ソフトウェア経済学の概要
－コスト分析・資産評価への科学的アプローチ－

株式会社 一（いち）　副社長　専任コンサルタント
大槻　繁

1．はじめに

『ソフトウェア経済学』、どこかできいたような、でもなんか新しそうな響きの言葉でしょう。

近年、産業パラダイムも、知識主導型へ急激に移行してきています。ものづくりからサービスへ、グローバル化・自由市場の浸透、ビジネスとシステムとの一体化もすすんでいる中で、社会/経済的な活動を説明する新しい理論が望まれています。

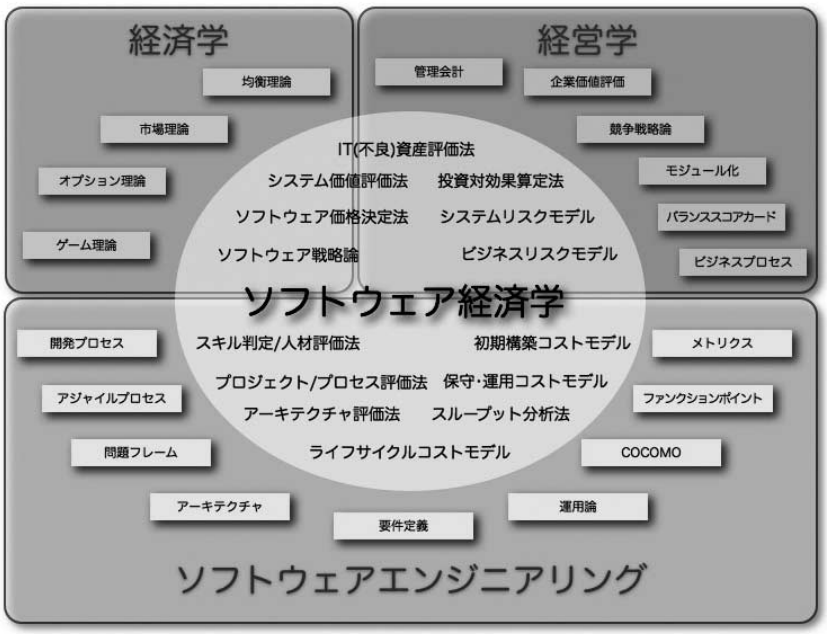


図１　ソフトウェア経済学の全体像

ソフトウェアは知的活動の成果です。社会になくてはならない存在になっており、人間の経済活動の大きな部分を占めているにもかかわらず、それを支える理論が不十分なのです。

本論文では『ソフトウェア経済学』という研究・技術領域を提唱し、その一つの説明事例として「コスト分析手法」と「ソフトウェア資産評価」をとりあげます。

Text by *Shigeru Otsuki*
日立製作所にてソフトウェアエンジニアリングの研究・開発に従事。2004年よりコンサルタント会社一（いち）副社長/専任コンサルタント。ITシステム関連の調達・開発プロジェクトの見積り評価､診断・改善のコンサルティングを行うかたわら､コストモデルや経済モデルの研究・開発を進めている。
著作に「ソフトウェア設計」（朝倉書店,1995年)、「ソフトウェアクリーンルーム手法」（日科技連,1997)、「大丈夫かあなたの会社のIT投資」（NTT出版,2002年）他多数。
IPA/SEC見積り手法部会委員、電子情報技術産業協会ソフトウェアエンジニアリング技術専門委員会委員、アジャイルプロセス協議会運営委員長・副会長。　http://www.1corp.co.jp

2．ソフトウェア経済学

ソフトウェアに関わる経済活動は、ますます重要になってきています。家庭でもワープロ、表計算、メールソフトなどのパッケージ製品が日常的に使われています。実際に対価も払っているでしょうし、バージョンアップの知らせなんかもきて、ソフトウェアってお金がかかるという実感もあるでしょう。企業活動でも、いわゆるIT投資という名のもとに、ソフトウェア開発に莫大な資金が投入されていますし、ソフトウェアなしで企業活動そのものが存続できなくなっています。

さほど重要な割に、ソフトウェアの値段が適正かどうか判断するよりどころはあまりに希薄です。企業が開発ベンダーやソフトハウスに開発を依頼する場合に、その契約額の妥当性の論拠はどこに求めればよいのでしょうか。

図１に『ソフトウェア経済学』の全体像を示

します。経済学は、近代資本主義における「富」に関する学問であり、ソフトウェアに関わる経済活動を説明する原理のよりどころとなり得る分野です。経営学の領域は、利益追求や社会貢献を目的とした企業・組織体に関する方法論を主として扱っています。そして、ソフトウェアエンジニアリングは、ソフトウェアの開発・保守、アーキテクチャ、プロセス、プロジェクトマネジメントに関する知識体系です。

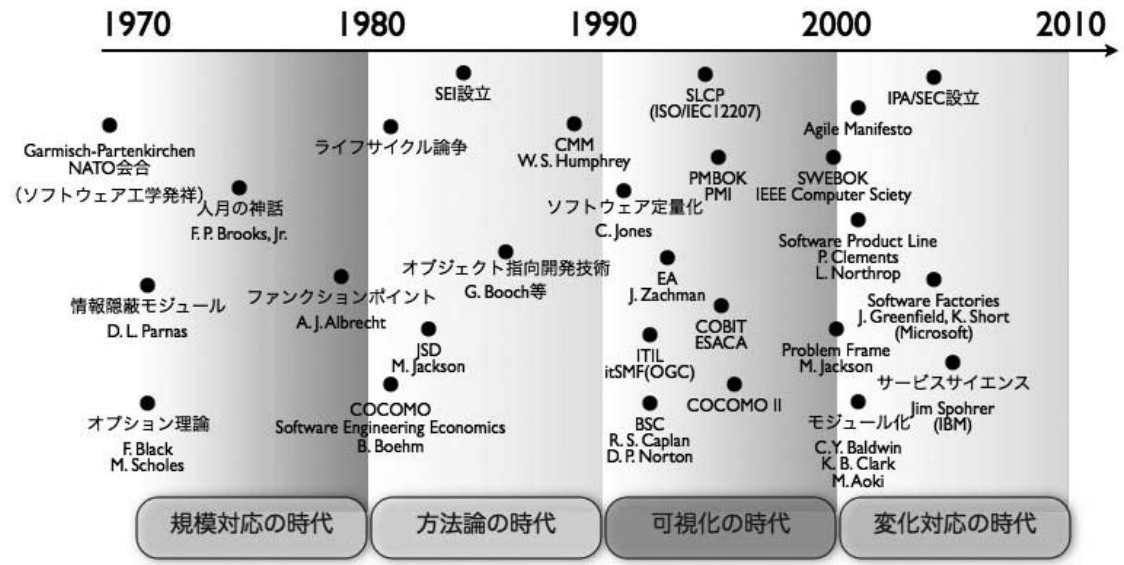
これ等三つの学問・技術領域を背景として、それ等を統合し、『ソフトウェア経済学』では、ソフトウェアに関わる以下の事項を明らかにしようと考えています。

- ❖ **ソフトウェア、システム、サービス等の無形財の利用、開発、保守、運用、破棄の総合的な社会/経済的な振舞い**
- ❖ **市場、組織、部門、プロジェクト、チーム、個人の一貫した社会/経済的な振舞い**
- ❖ **価値、価格、費用（コスト）の定式化と、これ等の間の関係**

3．潮流

図２は、近年の『ソフトウェア経済学』に影響があると思われる主要な理論提唱、出来事をまとめたものです。ソフトウェアエンジニアリングの発祥は、1968年のNATO会合といわれています。以降、さまざまな方法論や手法が提唱されています。70年代はプログラミングを系統的に行うことや、計算に関する基本的な原理が数多く生まれました。80年代は、方法論や開発プロセスに関心が寄せられるとともに、プロジェクトとして開発行為を捕らえ、データを収集・分析する定量化アプローチも出現しました。90年代には、知識体系を標準化し、経営的視点、運用といったより広範な取組みがなされました。90年代後半からインターネットが急速に普及したこともあいまって、今世紀に入り問題把握、要件定義、ビジネスプロセスと開発プロセスの統合等に関心が集まっています。

IBMの巨大なオペレーティングシステムの開発に携わったF. P. Brooks Jr.は、1975年の著作『人月の神話』の中で、ソフトウェアに関する本



BSC:Balanced Score Card　CMM:Capability Maturity Model　COBIT:Control Objectives for Information and related Technology
COCOMO:Constructive Cost Model　EA:Enterprise Architecture　IPA:Information technology Promotion Agency
ISACA:Information Systems Audit and Control Association　ITIL:Information Technology Infrastructure Library
itSMF:IT Service Management Forum　JSD:Jackson System Development　NATO:North Atlantic Treaty Organization
OGC:Office of Goverment Commerce　PMBOK:Project Management Body of Knowledge　PMI:Project Management Institute
SEI:Software Engineering Institute　SEC:Software Engineering Center　SLCP:Software Life Cycle Process
SWEBOK:Software Engineering Body of Knowledge

図２　ソフトウェア経済学に関連した年表

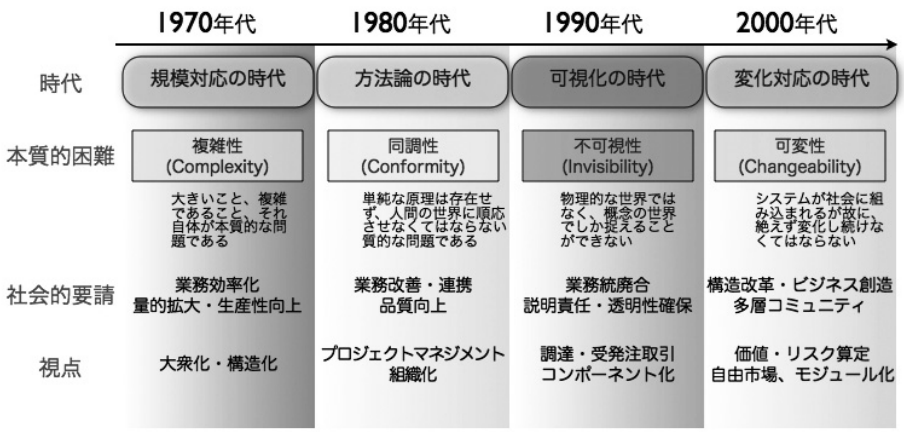


図３　ソフトウェアの本質的困難と時代の要請

質的困難として複雑性、同調性、不可視性、可変性をあげています。図３にこれ等の説明と、各年代、社会的要請、視点を整理したものを示します。筆者の整理は、大局的には本質的困難をそれぞれの時代で一つ一つ解決してきたように見ることもできるということです。無論、2000年代に入ったからといって、複雑性、同調性等の観点が不要になったわけではなく前提となったと見るべきで、新規の主題が可変性への対応となっていることです。

ソフトウェアの領域に、建築やハードウェア製造の技法・ツールやマネジメント手法を援用する場合には、上記のソフトウェアの本質的困難をどういう局面で解決しているかという視点で評価する必要があると考えています。

技術開発は時間を要するものです。ファンクションポイント法やCOCOMOは、現在各方面のコスト分析や見積りの現場で使われていますが、これ等が提唱されたのは1980年前後です。その間、プロジェクトデータの収集・分析、モデル式の洗練化等の地道な努力の積み重ねがありました。90年代になり可視化の要請もあり、調達の透明性を確保したり、見積りの第三者評価等に両手法は有効と考えられています。最近の変化対応の要請という観点では、アジャイルプロセスやインクリメンタル（段階的拡充）開発といった新しい開発プロセスでの分析手法が要求されていますし、ユーザ企業と開発企業という調達プロセスの中でも、見積りの算出、説明、交渉といった場面に応じた手法の活用方法を開発していく必要があります。

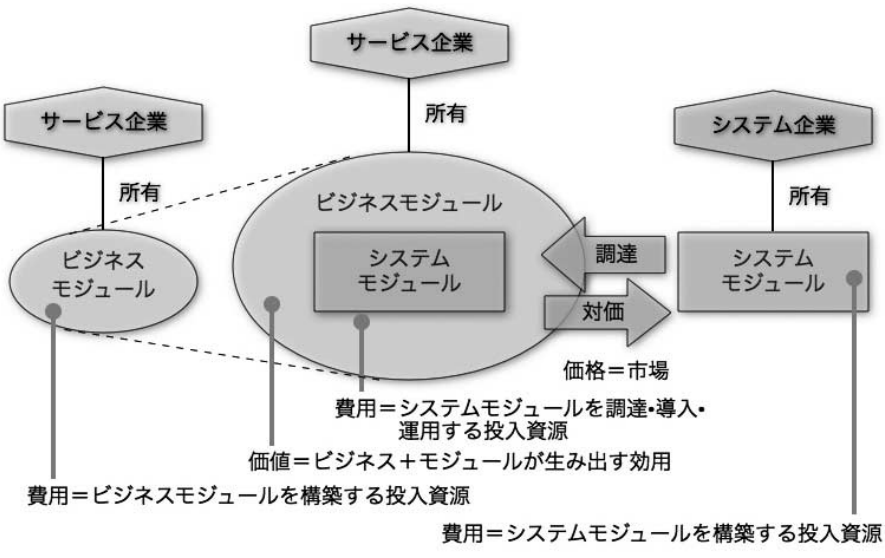


図４　ソフトウェアの価値、価格、費用の関係

ビジネスプロセスと開発プロセスとの親密な連携もすすんできており、コスト分析手法も規模と工数・期間の関係分析だけでなく、その周辺の活動を統一的に扱える体系に発展させていく必要があります。図４は、ソフトウェアに関わる価値、価格、費用（コスト）に関する関係を端的に表したものです。

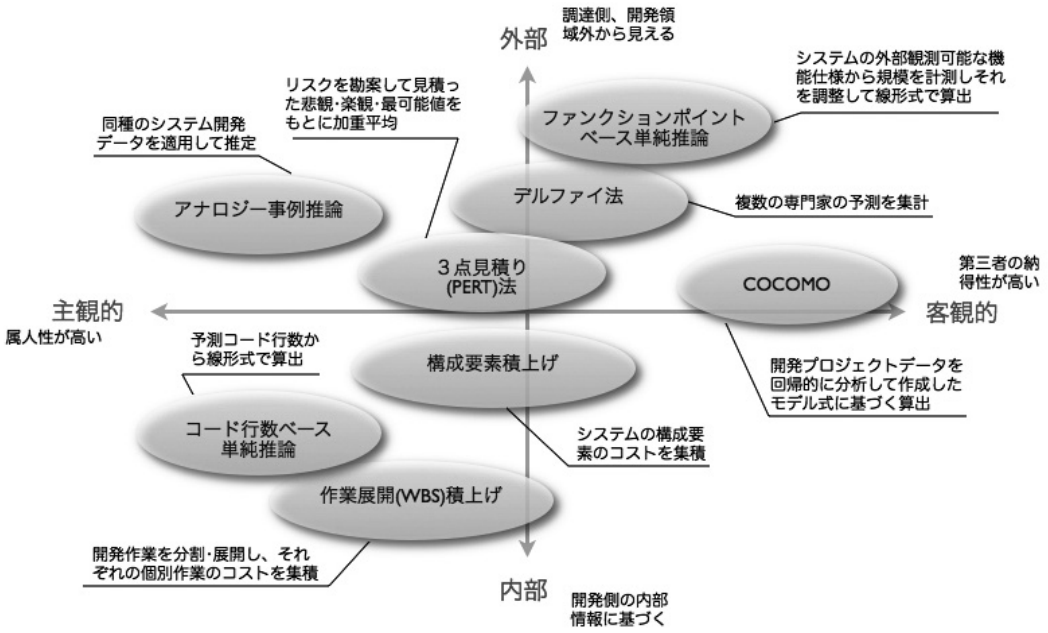


図５　見積り技術の体系

４．コスト分析手法

4.1．見積り技術の全体像

ソフトウェア開発における見積り手法は、開発の初段、プロジェクトの遂行、開発後の評価等の局面で、さまざまなものが複合的に使われています。図５は、代表的な見積り技術を整理したものです。図の二つの軸は、見積り担当者の力量への依存度が高く属人性が高い「主観的」なもの、第三者の納得性の高い「客観的」なものという横軸と、開発の世界の外側から評価が可能な「外部的」なもの、開発の実装や体制といった情報に基づく「内部的」なものという縦軸になっています。調達側や第三者評価で使え、かつ、論拠がしっかりしたものという観点では、図の右上のもの程、望ましいということになります。

4.2．ファンクションポイント法

近年、ファンクションポイント法（以降「FP法」と呼びます）の活用はいろいろな局面ですすんできています。FP法そのものは、システムの機能の大きさを表す尺度に過ぎませんが、開発世界の外部から観測可能である点で優れています。図６に示すように、システム全体の外部的な機能性に着目し、システムの規模を、データの集合（内部論理ファイル、外部インタフェースファイル）と、入出力（外部入力、外部出力、外部照会）によって表す方法です。近年、システムをゼロベースで開発することは希ですし、パッケージやコンポーネントを活用して実装していく場合に、規模を開発コード行数（LOC: Line Of Code）で把握するのは難しくなっていまから、FPで規模を把握するのは有効といえます。

FPは、建築でいえば「坪単価」のような活用法が考えられます。システムのFP当たりの開発費、保守・運用費といったものは、経営指標としても有効なものです。次節で述べる不良資産

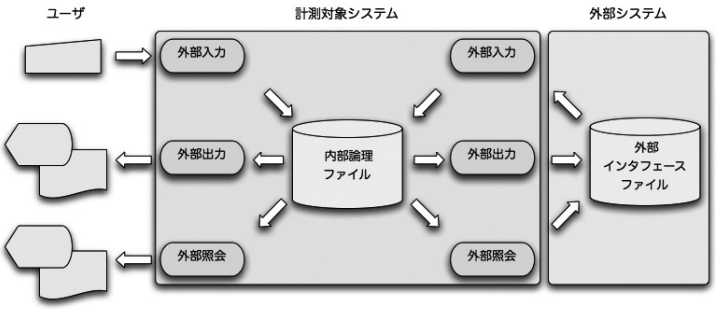


図６　ファンクションポイント法の着目点

の把握についても、ある企業の所有するIT資産が全体でどれだけのFP数で、不良資産化しているものがその内の何%であるといった表現も可能になります。

4.3. COCOMO

図５の一番右側に位置づけられるCOCOMO（COnstructive COst MOdel）は、規模から工数や期間をモデル式によって算出できる方法です。1981年のB.Boehmの著作，”Software Engineering Economics”によって提唱され、以降継続的に研究が進められてきています。図７に基本的な考え方の概要図を示します。

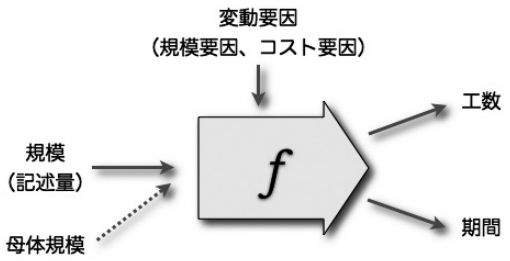


図７　COCOMOの考え方

COCOMOの特長、および、優れている点は、以下の通りです。

- **ソフトウェア開発が人間による「記述」の活動であるという普遍的な考え方に基づいている**
- **記述の規模から、工数、および、開発期間を算出する関数（算術式）を提示している**
- **規模が大きくなるとマネジメントオーバヘッドを生じ、コストが増大することを考慮している**
- **開発に関する変動要因を22種（規模要因５種、コスト要因17種）を定義しており、的確にコスト分析に反映できる**

コスト算出の関数の概要は、以下の通りです。

工数 ＝ A ×（サイズ）$^{\alpha}$ × Πコスト要因
α ＝ B ＋ 0.01 × Σ規模要因
開発期間 ＝ C ×（工数）$^{\beta}$
β ＝ D ＋ 0.2 × 0.01 × Σ規模要因
（A,B,C,Dは定数）

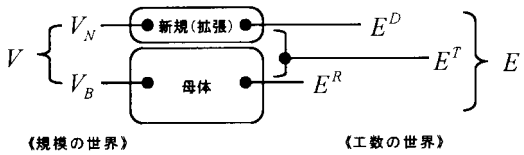
今後のコストモデルを整備していく際に重要になるのが、母体の扱いと時間概念の導入と考えられます。これを以下に簡単に描写しておきます。

V　**規模、**E　**工数、**Q　**品質（コスト要因）、**
P　**期間、**f　**コスト関数**
とした場合、COCOMOは、次のように定式化できます。

$$E = f_E(V,Q)、\quad P = f_P(V,Q)$$

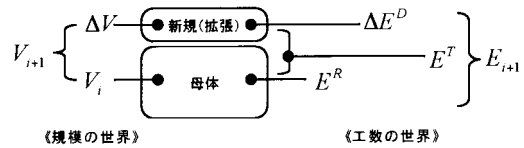
これを母体がある場合に拡張すると、
V_B　**母体規模、**　V_N　**新規（拡張）規模、**
E^R　**理解工数、**　E^D　**開発工数**
E^T　**全体調整（統合テスト）工数、**
とした場合、母体のある場合の定式化は以下のようになります。

$$V = V_B + V_N$$
$$E = E^R + E^D + E^T$$
$$= f_E^R(V_B,Q) + f_E^D(V_N,Q) + f_E^T(V,Q)$$



さらに、インクリメンタル開発プロセスへ拡張してみましょう。
V_i　　**母体規模（第** i **インクリメンタル）**
ΔV　　**新規（拡張）規模**

$$V_{i+1} = V_i + \Delta V$$
$$E_{i+1} = E^R + \Delta E^D + E^T$$
$$= f_E^R(V_i,Q) + f_E^D(\Delta V,Q) + f_E^T(V_i + \Delta V,Q)$$



インクリメンタルの各段階がすすむというのを時間の推移とみなし、連続的な時間 t を導入すると次のようになります。

V_t　**母体規模（時刻 t での母体規模）**
ΔV　**新規（拡張）規模、**　v　**開発速度**

$$V_{t+\Delta t} = V_t + \Delta V \qquad v = V' = \Delta V / \Delta t$$
$$E_{t+\Delta t} = E^R + \Delta E^D + E^T$$
$$= f_E^R(V_t, Q) + f_E^D(\Delta V, Q) + f_E^T(V_t + \Delta V, Q)$$

このような定式化を通して、時間当たりの開発量（生産性）の扱いを、COCOMOの延長線上でとらえることが期待できます。

4.4．費用対効果

費用対効果という視点でも、この時間の概念の導入は有効です。図８は、通常のウォータフォール型の開発の費用と効果の時間推移を表しています。開発に関する資本投下の判断には現在価値評価法（DCF: Discounted Cash Flow）を用いた手法を適用することもできます。一般的に運用フェーズに入ってから保守・改変の費用は初期段階では安定していますが、経年劣化がすすんだ場合や、ビジネス上の要求から改変の比率が増えて費用が増大し臨海点に達していくことが多いようです。

図９は、アジャイル開発等で用いられているインクリメンタル開発を行った場合の時間推移です。小さく始めて、状況を見て、段階的に資本を投下していくことになります。ビジネス上の要求で撤退といった判断もしやすく、こういった判断にはオプション理論等の金融工学の分析手法が適用されることもあります。

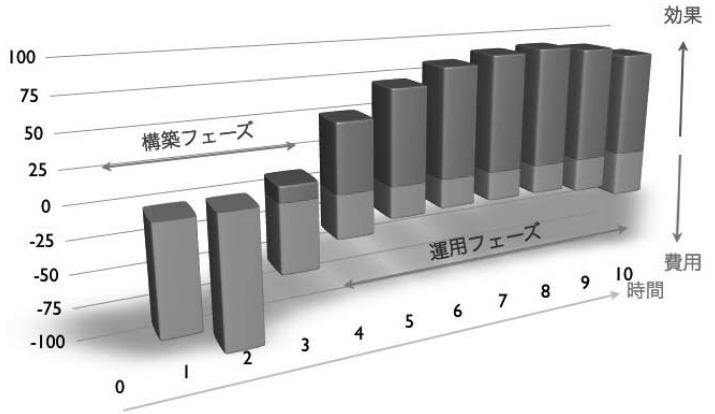


図８　ウォーターフォール型開発の効果と費用

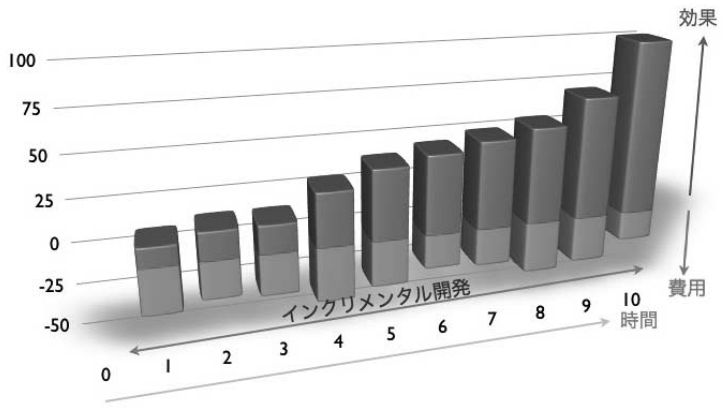


図９　インクリメンタル型開発の効果と費用

５．ソフトウェア資産評価

前節で述べたコストモデルは、個々のプロジェクトに関するもので、仕様が与えられた場合に、これを開発するための工数や期間を求めるものです。一方、組織や企業体という観点では、複数のプロジェクトがあり、定常的に開発活動が推進されています。製造業の工場でもソフトウェア開発は重要な活動で、組織全体での最適化がのぞまれています。特定の市場分野に対してあらかじめ用意された方法を用いて、共通のコア資産から開発される、共通し、管理されている機能を共有するソフトウェア集約型システムは「ソフトウェアプロダクトライン」と呼ばれています。

近年の経営環境においても、ストック型からフロー型へ移行してきており、業務全体の中からボトルネックを見出し改善する制約条件理論（TOC: Theory Of Constraint）も注目を集めています。

このような状況下でソフトウェア資産を把握し、管理することは経済的・経営的な視点でも重要です。衝撃的なデータですが、日本の企業のIT資産の内、約１／３が不良資産化しており、年間経費面でもその半分が無駄な費用であるという統計データも報告されてい

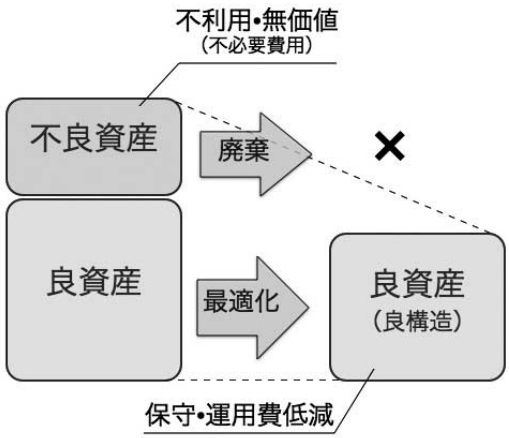


図10　ソフトウェア資産の対策

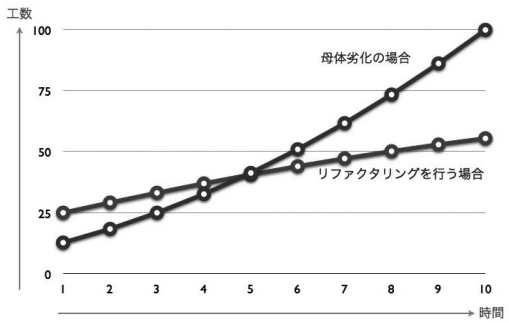


図11　母体改変のコスト経緯

ます。さらに、いくつかの解析の結果、プログラムコードのクローン率（コピー/重複）は1／4程度に達していることが多いようです。

不良資産を破棄し、保持しているソフトウェア資産を良構造に保っておくことは、基本的な対策の一つといえるでしょう。図10に、この対策の概要を示します。不良資産を特定し破棄するためには、システムやアーキテクチャを分析し、利用状況、活用状況を把握しておく必要があります。また、良資産に対しても、それをよりよい構造に変換するには、リファクタリング、部分的な再構成・再設計等の対策をとっていく必要があるでしょう。

図11は、初期母体量に対して毎年20%の追加・改変を行う場合の工数を、そのまま放置して母体構造が劣化していく場合と、リファクタリングによって母体量も1割程度低減して全体構造を良構造に保ち続けた場合の比較をCOCOMO型のコストモデルでシミュレーションしたものです。初段ではリファクタリングの工数が上積みされますが、途中で逆転しますし、工数増大のカーブも前者が累乗的に増加するのに対し、逆のカーブで工数が爆発しないのが見てとれます。

6．ソースコード解析とクリーニング

ソフトウェアの本質的困難の一つである「不可視性」は非常にやっかいな課題です。前節で述べたソフトウェア資産も、多くの企業では全貌を的確に把握をしていないようです。まして、その不良資産化状況や、構造の良し悪しを把握するのは困難を極めています。

こういった課題解決のきっかけになるものとして、まずソースコードを対象とした解析が考えられます。表12に、標準的な品質指標の中か

表12　ソースコードを対象とした品質指標

内部品質特性		ソースコード指標	
特性名	特性の説明	典型的指標	説明
一貫性	設計,製造の技法や表記法,用語などが統一されていること	表現順守率	コーディング規則などで規定される命名規則や文法に従っている文の比率
自己記述性	機能および機能間の関連が完結していること	コメント率	ヘッダー部の仕様記述に関する記述のコメント文の比率
データ共通性	データを内外のシステムと共通に使用できること	データ参照・被参照数	共通データの数,およびデータへの参照数
通信手順共通性	通信手順やインタフェースが共通化していること	モジュール参照・被参照数	通信やインタフェース・モジュールに対する参照・被参照数
アクセス可能性	プログラムの機能や関連装置を選択して自由に使用できること	特定モジュール参照数	プログラム機能や装置に関するモジュールの参照数
アクセス制御性	ソフトウエアやデータへのアクセスを制御できること	特定モジュール参照数	アクセス制御モジュールに対する参照数
アクセス監査性	ソフトウエアやデータへのアクセス記録を残せること	特定モジュール参照数	アクセス記録モジュールに対する参照数
堅固性	誤って操作しても,データやプログラムが破壊されないこと	前提条件充足率	入出力モジュールに対する前提条件の充足率
整合性	異常が発生してもデータやプログラムが破壊されないこと	実行条件充足率	モジュール全体に対する前提条件の充足率
モジュール性	ソフトウエアが構造化され,変更・修正などが局所的に済むこと	結合度	モジュール間の依存の度合い
単純性	仕様の実現方法が簡単であること	規模,複雑度	規模はコード行数（LOC）とHalsteadのメトリクス,複雑度はMcCabeのメトリクス
計測性	プログラムの動作状況を観察,観測できること	特定モジュール参照数	観測するモジュールに関するモジュール参照数
自己包含性	他のプログラムに依存しないで機能を満たせること	凝集度	モジュールが機能,情報を一元的にカプセル化している度合い
統一性	意味,表現,手順が一義的,同一的であること	表現順守率	コーディング規則などで規定される命名規則や文法に従っている文の比率
簡潔性	表現が短く,明解なこと	不要コード率,クローン率	論理的に実行されないか,利用条件によって実行されないコード行数率、コピー/重複のある比率
動的効率性	動作の応答時間,処理時間,スループットがよいこと	ダイナミックステップ数	実行時に実行されるステップ数
資源使用性	実行する際に,使用する資源量や時間が少なくて済むこと	メモリー占有量	実行時に実行されるローディングされるモジュール,データの占有量
拡張性	仕様の追加,変更に対して,容易に対応できること	結合度,凝集度	モジュール性と自己包含性を参照
ソフトウエアシステム独立性	特定のOS,コンパイラなどに依存しないこと	OS非依存モジュール率	OSモジュールを参照しないモジュール比率
マシン独立性	特定の機種,装置,端末などに依存しないこと	機種非依存モジュール率	機種依存モジュールを参照しないモジュール比率
データ独立性	特定のデータ,データベース管理システムなどに依存しないこと	DB非依存モジュール率	DBモジュールを参照しないモジュール比率
伝達性	プログラムの入出力形式や内容が使いやすく統一されていること	メッセージ表現順守率	メッセージに関する規則に順守しているメッセージ定義の比率

らソースコード解析によって把握可能であるものの一覧を掲げています。実際に、モジュール性（結合度）、簡潔性（クローン率等）は、４．３節で述べたコスト分析手法のコスト要因として活用することができます。

こういった客観的な指標値は、リファクタリング（ソフトウェアの機能や振舞いを保持しながら良構造に変換すること）や再構成・再設計の効果を予測・評価するのに活用できるばかりでなく、組織外へのアウトソーシングの管理指標としても利用することができます。図13に、一連のソースコードクリーニングのプロセスを示します。

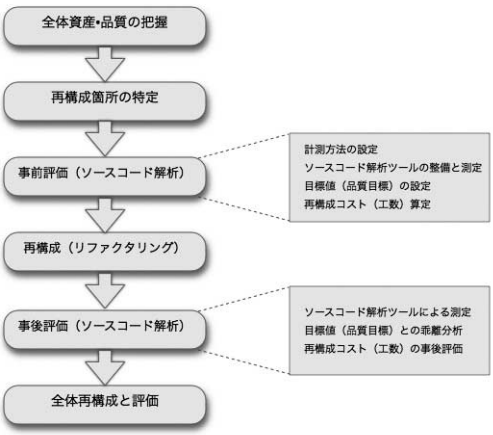


図13　ソースコード解析と再構成手順

ソフトウェア開発・保守・運用に対する施策は、上記のようなライフサイクル的な視点を含め、大きく以下の３点に集約されます。

- **良構造化：不良資産を破棄し、ソフトウェア資産（母体）を良構造に変換・保持すること**
- **自動化：ツール活用やプロセスの最適化によって、誤りを減らし、作業を効率化すること**
- **抽象化：人間の知的活動に使う言語や開発環境の抽象度を上げること**

７．おわりに

本論文では、『ソフトウェア経済学』の構想を簡潔に述べました。まだ着手したばかりであり、今後の検討すべき事項も山積しています。いくつかの方向性を以下に掲げておきます。

- **産業論・組織論で提唱されている「モジュール化」は、この分野の検討に有効であると考えています。複合的企業間連携、アウトソーシング、組織統廃合、さらには社会制度との関連も分析を進める予定です。**
- **企業価値算定、金融工学の知見の適用。特に、市場での価格決定の理論や、リスクの取り扱いについて整備していきます。**
- **バランススコアカード、企業戦略論は、ソフトウェアに関連した組織体では、経営層から実務層にいたる一貫した指針を与え、経済合理的な判断を促すという意味でも活用できると考えています。**
- **アジャイルプロセス、ソフトウェアセル生産、ソフトウェアプロダクトライン、ソフトウェアファクトリーズといった新しい開発パラダイムについても、費用対効果、コスト、市場価格等の観点でその有効性を説明できる論拠を『ソフトウェア経済学』が与えられるものと確信しています。**

《謝辞》

本検討を推進するにあたり、議論していただいた多くの方々に感謝いたします。特に、アジャイルプロセス協議会の見積り・契約WG、アジャイル・ソフトウェアセル生産WG、アジャイルTOC-WGの方々、IPA（情報処理推進機構）/SEC（ソフトウェアエンジニアリングセンター）、および、同見積り手法部会の方々、電子情報技術産業協会のソフトウェアエンジニアリング技術専門委員会の方々、各府省CIO補佐官の方々には多くの示唆を富むご意見をいただきました。また、本論文を執筆する機会を与えていただいたCATS社の渡辺政彦副社長に改めて感謝の意を表します。