松一夫(2012)『ＩＦＲＳ国際会計基準の基礎　第２版』中央経済社
・広川敬佑編著(2011)『ＲＦＰでシステム構築を成功に導く本　ＩＴベンダーの賢い選び方　見切り方』技術評論社
・広瀬義州編著(2011)『財務報告の変革』中央経済社
・広瀬義州(2012)『財務会計　第 11 版』中央経済社
・広瀬義州・藤井秀樹責任編集(2012)『財務報告のフロンティア』中央経済社
・広瀬義州(2014)『財務会計　第 12 版』中央経済社
・藤田晶子(2012)『無形資産会計のフレームワーク』中央経済社
・藤本隆宏（2011）「設計比較優位説のプロセス的基礎」（藤田昌久、長岡貞男編著『生産性とイノベーションシステム』日本評論社）
・誉田直美(2010)『ソフトウェア品質会計　ＮＥＣの高品質ソフトウェア開発を支える品質保証技術』日科技連
・町田祥弘(2011)「内部統制監査の課題と展望」（千代田邦夫・鳥羽至英責任編集『会計監査と企業統治』体系現代会計学第７巻 P. 371-420）中央経済社
・水野行雄(2009)『平成 21 年３月決算会社適用初年度　内部統制報告書の事例と分析』新日本法規
・森秀明(2003)『ＩＴ不良資産』ダイヤモンド社
・安延申・前川徹・田中辰雄(2009)『ビッグトレンド　ＩＴはどこへ向かうのか』アスペクト
・山崎秀彦編著(2010)『財務諸表外情報の開示と保証　ナラティブ・リポーティングの保証』同文舘出版
・リース事業協会編(2012)『リース・ハンドブック〔第 28 版〕』リース事業協会
・渡邊真治(2009)『金融業の情報化と組織に関する経済分析』多賀出版

・渡邉利和・川添貴生(2012)『仮想化インフラを構築する技術』インプレスジャパン

・ＡＩＣＰＡ(1998), *Statement of Position 98-1 "Accounting for the Costs of Computer Software Developed or Obtained for Internal Use"*, New York.(現在の「FASB ACS 350-40 -Internal-Use Software」)

・Carr, Nicholas G.(2004), *Does IT Matter?: information technology and the corrosion of competitive advantage*, Harvard Business School Press, Boston.(ニコラス・Ｇ・カー(清川幸美訳)(2005)『ＩＴにお金を使うのは、もうおやめなさい』ランダムハウス講談社)

・Chrissis, Mary Beth, Konrad, Mike, Shrum, Sandy(2007), CMMI: *Guidelones for Process Integration and Product Improvement*, 2nd Edition, Pearson Education, New Jersey.(メアリー・ベス・クリシス、マイク・コンラド、サンディ・シュラム(JASPIC CMMI V1.2 翻訳研究会訳(2009)『ＣＭＭＩ標準教本 第２版 開発のためのＣＭＭＩ 1.2版対応』日経ＢＰ社)

・Commision of the European Communities / International Monetary Fund / Organisation for Economic Co-operation and Development / United Nations / World Bank(1993), *System of Natinal Accouns 1993*, Brussels,Luxembourg,New York, Paris,Washington,D.C.. (欧州共同体委員会、国際通貨基金、経済協力開発機構、国際連合、世界銀行(経済企画庁経済研究所国民所得部編集)(1996)『1993 年改訂　国民経済計算の体系』上・下巻・索引、経済企画協会)

・Cusumano, Michael A., Selby Richard W.(1995),*Microsoft Secrets*, The Free Press, A Division of Simon & Schuster Inc., New York.(マイケル・Ａ・クスマノ、リチャード・Ｗ・セルビー(山岡洋一訳)(1996)『マイクロソフト・シークレット〈上〉』日本経済新聞社(初版))

・ＦＡＳＢ(1985), *Statement of Financial Accounting Standards No.86 "Accounting for the Costs of Computer Software to Be Sold, Leased, or Otherwise Marketed"*, Norwalk.(現在の「FASB ACS 985-20-25-2 - Costs of Software to be Sold, Leased, or Marketed」)(財務会計基準審議会(日本公認会計士協会　国際委員会訳)(1985)「財務会計基準書第 86 号　販売、リースその他の方法で市場に出されたコンピュータ・ソフトウエア原価の会計処理」)

・Federal Electric Corporation(1964), *PERT COST - A Programed Instruction Manual*, Federal Electric Corporation, Paramus.(Federal Electric Corporation(加瀬滋男訳)(1966)『プログラム学習によるＰＥＲＴ　ＣＯＳＴ入門』日本規格協会)

・Fogel, Karl(2006), *Producing Open Source Software :How to run a successful Free Software Project*, Oreilly Media, Sebastopol.(カール・フォーゲル(高木正弘、高岡芳成訳)(2009)『オープンソースソフトウェアの育て方』オライリー・ジャパン)

・Gamma,Erich, Helm,Richard, Johnson,Ralph, Vlissides,John(1995),*Design Patterns Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, Boston.(エリック・ガンマ、リチャード・ヘルム、ラルフ・ジョンソン、ジョン・ブリシディース(本位田真一,吉田和樹監訳)(1999),

『オブジェクト指向における再利用のためのデザインパターン　改訂版』ソフトバンク　クリエイティブ)

・Glass,Robert L.(2003), *Facts and Fallacies of Software Engineering*, Pearson Education, New Jersey.(ロバート・Ｌ・グラス(山浦恒央訳)(2004)『ソフトウェア開発の55の真実と10のウソ』日経ＢＰ社)

・ Greenfield, Jack, Short,Keith(2004), *Software Factories: Assembling Applications withPatterns, Models, Frameworks, and Tools*, Wiley Publishing, Indiana.(ジャック・グリーンフィールド, キース・ショート(野村一行監訳)(2005),『ソフトウェアファクトリー　パターン, モデル, フレームワーク, ツールによるアプリケーションの組み立て』日経ＢＰソフトプレス)

・Golden, Bernard(2005), *Succeeding with Open Source*, Addison Wesley Professional, Boston.(バーナード・ゴールデン(武藤健志監訳、トップスタジオ訳)(2005)『オープンソース成熟モデル―オープンソースソフトウェアとプロジェクトの実践的評―』毎日コミュニケーションズ)

・Gottesdiener, Ellen(2005), *The Software Requirements Memory Jogger*, GOAL/QPC, Salem.(エレン・ゴッテスディーナー（オージス総研訳)(2009)『実践ソフトウェア要求ハンドブック』翔泳社)

・ＧＲＩ (Global Reporting Initiative) (2006)『サステナビリティ　レポーティング　ガイドライン　Version3.0』ＧＲＩ, Amsterdam.

・Hagel, John Ⅲ(2002), *Out of the Box*, Harvard Business School Press, Boston.(ジョン・ヘーゲルⅢ世(遠藤真美訳)(2004)『今こそ見直したいＩＴ戦略』ランダムハウス講談社)

・Hand, John R. M. and Lev, Baruch(2003), *Intangible Assets: Values, Measures and Risks*, Oxford University Press, Oxford.(ジョン・R.M.ハンド、バルーク・レブ(広瀬義州、畫間 文彦、長束航、中嶋隆一、渡辺剛訳)(2008)『無形資産の評価』中央経済社)

・Haugan, Gregory T.(2002), *Effective Work Breakdown Structures*, Management Concepts, Washington,D.C.(グレゴリー・Ｔ・ホーガン(伊藤衡監訳)(2005)『実務で役立つＷＢＳ入門』翔泳社)

・ＩＡＳＣ(1998), *International Accounting Standards No.36 "Impairment of Assets"*, London.

・ＩＡＳＣ(1998), *International Accounting Standards No.38 "Intangible Assets"*, London.

・ＩＡＳＢ(2010), *"The Conceptual Framework for Financial Reporting"*, London.

・ＩＡＳＢ(2013), Discussion Paper DP/2013/1 *"A Review of the Conceptual Framework for Financial Reporting"*, London.

・ISO(1974), *ISO 2382-1:1974 Data processing -Vocabulary- Section 01: Fundamental terms*, ISO, Switzerland.

・ISO(1984), *ISO 2382-1:1984 Data processing -Vocabulary- Part 01: Fundamental terms*, ISO, Switzerland.

・ISO/IEC(1993), *ISO/IEC 2382-1:1993 Informayion technology -Vocabulary- Part 1: Fundamental*

terms, ISO/IEC Copyright Office, Switzerland.
・ＩＦＰＵＧ(ＪＦＰＵＧ監訳)(2005)『ファンクションポイント計測マニュアル　ﾘﾘｰｽ4.2.1J』ＪＦＰＵＧ, Tokyo.
・ＩＴＩＬ(2012a)『ＩＴＩＬ　サービスストラテジ　2011 edition』ＴＳＯ, Norwich.
・ＩＴＩＬ(2012b)『ＩＴＩＬ　サービスデザイン　2011 edition』ＴＳＯ, Norwich.
・ＩＴＩＬ(2013a)『ＩＴＩＬ　サービストランジション　2011 edition』ＴＳＯ, Norwich.
・ＩＴＩＬ(2013b)『ＩＴＩＬ　サービスオペレーション　2011 edition』ＴＳＯ, Norwich.
・ＩＴＩＬ(2013c)『ＩＴＩＬ　継続的サービス改善　2011 edition』ＴＳＯ, Norwich.
・Jacobson, Ivar, Griss, Matin, Jonsson, Patric(1997), *SoftwareReuse:Architecture, Process and Organization for Business Success*, Addison-Wesley Pub, Boston.(イヴァー・ヤコブソン、マーティン・グリス、パトリック・ジョンソン(杉本宣男、吉田幸彦、落合修、田中正仁監訳)(1999)『ソフトウェア再利用ガイドブック　アーキテクチャー, プロセス, 組織の変革による再利用ビジネス成功への道』トッパン)
・Jones, Capers(1993), *Assessment and Control of Software Risks*, Prentice Hall, New Jersey.(ケーパーズ・ジョーンズ(島崎恭一・富野寿監訳)(1995)『ソフトウェア病理学―システム開発・保守の手引』構造計画研究所 、共立出版)
・Jones, Capers(1995), *Patterns of Software Systems Failure and Success*, International Thomson Computer Press, Boston.(ケーパーズ・ジョーンズ(伊士誠一・富野壽監訳)(1999)『ソフトウェアの成功と失敗』構造計画研究所、共立出版)
・Jones, Capers(1996), *Software Quality: Analysis and Guidelines for Success*, intl Thomson Computer, Boston.(ケーパーズ・ジョーンズ(富野壽監訳)(1999)『ソフトウェア品質のガイドライン』構造計画研究所発行、共立出版発売)
・Jones, Capers(2007), *Estimating Software Costs: Bringing Realism to Estimating*, McGraw-Hill 2nd ed., New York.(ケーパーズ・ジョーンズ(富野壽、岩尾俊二監訳)(2009)『ソフトウェア見積りのすべて第２版』構造計画研究所発行、共立出版発売)
・Jones, Capers(2008), *Applied Software Measurement:Assuring Productivity and Quality*, McGraw-Hill 3rd ed., New York.(ケーパーズ・ジョーンズ(富野壽・小坂恭一監訳)(2010)『ソフトウェア開発の定量化手法　生産性と品質の向上をめざして第３版』構造計画研究所発行, 共立出版発売)
・Jones, Capers(2010), *Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies*, The McGraw-Hill, New York.(ケーパーズ・ジョーンズ(富野壽・岩尾俊二監修、水野哲博・吉田善亮監訳)(2012)『ソフトウェア工学のベストプラクティス　ソフトウェア工学の真の工学への発展をめざして』構造計画研究所発行、共立出版発売)
・Kline, Stephen Jay(1990), I*nnovation Styles In Japan and The United States cultural bases;*

implications for competitiveness, Stanford University, Stanford.（Ｓ．Ｊ．クライン(鴨原文七訳)(1992)『イノベーション・スタイル　日米の社会技術システム変革の相違』アグネ承風社)

・Laird, Linda M.,Brennan, Carol M.(2006),*Software Measurement and Estimation: A Practical Approach*, Wiley-IEEE Computer Society, Los Alamitos.(リンダ・Ｍ・ライルド、キャロル・Ｍ・ブレナン(野中誠, 鷲崎弘宜訳)(2009)『演習で学ぶソフトウェアメトリクスの基礎　ソフトウェアの測定と見積もりの正しい作法』日経ＢＰ社)

・Lattanze, Anthony J.(2009), *Architecting software intensive systems: a practitioners guide*, Taylor&Francis Group,LLC, Boca Raton.(アンソニー・Ｊ・ラタンゼ(橘高陸夫監訳)(2011)『アーキテクチャ中心設計手法　ソフトウェア開発の実践　ソフトウェア主体システム開発のアーキテクチャデザインプロセス』翔泳社)

・Lev, Baruch(2001), *Intangibles: Management, Measurement, and Reporting*, The Brookings Inst Press, Washington D.C..(バルーク・レブ(広瀬義州・桜井久勝監訳)(2002)『ブランドの経営と会計』東洋経済新報社)

・McClure,Carma(1989), *CASE is Software Automation*, Prentice-Hall, New Jersey.(カーマ・マックルーア(三井銀総合研究所訳)(1990)『ＣＡＳＥ』日経ＢＰ社)

・Moore, Geoffrey. A.(1991), *Crossing The Chasm: Marketing and Selling Technology Products to Mainstream Customers*, Harpercollins, New York.(ジェフリー・ムーア(川又政治訳)(2002)『キャズム　ハイテクをブレイクさせる超マーケティング理論』翔泳社)

・ＯＥＣＤ(2010), *National Accounts at a Glance 2010*, OECD, Paris.(OECD 編著(中村洋一監訳、高橋しのぶ訳)(2011)『図表でみる国民経済計算　マクロ経済と社会進歩の国際比較』明石書店)

・Pressma, Roger S.(2000), *Software Engineering: a Practitioner's Approach* 5th ed., McGraw-Hill, New York.(ロジャー・Ｓ・プレスマン(西康晴・榊原彰・内藤裕史監訳)(2005)『実践ソフトウェアエンジニアリング　ソフトウェアプロフェッショナルのための知識』日科技連)

・Pressman, Roger S.(2005), *Software Engineering: a practitioner's 6th edition*, McGraw-Hill, New York.(ロジャー・プレスマン(西康晴・榊原彰・内藤裕史監訳)(2005)『実践ソフトウェアエンジニアリング　ソフトウェアプロフェッショナルのための基本知識』日科技連)

・Project Management Institute(2006), *Practice Standard for Work Breakdown Structures-Second Edition-*, Project Management Institute, Newtown Square.(Project Management Institute(ＰＭＩ東京支部監訳)(2008)『ワーク・ブレークダウン・ストラクチャー実務標準　第２版』新技術開発センター)

・Porter, Machael E.(1998), *On Competition*,Harvard Business School Press, Boston.(マイケル・Ｅ・ポーター(竹内弘高訳)(1999)『競争戦略論Ⅰ』ダイアモンド社)

・Raymond, Eric Steven(1998), *The Cathedral and the Bazaar*, (エリック・Ｓ・レイモンド(山形浩生訳)(2010)『伽藍とバザール』ユニバーサル・シェル・プログラミング研究所)(http://www.catb.org/esr/writings/cathedral-bazaar/)

・Roth, Bud Porter(2002),*Request for Proposal: A Guide to Effective RFP Development*, Addison-Wesley, Boston.(バド　ポーター・ロス(渡部洋子監訳)(2004)『ＲＦＰ入門　はじめての提案依頼書』日経ＢＰソフトプレス)

・Solow, Robert M.(1987), *We'd Better Watch Out*, The New York Times Book Review July 12 p.36, New York.

・Suh, Nam P.(1990), *The Principles of Design*, Oxford University Press, New York.(Ｎ・Ｐ・スー(畑村洋太郎監訳)(1992)『設計の原理　―創造的機械設計論―』朝倉書店)

・Tanenbaum, Andrew and Woodhull, Albert S.(2006),*Operating Systems, Design and Implementation*, 3rd Edition, Peason Education, New Jersey.(アンドリュー・Ｓ・タネンバウム、アルバート・Ｓ・ウッドハル(古澤康文、木村信二、永見明久、峯博史訳)(2007)『オペレーティングシステム　第３版　設計と実践』ピアソン・エデュケーション)

・Tidd, Joe, Bessant, John, Pavitt, Keith(1997),*Managing Innovation: Integrating Technological, Market and Organizational Change*, John Wiley & Sons, Hoboken.(ジョー・ティッド、ジョン・ベサント、キース・パビット(後藤晃、鈴木潤訳)(2004)『イノベーションの経営学――市場・組織の統合的マネジメント』ＮＴＴ出版)

・Tobin, James(1966), *National Economic Policy*, Yale University Press, New Haven.(ジェームズ・トービン(間野英雄、海老沢道進、小林桂吉訳)(1967)『国民のための経済政策』東洋経済新報社)

・Tobin, James(1980), *Asset Accumulation and Economic Activity: Reflection on Contemporary Macroeconomic Theory*, Basil Blackwell, Oxford.(ジェームズ・トービン(浜田宏一、藪下史郎訳)(1981)『マクロ経済学の再検討』日本経済新聞社)

・Tobin, James(1998), *Money, Credit, and Capital*, The McGraw-Hill, New York.(ジェームズ・トービン(藪下史郎、大阿久博、蟻川靖浩訳)(2003)『トービン　金融論』東洋経済新報社)

・Tracz, Will(1995), *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*, Addison-Wesley Pub, Boston.(ウィル・トレイツ(畑崎隆雄・林雅弘・鈴木博之訳)(2001)『ソフトウェア再利用の神話』ピアソン・エデュケーション)

・Weber, Steven(2004), *The Success of Open Source*, Harvard University Press, Cambridge.(スティーブン・ウェバー(山形浩生、守岡桜訳)(2007)『オープンソースの成功　政治学者が分析するコミュニティの可能性』毎日コミュニケーションズ)

・Wellman, Frank(1992), *Software Costing: An Objective Approach to Cost of Computer Software*, Prentice Hall, New Jersey.(フランク・ウェルマン(櫻井通晴監訳、新江孝訳)(1996)『ソフトウェア原価の見積りと管理』白桃書房)

・Yourdon, Edward(2004),*Death March 2nd Edition*, Pearson Education, New Jersey.(エドワード・ヨードン(松原友夫・山浦恒央訳)(2006)『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)

筆者の既発表論文

・長田芙悠子(2012)「ソフトウェア再利用に適合的な会計処理―ソフトウェアに係る会計基準の不備を補正するために―」『経理知識』第 91 号 (P.63-78(研究ノート))、明治大学国家試験指導センター経理研究所

・長田芙悠子(2013)「ソフトウェア開発の失敗に関する会計処理案―ソフトウェアの仕損を会計ではどのように捕捉すればよいか―」『明治大学専門職大学院研究論集』第５号(P.59-78)、明治大学専門職大学院

・長田芙悠子(2013)「ソフトウェア開発における研究開発と非研究開発―ソフトウェアに係る会計基準のセントラル・ドグマの解体へ―」『経営学研究論集』第39号(P.89-109)、明治大学大学院

・長田芙悠子(2013)「クラウド・コンピューティングが惹起する会計的問題―クラウド事業者並びに利用企業の実態調査に基づく考察―」『経理知識』第 92 号 (P.21-43(研究ノート))、明治大学国家試験指導センター経理研究所

・長田芙悠子(2014)「ソフトウェア保守に適合的な会計処理―ソフトウェアに係る会計基準の「隙間」を匡正する提案―」『経営学研究論集』第 40 号(P.89-110)、明治大学大学院

・長田芙悠子(2015)「コンテンツ会計の前梯的考察―ゲームコンテンツ資産の会計慣行における問題点―」『経営学研究論集』第43号(P.1-22)、明治大学大学院

付録　ソフトウェア用語集

１．組織・団体

［001］**ＣＳＡＪ**　Computer Software Association of Japan の略号。日本語正式名：一般社団法人　コンピュータソフトウェア協会。1982 年設立。業界団体。パソコン・ソフトないしパッケージ・ソフトを主たる事業とする企業が加盟。会員企業数計 409 社（2014 年 6 月 1 日現在)。
(http://www.csaj.jp/)

［002］**ＩＦＰＵＧ**　International Function Point Users Group の略号。FP 法の普及を図る国際的な団体。≪参照→［000］FP 法≫
(http://www.ifpug.org/?lang=ja)

［003］**ＩＰＡ**　Information-technology Promotion Agency の略号。日本語正式名：独立行政法人　情報処理推進機構。2004 年設立だが、前身である認可法人　情報処理振興事業協会の設立は 1970 年（ＩＰＡはその時からの通称)。経済産業省（旧通産省）所管。情報処理技術者試験の実施機関。国策である情報産業の育成を立法化した機振法（機械工業振興臨時措置法、1956 年制定）～機情法（特定機械情報産業振興臨時措置法、1978 年制定）の推進機関。
(http://www.ipa.go.jp/index.html)

［004］**ＩＳＯ**　International Organization for Standardization の略号。国際標準化機構。電気分野を除く工業分野の国際的な標準である国際規格を策定するための民間の非政府組織である（電気分野の同様の組織はＩＥＣ：International Electrotechnical Commission、国際電気標準会議である)。ソフトウェア工学の各種標準以外では、ISO9001（品質マネジメントシステム)、ISO27001（情報セキュリティマネジメントシステム）がソフトウェアに関連のある代表的な標準である。

［005］**ＪＥＩＴＡ**　Japan Electronics and Information Tecnology Industries Association の略号。日本語正式名：一般社団法人　電子情報技術産業協会。業界団体。電子工業（ハードウェア）を主たる事業とする企業が加盟。正会員企業数 279 社（2014 年 5 月 13 日現在)。
(http://www.jeita.or.jp/)

［006］**ＪＦＰＵＧ**　Japan Function Point Users Group の略号。日本語正式名：日本ファンクションポイントユーザ会。1994 年設立。1996 年にＩＦＰＵＧの日本支部となった。FP 法の普及を図る団体。教条的ではなく、他の測定手法の研究会を開催する等のことも行なっている。

(http://www.jfpug.gr.jp/)

［007］ＪＩＰＤＥＣ　Japan Institute for Promotion of Digital Economy and Community の略号。財団法人　日本情報処理開発センター（ＪＩＰＤＥＣ、1967 年設立)、財団法人　日本経営情報開発協会（ＣＵＤＩ、1968 年設立)、財団法人　情報処理研修センター（ＩＩＴ、1970 年設立)の３団体が 1976 年に統合し、財団法人　日本情報処理開発協会（ＪＩＰＤＥＣ）となった。経済産業省（旧通産省）所管。2011 年、一般財団法人　日本情報経済社会推進協会に改称。Ｐマーク（プライバシーマーク）やＩＳＭＳ（情報セキュリティマネジメントシステム）の審査機関。長らくＩＰＡとの役割分担、独立的な意義が問題視されてきたが（かなり重複した活動を行なっていた時期があった)、2011 年の改組に伴い、事業が縮小された。

(http://www.jipdec.or.jp/)

［008］ＪＩＳＡ　Japan Information Technology Services Industry Association の略号。日本語正式名：一般社団法人　情報サービス産業協会。社団法人　日本情報センター協会（1970 年設立)、社団法人　ソフトウェア産業振興協会（1970 年設立）が 1984 年に合併。業界団体。法人会員数 517 社（2014 年 6 月 1 日現在)。業界団体は他にも幾つかあるが、歴史も古く、最も有力な団体と言えるかもしれない。会費が比較的高いこともあり、主として大手のソフトウェア企業が会員となっている。ソフトウェア会計に関する調査・研究を時々に行ない、提言を行なっている、ほぼ唯一の団体と言える。

(http://www.jisa.or.jp/)

［009］ＪＳＡ　Japanese Standards Association の略号。日本語正式名：一般財団法人　日本規格協会。日本工業規格（JIS）原案の作成、JIS 規格票の発行、出版物（『JIS ハンドブック』等)の発行等を行なう法人。ソフトウェア関係の JIS 規格は、ほとんどが ISO/IEC 標準を日本語化したものである。

(http://www.jsa.or.jp/)

［010］ＪＵＡＳ　Japan Users Association of Information Systems の略号。日本語正式名：一般社団法人　日本情報システム･ユーザー協会。1962 年設立の日本データ･プロセシング協会が、1992 年に社団法人日本情報システム・ユーザー協会に改組。ユーザ企業の団体。正会員企業数計 2,527 社（2014 年 6 月 1 日現在)。『企業ＩＴ動向調査報告書』、『ソフトウェアメトリックス』の各年度版を発行している。ソフトウェア会計実務に関するセミナーを時々に開催し、啓蒙活動を行なっている。

(http://www.juas.or.jp/)

［011］**ＳＥＩ**　Software Engineering Institute の略号。カーネギー・メロン大学のソフトウェア工学研究所のことであるが、固有名になるほど高名である。ＣＭＭＩ［999］を考案した実績がある。

２．ソフトウェア技術

［101］**ISMS**　Information Security management System（情報セキュリティマネジメントシステム）の略号。ISO/IEC 27001(JIS Q 27001)に準拠。認証取得組織 4,507（2014 年 6 月 13 日現在)。

［102］**ITSS**　IT Skill Standards（ＩＴスキル標準）の略号。情報処理技術者試験の試験区分よりも、職種を拡げ、スキルレベルをレベル１～レベル７とし、情報サービス産業における人材のスキルの指標としての普及を目指している。今のところ、いつものパターンだが、大手のソフトウェア企業が下位企業への委託の単価切り下げに「悪用」している程度の利用が主たるものである。

［103］**ITSMS**　IT Service management System（ＩＴサービスマネジメントシステム）の略号。ISO/IEC 20000 に準拠。運用を主たる事業とする企業が資格取得する。認証取得組織 190（2014 年 6 月 13 日現在)。

［104］**ＩＴＩＬ**　Information Technology Infrastructure Library の略号。ＩＴサービスマネジメントにおけるベストプラクティス（成功事例）をまとめた書籍群。1989 年、イギリス政府のＣＣＴＡ（Cental Computer and Telecommunication Agency：中央コンピュータ電気通信局、イギリス政府商務庁の下部組織）によって公表された。現在はｉｔＳＭＦ（IT Service Management Forum）という非営利団体が運営している。2001 年Ｖ２、2007 年Ｖ３に改訂され、最新版はＩＴＩＬ2011 である。ISO/IEC 20000（ＩＴサービスマネジメント）となり、ＩＴＳＭＳ（IT Service Management System）という認証資格のベースになっている。当初はシステム運用の標準という性格が強かったが、次第に「ＩＴサービス」という概念を拡張・洗練させ、システムに関わる広範囲の標準へと進化している。

［105］**アクティビティ**（activity）　≪参照→［131］**フェーズ**≫

［106］**ＡＳＰ**　Application Service Provider の略号。アプリケーション・ソフト等のサービスをネットワーク経由で提供する事業者。今日では、クラウド・コンピューテング（cloud computing)、あるいはその中の **SaaS**（Software as a Service）がほぼ同義である。

［107］**FP**　Function Point の略号。**ファンクション・ポイント**とも表記する。ソフトウェアの規模を表す尺度の１つで、機能（function）の規模をポイント数で表す。≪参照→［108］FP 法≫

［108］**FP 法**(function point method)　1979 年、ＩＢＭのアレン･アルブレヒト(Allen J. Albrecht)が考案したソフトウェアの規模を測定する手法の１つである。機能（function）に着目し、それを実現する入出力の画面・帳票・ファイルの数や取り扱う項目数等を点数付けしていくのである。ＩＦＰＵＧが標準化し、普及活動を行なっている。但し、普及に伴い、標準とは異なる幾つかのバリエーションが表れている。

［109］**LOC**　Lines Of Code の略号。**行数、ステップ数、ライン数**とも言う。ソフトウェアの規模を表す尺度の１つで、プログラムの行数のことである。測定する対象のプログラム形態は、ソースプログラム（ソースコード）である。それをより明確に示したい場合には**SLOC**(source lines of code)と言う。ソフトウェアの規模が大きい場合には、1,000 行単位の**KLOC**、1,000,000 行単位の**MLOC**等で表す。

［110］**ＯＳ**（Operating System）　オペレーティング・システムとも言う。ハードウェア寄りのソフトウェアを基本ソフトウェアと呼ぶが、その中で最もハード寄りなのがＯＳであると言える。また、あらゆるソフトウェアの中で、最も多く利用されているのがＯＳである。特定のソフトウェアが作動中であっても、常に割り込みを察知し、制御を変更できるように、メモリに常駐し、待機状態におり、そういう意味では常に「不眠不休」の状態にあると言える。

［111］**オープンソース**（Open Sourse）　OSS(Open Sourse Software)、フリーソフトウェア（free software）とも言う。Free は、「無償」と「自由」の両義性があり、当初は両方を含意させた思想的な運動の性格が濃厚であった。次第に、そうした性格は後景に退いたが、商用ソフトウェアとは一線を画した形で、オープンソースの存在感を有しつつ利用され、保守が続けられている。

［112］**機械語**（machine language）　**マシン語**、とも言う。ハードウェア（マシン）上で、実際に動作するプログラムは、言語的には機械語で表現されているものだけである。２進コード（０と１）で記述する。ハードウェアに圧倒的に依存しているので、ハードウェアの機種毎に各々異なった機械語がある、というのが実状である（類似的ではあるけれども）。

［113］**規模**（scale）　ソフトウェアの規模は一般的には LOC 又は FP で表す。概算規模としては画面数や帳票数で表すこともある。また、ビジネス上の表現としては、開発工数や開発費で表すこ

ともある。

［114］**工数**（workload）　人が一定期間（時間）従事することで遂行する作業量のこと。単位は、作業規模により、人時、人日、人月、人年を使う。ソフトウェアでは、人月という単位が最も多く使われる。

［115］**コンパイラ**（compiler）　プログラミング言語で書かれたソース・プログラム（原始プログラム）をハードウェア上で作動できる形態、オブジェクト・プログラム（目的プログラム）に変換（翻訳）するツール（ソフトウェア）である。第１世代のプログラミング言語である機械語はそのままで作動できるが、第２世代のプログラミング言語であるアセンブリ言語（assembly language）からは機械語への変換が必要であり、アセンブラ（assembler）により変換を行なう。それをプログラミング言語一般に拡張したのがコンパイラである。大凡の機能（処理手順）は、字句解析―構文解析―意味解析―変換―最適化、である。各言語から一挙に機械語に変換することもあるが、多くは各言語から一旦中間言語ないしアセンブリ言語に変換し、中間言語ないしアセンブリ言語から機械語に変換するという２段階方式を採る。また、予め変換しておくプリ・コンパイラ方式と実行時に変換するインタープリタ（通訳）方式がある。

［116］**ＣＭＭＩ**　Capability Maturity Model Integration（能力成熟度モデル統合）略号。元々はＣＭＭ（Capability Maturity Model：能力成熟度モデル）として開発されたが、2000 年に発展的に改訂され、今日に到っている。組織（企業等）のソフトウェア成熟度をレベル１～レベル５で評価する。カーネギー・メロン大学のソフトウェア工学研究所（ＳＥＩ）が考案したものであり、企業が認証取得する資格となっている。

［117］**生産性**（productivity）　元々は経済学用語で、労働生産性（labor productivity）、全要素生産性（Total Factor Productivity：ＴＦＰ）といった区分があるが、ソフトウェア開発等での生産性は専ら労働生産性のことであり、しかも「労働」を冠しない。生産性（KLOC/人月、FP/人月）は、規模／工数により算出する。

［118］**JCL**　Job Control Language（ジョブ制御言語）の略号。メインフレーム系で使われるバッチ・ジョブ制御用のスクリプト言語である。プログラム内の論理ファイル名と、実際の物理ファイル名などを関連付ける。

［119］**ソフトウェア工学**（software engineering）　1968 年、ＮＡＴＯ（北大西洋条約機構）の Software Engineering Conference 以来用語としては普及した。但し、呉呉も誤解してはならな

いのは、ソフトウェアは一応工学的なものとして看做されており、ソフトウェア業界等での技術革新等は目覚ましいが、いわゆる大学における「学問」としてのソフトウェア工学は低調（低水準）であり、それが実務やソフトウェア業界に寄与したことは僅かしかない。そういう意味で、広義のソフトウェア工学は広大な領域としてあるが、狭義の（大学における学問としての）ソフトウェア工学は取るに足りないものである。特に致命的なのは、大学の教員で、大規模なソフトウェア開発の経験を有するものはごく限られているので、それを教授できないことである。

［120］**タスク**（task）　≪参照→［131］**フェーズ**≫

［121］**ＷＢＳ**　Work Breakdown Structure の略号。プロジェクト管理における作業項目の細分化・洗い出しを行なうための手法である。

［122］**情報の単位 K,M,G,…**　1B(BYTE：バイト)=8b(bit：ビット)。1K(キロ)B=1,000B=10^3B≒2^{10}B、1M(メガ)B=1,000KB=10^6B≒2^{20}B、 1G(ギガ)B=1,000MB=10^9B≒2^{30}B、1T(テラ)B=1,000GB=10^{12}B≒2^{40}B、1P(ペタ)B=1,000TB=10^{15}B≒2^{50} B、1E(エクサ)B=1,000PB=10^{18}B≒2^{60}B、 1Z(ゼタ)B=1,000EB=10^{21}B≒2^{70}B、1Y(ヨタ)B=1,000ZB=10^{24}B≒2^{80}B。

［123］**ＤＢＭＳ**（DataBase management System）　ＤＢ（DataBase）はファイルの発展形である。多様なアクセスができること（条件検索等）、ファイルの堅牢性（障害時のデータの保証ないし復元の高確度性等）が特長である。そのため、ＯＳのファイル管理機能に基づき、固有のソフトウェア（マネジメント・システム）により処理・管理を行なう。

［124］**人月**（man-month）　工数の尺度単位である。１人の技術者が１ヶ月、所定時間（例えば168H）作業をすると１人月であり、同じ１ヶ月でも1.5倍の時間（252H）作業をすれば1.5人月とカウントする。フレデリック・ブルックス（Frederick Phillips Brooks, Jr.）の名著『人月の神話』（*The Mythical Man-Month: Essays on Software Engineering*）で人口に膾炙したとも言える。

［125］**標準**（Standards）　ソフトウェアの各種標準は、原語では同じStandardsだが、会計「基準」とは性格が大きく異なる。それを弁別しておかないと、誤解が生ずる可能性があるので、注意を要する。端的に言って、ソフトウェアの各種標準にはいわゆる規制力、規範力はほとんどないのである。例えば、一般的な工業分野ではＪＩＳ規格は相当程度遵守されるだろうが、ソフトウェア分野のＪＩＳ規格に関してはそもそもそういうものがあることを知っている技術者すらごく限られているであろう。近年では、情報セキュリティ等に関してはかなり知られるようになってきたが、技術的な標準に関しては知名度すら向上しているとは言えない。標準化は、個々の開発プロジェク

トないし企業単位で行なわれるようになってきたが、その延長上で大手のソフトウェア企業系列で広がるくらいが限度で、それ以上の広がりでは拘束力がそれほど強くないデファクト・スタンダードがある程度あるくらいである。ソフトウェア分野で最も遵守率の高い標準は、通信プロトコルとプログラミング仕様である。前者は、遵守しなければそもそも接続できないからである。後者は、遵守しないとコンパイラでエラーとなり、オブジェクト・モジュールが生成されないか、もしくは生成されても動作が不安定になる可能性があるので、余程の理由（高性能としたいために裏技を使う等）がない限り標準を遵守するのである。

［126］**フレームワーク**（Framework）　会計分野では近年フレームワークと言えば、大凡ＩＦＲＳ等の「概念フレームワーク」を指すことが多いが、ソフトウェア分野で近年多用される**フレームワークというの**は抽象的なものではなく、専ら具体的なツール群のことである。小さな局所的なものから、システム・アーキテクチャの全域に対応する包括的なものまである。開発ツールとしてだけではなく、本番運用におけるコンポーネントとして動作し続けるものもある。

［127］**プログラム形態**　プログラミング言語により記述した段階からハードウェア上で作動する段階までの形態的な変化を捉えるものである。概念的なモデルとしては、**ソース・プログラム**→（コンパイル）→**オブジェクト・プログラム**→（リンカ）→**ロード・プログラム**、となる。プログラムの替わりに、**モジュール**、**コード**、とも言う。

［128］**プログラムの呼称**　プログラムに関して、様々な呼び方（言い方）がある。**プログラム**（program）、**モジュール**（module）、**ルーチン**（routine）、**マクロ**（macro）、**部品**（parts）、**関数**（function）などと言う。

［129］**プログラミング言語**　第１世代の機械語、第２世代のアセンブリ言語に引き続き、第３世代の言語はＦＯＲＴＲＡＮ、ＣＯＢＯＬを始めとして数多く出現し、今日でも使われているものも少なくない。４ＧＬ（第４世代言語）、あるいはそれ以降のスクリプト（Script）言語も陸続と出現している。プログラミング言語は、数え方にもよるが、総数800とも2,500とも言われるが、創成期以来変わっていないのは、実行形態は唯一機械語だということである。

［130］**プロジェクト管理**　ソフトウェア開発が大規模化するに連れ、プロジェクト管理は欠かせないものとなった。今日では、管理ツールを利用する場合も多い。それにも関わらず、プロジェクトの失敗は顕著に減少していないし、膨大なプロジェクト管理文献（学術書から実用書に到るまで）が刊行され続けているが、顕著に効果を発揮したと人口に膾炙されるようなものは１つも出現していない。非常に興味深く、且つ難解を極める領域である。

［131］**フェーズ**（phase）　**工程**とも言う。開発の工程を幾つかに分割した、１つの作業単位である。例えば、基本設計フェーズ、詳細設計フェーズ、プログラム製造フェーズ（単体テストフェーズを含む）、結合テストフェーズ、総合テストフェーズ、といった分割の仕方がある。同じフェーズ内をより細かな作業単位に分割したものが**アクティビティ**である。アクティビティをより細かな作業単位に分割したものが**タスク**である。**フェーズコアクティビティコタスク**、という包含関係となる。

［132］**要求工学**（Requirements engineering）　比較的近年、次第に、ソフトウェア開発プロジェクトの失敗はより上流工程で仕様を明確にしなければ回避することは難しいという経験から、より上流工程を重視する傾向が顕著になってきた。その一環として、要求（要件）を工学的な手法により明確化しようとする分野である。

［133］**リンカ**（linker）　リンケージ・エディタ（linkage editor：連係編集プログラム）とも言う。機械語の個々のプログラムを結合し、実行可能な状態にするツール（ソフトウェア）である。予め実行前に結合状態にしておく静的リンク（Static Link）と、実行時に結合する動的リンク（dynamic link）という２つの方式がある。