ソフトウェア経済学のすすめ
－価値指向のソフトウェア開発アプローチ－

株式会社 一（いち）　副社長／専任コンサルタント
CATS先端研究所（CAL）技術顧問
大槻　繁

はじめに

ソフトウェア開発でがんばったから、残業いっぱいしたからお金ちょうだいという時代は終焉しつつあります。社会への貢献、インパクトの大きさ、効果のすばらしさといった《価値》で、仕事を測り、客観的に評価することができなくてはなりません。『ソフトウェア経済学』は、ソフトウェアに携わるあらゆる人々の幸福達成のための理論や手法を構築していこうという、新しいムーブメントです。

筆者は、システムのコスト分析を仕事としており、専門家として数多くのプロジェクトの見積り評価を手がけてきた経験から、新しい理論開発の必要性を心より切望するようになりました。自らの仕事の価値を高めるためには、価値そのものが何かをしっかりと問うことから始めなくてはなりません。

１．価値を起点に考える

ソフトウェア経済学のアプローチ

ソフトウェアづくりに関するいろいろな現象を解明し、それに携わる人々が幸せになるためには、ソフトウェアエンジニアリングを経済学や経営学の観点をとりいれて総合的に見直す必要があるというのが、『ソフトウェア経済学』の基本的な立場です。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。

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。
IPA/SEC見積手法部会委員、同 ソフトウェア価値評価WGリーダ、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分科会委員、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

- **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**

本論文では、三番目の項目である価値（value）、価格（price）、費用（cost）について論じてみようと思います。特に、「価値」は、概念そのものの定義も難しいですし、それを測ることはもっと大変です。そうでありながら、価値を起点にソフトウェアをとらえるということがビジネス上、経済活動上もとても重要なのです。

過激ないい方をすれば、今までの伝統的な開発方法では、ソフトウェアの仕様を決めて、その仕様を満たすソフトウェアがどのような価値を生むのかということについては、思考停止してきました。世の中の請負受託開発においても、開発者側は、仕様はユーザ側からの要求をまとめた合意事項であり、前提として与えられるものであって、仕様さえ固まってしまえば、それを満たすソフトウェアを開発することに注力すればよいという立場で済みました。ところが、昨今の競争の激化、技術革新の早さ、ビジネス環境の変動はとても早く、仕様を確定しても次々に変更を余儀なくされ、旧来の伝統的な開発プロセスでは対応しきれなくなっています。

２．開発技術の価値

COCOMO再考

まず、開発者の世界での価値から考えてみましょう。同じものを作るのに、ソフトウェア開発を効率化して費用を削減できたとしたら、それは開発者にとっての価値といえます。従来の

請負受託開発で、仕様が確定したら、後はどれだけ費用を抑えて開発できるかが、開発企業にとっての戦略になりますし、収益を生み出す源泉にもなります。

ソフトウェア開発の世界では、開発費用の大部分は人件費です。人が一定の期間、開発に従事することになるので、これを「人月」で定量化し、予測やマネジメントに活用してきた歴史があります。どれくらいの規模で、どれくらいの難しさだったら何人月、何カ月かかるかといったプロジェクトデータに基づく分析が行われ、その代表的なものが南カリフォルニア大学のバリー・ベーム（Barry Boehm）教授の提唱したCOCOMO（COnstructive COst MOdel）です。

30年近いデータ蓄積、モデル（計算式）の改訂等の研究がなされ、今でも新しい開発プロセスに適合した方法も勢力的に検討されています。図１に、COCOMOの概念と、2000年版であるCOCOMO II.2000の具体的な計算式を示します。

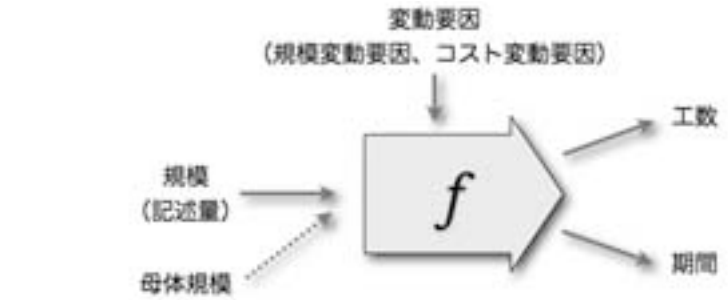
簡単な計算をしてみましょう。C++でもJavaでもかまいませんが、とにかく１万行のコードを書かなくてはならないとしましょう。プロジェクトの制約やソフトウェアの難しさなどは一切捨象してしまい、一般的で標準的なものを開発すると仮定すると、図１の計算式に従い、瞬時に37人月で11.6カ月かかると算出することができます。

COCOMOがマネジメント手法や開発技術の発展が急速に進む中で、長年にわたって多くの場面に適用可能であるのは、ソフトウェア開発の活動の基本が、人間の「記述」であるという普遍的な考え方に基づいているからです。つまり、どんな言語が与えられたとしても、人間の知的生産性というのは変わらないだろうということです。このある種、割り切った見方がCOCOMOのすばらしいところだといえます。

プログラミング言語の抽象度

COCOMOの計算式の入力がプログラミング言語の記述コード行数であるというのは、とても興味深い観点を与えてくれています。システムのインタフェースは、プログラミング言語とは独立ですから、システムの規模をインタフェースから推定したものと、それを実装するプログラミング言語の記述量という規模との間の関係とはどのようになっているのでしょうか。

具体的にインタフェースの規模としてよく使われているファンクションポイント（以降FPと略します）と、それを実現するためのプログラミング言語ごとの記述コード行数との関係を表１



COCOMO II 2000 のモデル式

$$\text{工数（人月）} = 2.94 \times \text{規模（KSLOC）}^{B} \times \prod_i \alpha_i \qquad \alpha_i：\text{コスト変動要因}$$

$$\text{期間（月）} = 3.67 \times \text{工数（人月）}^{F} \qquad \alpha_i \text{ の標準値は } 1.00$$

$$B = 0.91 + 0.01 \times \sum_j \beta_j \qquad \beta_j：\text{規模変動要因}$$

$$F = 0.28 + 0.2 \times (B - 0.91) \qquad \beta_j \text{ の標準値の合計 } \sum_j \beta_j \text{ は } 18.97$$

図１　COCOMO（COnstructive COst MOdel）

に示します。同じ1FPがアセンブラ言語で320行、Cobol （ANSI 85）で91行、Javaで53行といった対比になります。つまり、表1はプログラミング言語の「抽象度」を表しているといえます。高級な言語ほど少ない記述行数で所望の機能を実現できるということになります。

再度、ちょっとした計算をしてみましょう。100FPの仕様があったとして、これをいろいろな言語で実装することを想定します。アセンブラ言語だと100FP×320行/FP = 32,000行、Cobolだと100FP×91行/FP = 9,100行、Javaだと100FP×53行/FP = 5,300行になります。これを図1のCOCOMOの式で工数と期間とを計算すると、それぞれ、132.9人月（17.4カ月）、33.3人月（11.2カ月）、18.4人月（9.3カ月）となります。

もし、ある開発企業が1FPを1行で実装できる超高級言語阿吽（ア・ウン：仮の名前です）をもっているとしたら、0.2人月（2.3カ月）で開発ができ、同業他社がJavaで開発しているとしても1/100程度のコストで、かつ、短納期で対応できることになります。

つまり、言語の高級化というのは、開発企業にとっての重要な戦略の一つとなるでしょうし、現にDSL（Domain Specific Language）が注目されている理由も、対象領域を絞りながらも、記述量を減らすことによる経済効果が大きいからとみなすことができるでしょう。

コスト要因の示唆するもの

COCOMOのコストモデルで、規模以外については、表2に示すコスト変動要因（cost driver）と、表3に示す規模変動要因（scale factor）によって調整することができます。

前者は大きな開発ほど、より大きなオーバヘッドがあって伝統的な開発手法では累乗で効いてくるので、その曲線の反り具合の形状を決めるものです。後者は各プロジェクトの工数に比率として効いてくるものです。

例えば、コスト変動要因の人的要因の分析者の能力（ACAP: Analyst Capability）とプログラマの能力（PCAP: Programmer Capability）について考えてみましょう。標準値的な分析者、プログラマが携わる場合にはコスト計算はそのままですが、世の中全体の上位10%以内に入るような優れた分析者の場合には0.71倍、同様に上位10%以内に入るような優れたプログラマの場合には0.76倍です。これは全体の工数にかかる比率です。先ほどの1万行のコード記述の計算例でいうと、分析者とプログラマ両者が優良の場合には、37人月×0.71×0.76 =19.96人月と標準的な人々を集めて作った場合の約半分で済むことになります。人材の採用や育成が、開発企業にとっていかに大切かがわかると思います。

SLOC:ソースコード行数、
UFP:未調整ファンクションポイント

プログラミング言語	SLOC/UFP
アセンブラ	320
C	128
C++	55
Cobol(ANSI 85)	91
データベース	40
Java	53
Visual C++	34

表1　プログラミング言語の抽象度

もう一つ具体的な計算をしてみましょう。よくCMM （Capability Maturity Model）のレベルを表明する開発企業がありますが、その効果がCOCOMOでどのように分析されるかをみてみましょう。規模変動要因にプロセス成熟度（PMAT）が、CMMレベル1とレベル5は、それぞれ最低値7.8と、最高値0.0と解釈してみると、1万行の開発の場合で、それぞれ39.7人月と33.2人月、10万行の開発の場合で、それぞれ537.2人月と375.1人月となります。つまり、CMMレベルの差異で2～3割の工数削減になります。規模変動要因という名前からもわかるように、これは開発規模が大きくなればなるほど、比率は広がっていきます。開発企業が真にプロセス成熟度を上げるように投資をすることによって、生産性という観点だけでも数十パーセントの向上が見込まれることになります。しかも大規模なプロジェクト程、効果が高いということも予測することができます。

その他にもツールの整備状況、チーム強度、

α_i：コスト変動要因 (Cost Driver)

	コスト変動要因	略号	値の範囲
プロダクト要因	信頼性要求度	RELY	0.82 ～ 1.26
	データベースサイズ	DATA	0.90 ～ 1.28
	プロダクトの複雑度	CPLX	0.73 ～ 1.74
	再利用性	RUSE	0.95 ～ 1.24
	文書化	DOCU	0.81 ～ 1.23
プラットフォーム要因	実行性能制約	TIME	1.00 ～ 1.63
	メモリ制約	STOR	1.00 ～ 1.46
	プラットフォーム安定性	PVOL	0.87 ～ 1.30
人的要因	分析者の能力	ACAP	1.42 ～ 0.71
	プログラマの能力	PCAP	1.34 ～ 0.76
	アプリケーションの経験	APEX	1.22 ～ 0.81
	プラットフォームの経験	PLEX	1.19 ～ 0.85
	言語／ツール経験	LTEX	1.20 ～ 0.84
	定着率	PCON	1.29 ～ 0.81
プロジェクト要因	ツール充実度	TOOL	1.17 ～ 0.78
	コミュニケーション	SITE	1.22 ～ 0.80
	スケジュール	SCED	1.43 ～ 1.00

表 2　COCOMOのコスト変動要因

β_j：規模変動要因 (Scale Factor)

規模変動要因	略号	値の範囲
問題領域経験度	PREC	6.20 ～ 3.72 ～ 0
プロセス柔軟性	FLEX	5.07 ～ 3.04 ～ 0
リスク管理レベル	RESL	7.07 ～ 4.24 ～ 0
チーム強度	TEAM	5.48 ～ 3.29 ～ 0
プロセス成熟度	PMAT	7.80 ～ 4.68 ～ 0

表 3　COCOMOの規模変動要因

リスク対応のレベルなど多くの要因をCOCOMOのモデルを使って具体的に経済効果を算定することができます。開発企業で、技術力向上や人材育成は、経営戦略上重要な意思決定になりますが、どういう施策をうてば、どれくらいの効果が上がるかということをCOCOMOは教えてくれています。最近のはやり言葉でいえば、経営戦略を実現するためのアクションプラン設定時の重要業績評価指標（KPI: Key Performance Indicator）の論拠を与えることができるといえます。

3．価値を考える観点

価値ベースソフトウェアエンジニアリング

バリー・ベーム教授は、2003年に価値ベースソフトウェアエンジニアリング（Value-Based Software Engineering）という考え方を提唱し、2006年に同名の書籍（ISBN-10 3-540-25993-7 Springer）も出版しています。ソフトウェアエンジニアリングの諸活動を、それを取り巻く周辺世界を含めてきちんと位置づけようというものです。７つの観点をあげています。

①便益実現分析
（Benefits-Realization Analysis）
②ステークホールダ価値抽出・調整
（ Stakeholder Value Proposition Elicitation and Reconciliation）
③ビジネスケース分析
（Business Case Analysis）
④継続的リスク･チャンスマネジメント
（Continuous Risk and Opportunity Management）
⑤システム･ソフトウェア並行エンジニアリング
（Concurrent System and Software Engineering）

⑥価値ベース監視・制御
（Value-Based Monitoring and Control）
⑦変化というチャンス
（Change as Opportunity）

ご利益（りやく）の追跡

第1は、便益実現分析（Benefits- Realization Analysis）です（図2）。実現手段や実装方法が、真に目的とする成果を出せるかどうかを整理する必要があります。システムを使うために新たなオーバヘッドが発生して、かえって利用者の業務を阻害してしまうといったことがないように、成果の因果関係（result chain）をステークホールダ間で調整しなくてはなりません。

例えば、営業支援システムを検討する場合には、営業担当者の注文処理業務を効率化したいのか、販売量を多くしていきたいのかといった事項で、それぞれ評価の観点も変わってきます。これを単にオーダエントリの仕様を決めるところで思考停止してしまうと、使われないシステムを作りかねません。

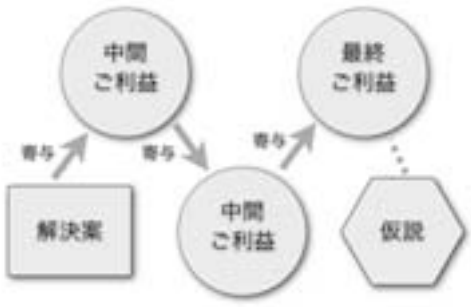


図2　ご利益（りやく）の追跡

ステークホールダ間の調整

第2は、ステークホールダ価値抽出・調整（Stakeholder Value Proposition Elicitation and Reconciliation）です（図3）。ソフトウェア開発におけるステークホールダ（利害関与者）とは、開発の意思決定に重要な影響を及ぼす人々という意味です。

代表的なステークホールダは、ユーザ、調達者、開発者、運用者でしょう。ユーザは欲張りな機能を早く欲しがります。調達者は予算を絞ってしっかりと契約で縛りたいと考えます。開発者は安定した仕様と緩い契約を要望するでしょう。運用者は新システムへの移行や維持管理の容易性を求めます。各ステークホールダの要望を明確に抽出し、かつ、多くの事項はステークホールダ間で競合するので調整も必要になります。

ユーザの要求を吸い上げ、合意形成を行うためにプロトタイピングを行うといったことは、ステークホールダ調整の一つの手法といえます。

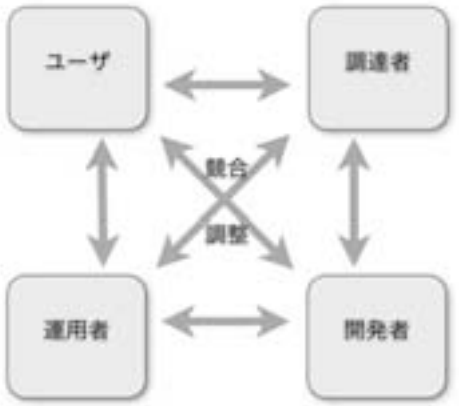


図3　ステークホールダ間の調整

ビジネス戦略の検討

第3は、ビジネスケース分析（Business Case Analysis）です。ソフトウェアを開発してからリリース、運用する際のライフサイクルの観点で投資効果（ROI: Return On Investment）を分析する必要があります。

$$ROI = （リターン - 投資）／投資$$

です。

早めにそこそこのものを市場投入して、そこそこのリターンを得るパターン、じっくりと高度なものを市場投入して大きいリターンを得るパターンといった、いくつかの場合を想定して分析を進める必要があるでしょう（図4）。競合他社がどういうタイミングで市場投入してくるかといった不確定要素も盛り込む必要があるかもしれません。

定量化できない事項も分析する必要があります。一早く市場へ製品を出すことによって、他社を牽制したり、製品のシェアを確保したいといったこととか、ゆっくり二番煎じでリスクを

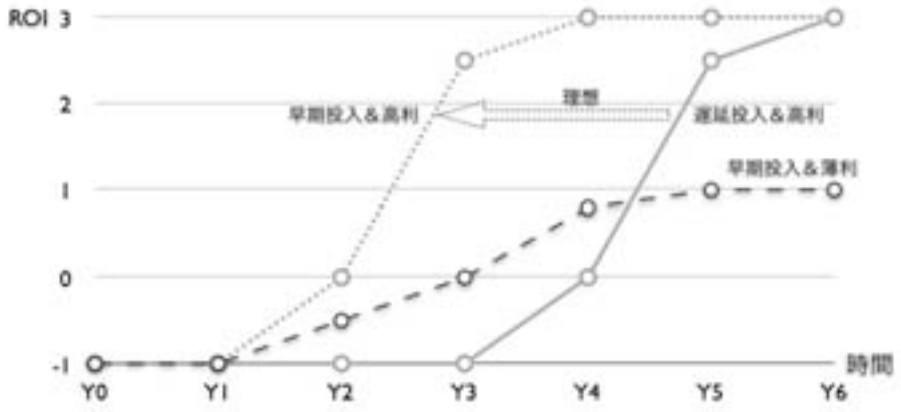


図4　ビジネス戦略の検討

低減して品質で勝負したいといったことなども経営戦略上の意思決定事項になります。

リスクとチャンス

第4は、継続的リスク・チャンスマネジメント（Continuous Risk and Opportunity Management）です。システムの計画初段での投資効果分析のみならず、プロジェクト進行中もリスクやチャンス（オポチュニティ）の分析を行う必要があります。開発者の志向性等も配慮し、リスク回避型なのか挑戦型なのかといった把握もしなくてはならないでしょう。

また、ベーム教授が書籍「アジャイルと規律」（日経BP社, ISBN4-8222-8192-2, 2004年）で述べているように、あらかじめ計画をしっかりとたてておく計画駆動型と、逆に柔軟に対応していくアジャイルプロセス型とがあって、リスクエクスポージャ（RE: リスク損失）を最適にするような計画を行うための工数を設定（スィートスポットと呼びます）すべきであるといわれています（図5）。

新しい開発プロセス

第5は、システム・ソフトウェア並行エンジニアリング（Concurrent System and Software Engineering）です。従来のウォータフォール型の順序型の開発ではなく、アジャイルプロセスに代表される並行型の開発が台頭してきています。これはシステムエンジニア、マーケッティング担当者、アーキテクト、開発者等が同時並行的に作業を進めていくものです。価値をベースにするということは、適用領域の情況やビジネス変化が激しい場合には、これに同期した開発プロセスを採用していく必要があります（図6）。

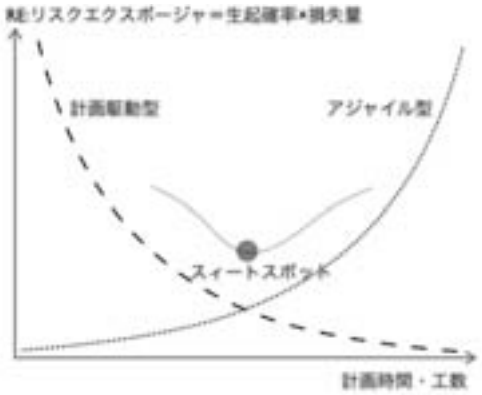


図5　リスクとチャンス

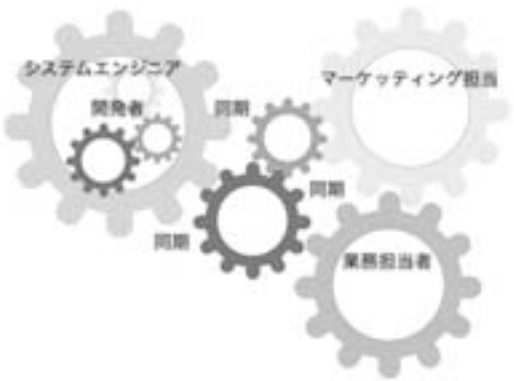


図6　新しい開発プロセス

価値の監視・制御

第6は、価値ベース監視・制御（Value-Based Monitoring and Control）です。プロジェクトの監視・制御の一般的方法としては、アーンド・バリュー法（Earned Value Management）として知られています。これは、あらかじめ仕様が確定している場合に、開発過程の情況を出来高で把握しようという発想です。ただし、この手法は作業が詳細に決めることができる建築現場や、ひたすら画面定義やコンテンツを整備していくような定形型の業務には向いています。すなわち、上記前第4項で述べた「計画駆動型」の場合には適用できます。

一方、仕様が変更されたり、挑戦的な技術開発がある場合には適しません。しかし、こういった監視・制御の仕組みをシステムの本来の目的や価値にまで範囲を広げてマネジメントを進めていくことは有効です（図7）。

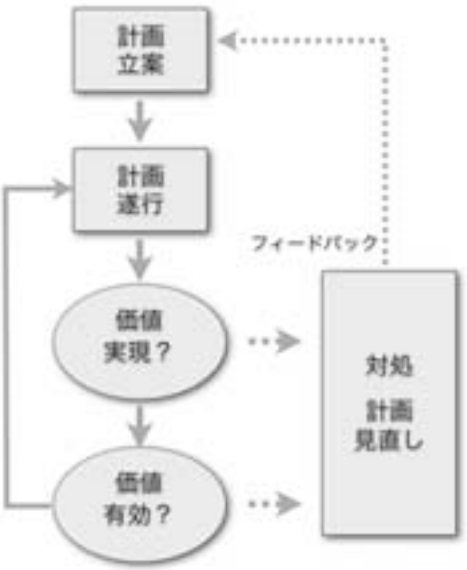


図7　価値の監視・制御

変化への対応

第7は、変化というチャンス（Change as Opportunity）です。変化への対応は、システムの価値を上げる源泉でもあります。インターネット、GPS（Global Positioning System）などの技術革新や普及によって、従来のシステムも劇的な変化を遂げています。ソフトウェアエンジニアリングの観点からも、独立性の高いモジュール化を進めたり、アジャイルプロセスといった変化そのものに正面から取組む開発方法が現れてきています。

4．価値評価手法の概観

ソフトウェア以外の価値評価手法

価値を評価する手法が最も整備されているのは企業価値評価の分野です。昨今の企業合弁買収（M&A: Mergers and Acquisitions）が話題になっていることからもわかりますが、企業そのものも、「もの」と同様に売り買いができますし、値段もつけられる世の中になっています。

企業活動は損益計算書、貸借対照表、キャッシュフロー計算書といったいわゆる財務諸表としてお金の面は把握できますから、株価算定を含めていろいろな方法が開発されています。表4はその概要をまとめたものです。

特に事業価値を評価する手法としてのDCF（ディスカウントキャッシュフロー）法が、システムやソフトウェアの評価への展開可能性という意味で有望です。お金にまつわる評価をしなくてはならない場合には、こういった一般的な手法をソフトウェアの世界に適用していくことも検討していくとよいでしょう。

ソフトウェアの価値評価手法

ソフトウェアの価値評価の難しさは、いわゆるフレデリック・ブルックスジュニアのいう「ソフトウェアの本質的困難」をそのまま引きずっています。大きくて複雑であること、社会や周囲技術との順応性があること、見えないこと、どんどん変化・変貌することです。

価値そのものの定義も難しいですし、万人が納得するような合意点も見出しにくいでしょう。価値そのものの定量化も難しいですし、そして、たとえシステムが変わらなくても（無論、どんどん変わるのですが）、価値が人によっても、時と場合によっても変化します。とりあえずはお金にまつわる投資やリターン、事業戦略等にかかわるところだけでも分析するとしたら、表5のような整理になります。

表5の特徴の欄に記載されているように、上から下へだんだん分析が難しくなっていきます。つまり、複数の事業シナリオへの考慮、確率分布の導入、不確実性の検討、企業全体の最適化

といった事項を評価手法の中に取り込んでいく必要があり、それぞれ難しい問題を含んでいます。さらに最近では、人間の意思決定に関する行動パターンまで織り込むことなども検討されています。

評価手法	説明	特徴
純資産法	貸借対照表の資産から負債を差し引く方法で、簿価をそのまま適用する簿価純資産法と、時価を評価して適用する時価純資産法とがある。	キャッシュフローを考慮していない
収益還元法	企業が将来生み出す（会計上の）純利益を予想し、現在価値に割り戻して算定する手法	設備投資、資本増減を考慮していない
DCF法	事業の投資に対する将来のキャッシュフローを予測し、割引率を設定し、現在価値に割り戻し、事業全体の投資価値を推定する手法	成長シナリオ（事業計画）に基づくキャッシュフローを分析可能
配当還元法	株主配当を現在価値に割り戻して算定する手法	配当配分方針に依存してしまう
類推法	類似した会社、業種から推定する手法	類似性の判定が恣意的になる
株価指標法	PER（株価収益率）、PBR（株価純資産倍率）、PCFT（株価キャッシュフロー倍率）、EV/EBITDA、配当利回り、PSR（株価売上高倍率）等の指標から推定する手法	株主価値の観点 市場での企業間比較に活用

表 4　一般の価値評価手法

評価手法	説明	特徴
DCF法 (Discounted Cash Flow)	事業の投資に対する将来のキャッシュフローを予測し、割引率を設定し、現在価値に割り戻し、事業全体の投資価値を推定する手法	単一シナリオ 確定的推定
意思決定木 (Decision Tree)	意思決定（戦略）のシナリオを決め、選択肢を確率分布によって分析し、全体の事業価値を推定する手法	動的で柔軟なシナリオ 確率的推定
リアルオプション (Real Option)	金融のオプション理論（ブラック・ショールズ）を事業（プロジェクト）に適用する手法で、意思決定の延期、拡大、縮小、中止、段階的推進等のオプションを扱う手法	不確実性の分析
ポートフォリオ (Portfolio)	複数の事業（プロジェクト）間の協調・競合関係を分析し、組織資源の最適配置を考慮し、全体の価値を推定する手法	全体最適（企業戦略）

表 5　ソフトウェアの価値評価手法

DCF法

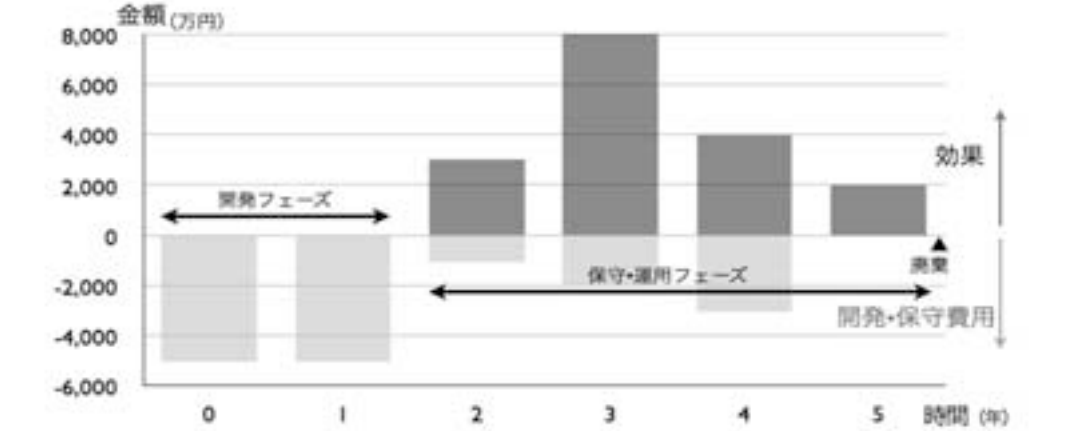
DCF法は、事業が生み出す将来のキャッシュフローを現在価値に割引いたものの総和を事業価値とみなす手法です。考え方としては、将来手に入るキャッシュを、今（現在）得ようとしたら金利分（複利計算）を差し引いたものに相当するということを適用して、将来という時間を全て現在の価値に割引いて総和を求めるものです。

図８の下に計算式を示したようになります。割引率rは、資本コストを用います。資本コストは、企業が資金調達をする際の有利子負債と株主資本コストとから、企業内の両者の加重平均によって求められます。有利子負債は銀行との契約等によって定められますが、株主資本コストは、株主がどのようにリスクを予想し、どれだけのリターンを求めるかによるため、その算定法は複雑なものになっています。

DCF法の算定例を図８に示します。初期投資としてまず5,000万円投下し、続いて次年度（第１期）も同額投下、つまり、まず最初は開発に専念するというパターンです。第２期にリリースしてそこで効果（リターン）を得ることができます。第３期に効果が最大に達し、以降半減していくというシナリオです。リリース後の費用についても第２期が1,000万円で、年次を追って増加して第４期に3,000万円に達し、第５期の最後に寿命を迎えて破棄されるというものです。よくありそうなシステムのライフサイクルです。

全ライフサイクル中の投下額の総額はキャッシュ・アウトフローとして16,000万円、システムが生み出す効果はキャッシュ・インフローとして17,000万円です。単純な計算をすれば、1000万円の価値があるように見えますが、これをDCF法の計算式で割引いて計算すると、図７の表の下２段にあるようになり、正味現在価値NPVは、375万円の赤字になります。

このように、DCF法は、時間軸というものを割引率という形で集約して価値を計算する手法として有効です。



効果			3,000	8,000	4,000	2,000	17,000
費用	5,000	5,000	1,000	2,000	3,000		16,000
割引効果			2,721	6,911	3,291	1,567	14,490
割引費用	5,000	4,762	907	1,728	2,468		14,865

割引率を$r = 0.05$と仮定　効果 − 費用 = 1,000　（万円）
NPV = 割引効果 − 割引費用 = △375

$$NPV = \sum_{t=0}^{N} \frac{CIF_t - COF_t}{(1+r)^t} + \frac{TV_N}{(1+r)^N}$$

NPV：正味現在価値　CIF：キャッシュ・インフロー　COF：キャッシュ・アウトフロー
TV：投資の終価　t：時間（期）　N：最終期　r：割引率

図８　DCFの評価例

意思決定木法

価値を評価することの難しさは、将来を予測しなくてはならないことです。物事がどうなるかは、自分自身で全てを決められるわけではありません。情況に応じて手を打ち、その結果を受けて、次の手を打つということを繰り返していきます。この場合分けを記述するために木構造を使ったものが意思決定木です。

図9にその考え方の概要を示します。木のノードが時間とともに展開されていきます。アクション（例えば技術開発）を行った場合に、その結果を評価して意思決定を行います。結果がどうなるかは確率として予測します。意思決定というのは、前段の結果を受けてさらに投資をするとか、中止をするといった決断を意味します。時間が左から右に流れていきますから、この各ノードにそれぞれの場合の効果や費用を割り当てていけば、それを集約してプロジェクト全体の効果や費用も算定することができます。時間が経過することを考慮して前段のDCF法を適用して、正味現在価値で評価を行うこともできます。

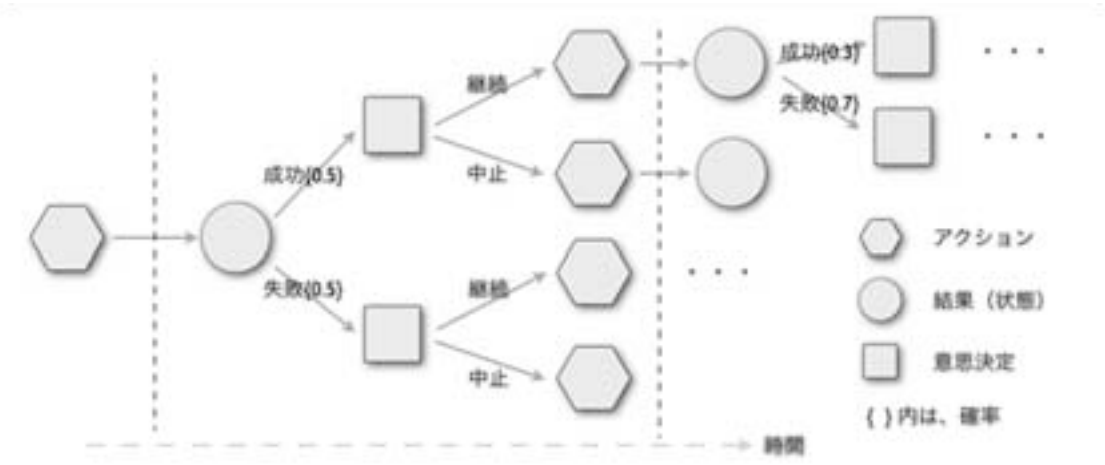


図9　意思決定木の考え方

これからの価値指向アプローチ

本論文では、今までのソフトウェアエンジニアリングのコストモデル、一般の金融関連の領域で使われている評価手法を概観し、新しい時代のソフトウェアづくりに必要な経済・経営的な「価値」の評価方法について紹介しました。

ソフトウェアの本質的特性をとらえた有効な方法を探究していく必要があります。意思決定、戦略、市場、不確実性、人間の心理・社会的な行動等、まだまだ織り込んでいくべき項目がたくさんあります。定量化が難しいモチベーションや取引に現れないような事項も考慮していかなくてはなりません。そしておそらく、未だ語られていないような「価値」を創出することこそが、新世代のソフトウェアづくりなのかもしれません。

謝辞

本書を編集・執筆するにあたり、極めて多くの方々にお世話になりました。そもそも『ソフトウェア経済学』と銘名する以前から、見積り評価やプロジェクト支援のコンサルティングをしながら多くの問題意識や知見を得ることができ、その折にお世話になった方々も数多くいらっしゃいます。『ソフトウェア経済学』という名前での正式の発表は、CATS社のZIPC WATCHERS誌 Vol.10です。その折に執筆の機会を与えていただいたのは同社渡辺政彦副社長です。氏には組込みシステムや製造業への展開の観点を示唆していただきました。電子協(JEITA)、PMI東京、PM学会、EASEプロジェクトの方々にも発表の機会を与えていただく等、ご支援をいただきました。とりわけ二木厚吉教授（北陸先端大）には、本領域のプロモーショ

ンもしていただきました。ここに感謝の意を表します。

参考文献

ソフトウェア経済学に関する初出から一連の発表は以下のようになります。

［1］大槻繁, ソフトウェア経済学の概要, ZIPC WATCHERS Vol.10, CATS社, 2006年9月.
［2］大槻繁, アジャイル・セル生産のためのソフトウェア経済学, アジャイルプロセス協議会/アジャイル・ソフトウェア・セルオープンフォーラム, 2006年10月.
［3］大槻繁, ITプロジェクト見積りの勘と度胸からの脱却：価格・コスト・価値を取り巻く新視点「ソフトウェア経済学」, ソフトウェアPMI東京支部主催プロジェクトマネジメント2006年第8回月例テクニカルセミナー, 2006年11月.
［4］大槻繁, 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, JEITA情報システム技術シンポジウム, 2006年11月.
［5］永井昌子, 大槻繁, システム改善を目的としたクローン分析ツールの適用, ソフトウェア工学工房, 2007年2月.
［6］大槻繁, ソフトウェア経済学：マネジメントのためのコスト･価値･価格の考え方, プロジェクトマネジメント学会2007年度春季研究発表大会キーノート, 2007年3月
［7］大槻繁ほか,ソフトウェア経済学:概要＋アジャイルプロセス＋見積り手法編, エンジニアマインドVol. 5特集, 技術評論社, 2007年7月
［8］大槻繁ほか：ソフトウェアの価値はどこにある：ソフトウェア経済学のすすめ, ソフトウェア技術者サミットin高松, 2007年7月
［9］大槻繁ほか, 続・ソフトウェア経済学:価値指向のソフトウェア開発アプローチ編, エンジニアマインドVol. 6特集, 技術評論社, 2007年9月