ソフトウェア工学 80－12
（１９９１．７．１９）

品質を加味したコスト評価モデルの提案

堀田勝美　逆井義文　浅見秀雄　渡辺　孝　福山峻一

ＮＴＴ　ソフトウェア研究所

ソフトウェアの価値を機能面だけでなく品質面からも評価できるソフトウェア開発コスト評価モデル：ＣＯＳＭＯＳ* を提案する。本モデルは、ソフトウェアの発注者が発注仕様情報だけを元に精度良くソフトウェアのコストを見積れるようにすることを狙いにしている。

本稿では、ソフトウェア品質特性に関するＩＳＯ／ＳＣ７での検討結果を参考に、特に、コストに変動を与える品質特性要因の抽出と発注条件として指定可能な各要因の評価尺度の設定方法を提案して、品質を加味したコスト評価モデルの成立性を議論する。

＊ＣＯＳＭＯＳ：COSt MOdel for Subcontract

Proposal of a Cost Estimation Model considering Software Quality

Katsumi HOTTA, Yoshifumi SAKASAI, Hideo ASAMI,
Takashi WATANABE, and Shun'ichi FUKUYAMA

NTT Software Laboratories,
Nippon Telegraph and Telephone Corporation
1-9-1 Konan, Minato-ku, Tokyo, 108 Japan

We propose a software development cost estimation model, named COSMOS,* which enables to estimate software value on its quality as well as its function. The aim of this model is to enable orderers to estimate software cost accurately with only its order specifications to subcontractors.

In this paper we extract quality factors which affect software cost refering the result of study in ISO/SC7, and propose measurements of the factors specifiable as order conditions. And we discuss the existence of a cost estimation model considering software quality.

*COSMOS: COSt MOdel for Subcontract

1．まえがき

本稿では下記要求を満たすソフトウエア開発コストの評価（着工時のコストの見積りと完了時の生産性評価）モデル：ＣＯＳＭＯＳ* を提案する。

*) COSMOS : COSt MOdel for Subcontract

(1) ソフトウエアの価値を機能面だけでなく品質面の努力も加味した評価ができること。

(2) 発注者が発注仕様に規定した項目だけを用いて評価できること。

ここで、発注者とは要求定義以後の設計／製造／試験といったソフト開発工程の全体または部分を他者（受注者）に依頼する者とする。

従来、ソフトウエアの開発コスト見積りは先ず納入品のソースコード規模：ＤＳＬ（Delivered Source Lines of code）を経験則的に推定し、これから所要工数を推定して積算していた。すなわち、規模に換算可能な機能特性に着目した評価になっていた。このため保守性とか移植性の確保といった品質向上努力の評価が不十分であった。

本稿では特にこの点に注目し、ソフトウエアの品質特性に関するＩＳＯでの検討結果を参考に、コストの変動要因となる品質特性（コスト要因）の選定と発注条件として規定可能な評価尺度の設定を試みる。

規模だけでなく品質面も加味した実用的な評価モデルとしては、Boehm のＣＯＣＯＭＯ（COnstructive COst MOdel）が有名である＜表１＞。しかし、ＣＯＣＯＭＯには開発プロジェクトの運営方法や要員の技能といった受注者側でないと把握しきれないコスト要因があり、発注者側がこれを正確に把握して評価を行うのは困難である。よって、本稿ではＣＯＣＯＭＯの考え方を踏襲しながら、発注者側で規定可能なコスト要因のみを用いて評価できるモデルの確立を試みる。

評価モデルＣＯＳＭＯＳの確立により次の効果が期待できる。すなわち、発注者は、発注仕様中に規定した自身で制御可能なコスト要因の値を用いて、ソフトウエアの価値をより精度高く、客観的かつ定量的に評価可能になり、価値に見合う適正価格でソフトウエアが入手出来るとともに、所望する品質で納品出来る会社への発注が可能になる。また、受注者側としても、要員の技能を高め、環境を整備して品質の向上に努めることにより、より多くの利益をあげることができるであろう。

表１ ＣＯＣＯＭＯ（中間）モデル[1]

$$EMM = C \cdot (KDSI)^{b} \cdot f_1 \cdot f_2 \cdot f_3 \cdots\cdots f_{15}$$

EMM：見積り工数（人月）

C：開発モードによって決まる係数

KDSI：納入ソースコード規模（ｷﾛｽﾃｯﾌﾟ）

f_i：コスト要因の影響度を表す努力係数

属性	コスト要因	努力係数
ソフトウエア製品	ＲＥＬＹ：Required software reliability; ソフトウェアに要求される信頼性	f_1
	ＤＡＴＡ：Data base size; 処理すべきデータベースのサイズ	f_2
	ＣＰＬＸ：Product complexity; ソフトウェア製品の複雑性	f_3
計算機	ＴＩＭＥ：Execution time constraint; システムに課せられた実行時間の制約	f_4
	ＳＴＯＲ：Main storage constrant; 主記憶の制約 〔プログラムのメモリ平均占有率〕	f_5
	ＶＩＲＴ：Virtual machine volatility; ＯＳ，ハードの変更頻度	f_6
	ＴＵＲＮ：Computer turnaround time; プログラム開発時のコンピュータ・ターンアラウンド・タイム	f_7
要員	ＡＣＡＰ：Analyst capability; アナリスト（システム設計者）の能力	f_8
	ＡＥＸＰ：Application experience; 同様なアプリケーションの経験度合	f_9
	ＰＣＡＰ：Programmer capability; プログラマ（プログラム設計以降の担当者）の能力	f_{10}
	ＶＥＸＰ：Virtual machine experience; ＯＳ，ハードの経験度合	f_{11}
	ＬＥＸＰ：Programming language experience; 使用されるプログラミング言語の経験度合	f_{12}
プロジェクト	ＭＯＤＰ：Use of modern programming practices; ソフトウェア開発技法の使用度合	f_{13}
	ＴＯＯＬ：Use of software tools; 使用ソフトウェア・ツールの内容	f_{14}
	ＲＣＥＤ：Required development schedule; 要求される開発期間の制約	f_{15}

2．提案するコスト評価モデル

2．1　コスト評価のメタモデル

ソフトウェアのコストは一般的に、開発する製品の量と単位量当りの工数により評価される。しかし外注の場合、発注者側では工数を正確には把握できないため、直接把握可能な単位量当りのコストを用いた次式で評価する方が、むしろ正確な評価ができる。

$$Y = M \cdot y \qquad \cdots (1)$$

Y：製品コスト
M：製品の量
y：単位量当りのコスト

2．2　製品の量（M）の評価尺度

ソフトウェア製品は一般にプログラムとドキュメントから構成される。従ってソフトウェア製品の量はこれらを合算した量で表わすことが考えられる。しかし、現状ではプログラム量とドキュメント量を同一尺度で扱うための換算方法が無いため、本モデルでは従来通り納入プログラムの規模（ＤＳＬ*）を製品の量として用いる。

$$M = DSL \qquad \cdots (2)$$

しかし、ドキュメントはプログラムの使用性や保守性等の品質に大きく影響を与えるので、本モデルでは、後述する単価要因の評価尺度として捉える。

ＤＳＬとしては表２に示すコードがその計測候補に挙げられる。ここで、コメント文等はドキュメントと同様に単価要因の評価尺度として捉えるのが適当であるため、ＤＳＬの対象から除外する。

ＤＳＬの推定は、ファンクションポイントから推定する方法等いくつかの方法が提案[2]--[8]されており、それらの論文に依存することとするが、ＤＳＬは機能を実現するものとして３章で述べる規模要因に依存してその値が決定されるものであるため、これらを加味した推定方法について検討の余地がある。

*DSL:Delivered Source Lines of code

2．3　単位量当りのコスト（y）の評価尺度

前節により、製品の量の尺度をＤＳＬとすることから、単位量当りのコストはプログラムのステップ当りのコストすなわちステップ単価になる。

本稿では、ステップ単価は３章で述べる各種の品質向上努力の程度によってその値が変動するものとみなす。品質向上努力のうち計測可能（発注時に努力レベルを指定可能）な要因を単価要因（Ｑｉ）と呼び、Ｑｉ毎に評価尺度を定め、その値域に応じて定められる単価への影響度合（努力係数）を q_i とするとステップ単価は次の式で表わされる。

$$y = K \cdot \prod_{i} q_i \qquad \cdots (3)$$

K　：定数（プログラムの種別により定まる標準的なステップ単価）
q_i：単価要因毎の評価尺度の値域に応じて定められる努力係数
i　：努力係数の数

2．4　ＣＯＳＭＯＳ評価式

以上から、ソフトウェア製品のコスト（Ｙ）は次式で表わされる。

$$Y = K \cdot DSL \cdot \prod_{i} q_i \qquad \cdots (4)$$

表２　ＤＳＬとしての計測候補コード

内訳				計測対象コード*
Ⅰ	Ⅱ	Ⅲ	Ⅳ	
納入プログラム	プログラム本体	ソース	データ宣言文	○
			実行文	
			マクロ文	
			コンパイラ制御文	
			デバッグ文	×
			コメント文	×
			空文	
		ＪＣＬ		○
	データ			○
	中間生産物	テスト	単体試験用	×
		プログラム	結合試験用	
		テスト	単体試験用	
		データ	結合試験用	
		ツール	スタブ	×
			ドライバ	
			その他ツール	

（注）○：計測対象　×：計測対象外
（ただし、発注時指定の場合を除く）

3．単価要因の抽出と評価尺度の設定[9]

3．1　単価要因の抽出

ステップ単価に影響を及ぼす単価要因（Ｑｉ）を抽出するため、ＩＳＯ／ＩＥＣ　ＪＴＣ１／ＳＣ７配下のソフトウェア開発・システムの文書化調査研究分科会で検討されている「ソフトウェアの品質特性分析レポート」[10]を参考に、品質特性項目を次の２つのコスト要因に大別した。

(1)規模要因…プログラムの規模に影響を及ぼす要因

(2)単価要因…品質向上努力を表わす要因

その結果、１４個の特性項目が規模要因に分類され、残る１２個が単価要因候補として抽出された。さらに単価要因候補の中から計測することが可能な追跡可能性、自己記述性等の９特性を単価要因（Ｑｉ）として抽出した。表３に抽出結果を示す。

ここで上記レポートの各品質特性項目がどちらの要因に属するかの判断は次のようにして行っている。例えば、単価要因としては、機能的には同じソフトウェアでも、機能追加やバグ解析時のトレースがしやすい（追跡可能性）、一連のドキュメントが判読しやすい（一貫性）、あるいは改造し易い（モジュール性）等、納入後発注者が保守や改良をする上で効果的な作りになっているといった性質を持つ、計測可能な要因を単価要因に分類した。また、計算正確性やアクセス可能性、堅固性等、発注仕様書に実現機能を指定し、結果としてプログラムの規模に反映される性質を持つ要因を機能特性とした。

3．2　単価要因の評価尺度

前章で抽出された９つの単価要因について、対応する評価尺度と努力係数q_iレンジの設定方法を以下に提案する。この評価尺度の計測をもとにq_iを定め、前章で提案したモデル式により発注時見積り及び納入時評価を定量的に行うことが可能になる。

（１）追跡可能性（Q_1）

各開発工程で仕様化された機能が次工程へ正しく展開されている程度を示すものである。特に設計段階での不備を後工程へ持ち越すとその修復に多大な工数がかかると一般に言われている。そこで設計段階の追跡可能性を重視することとし「ステップ当りの設計ドキュメントのレビュー指摘項目数」を評価尺度として採用する。

評価尺度１：$\dfrac{\text{レビュー指摘項目数}}{\text{ＤＳＬ}}$

努力係数q_1レンジ：尺度の値が小さいほど品質がよく、q_1が高い（ただし、厳密なレビューを実施することが前提）

（２）一貫性（Q_2）

ソフトウェアの開発手法、手順、ドキュメント様式等に関し、ソフトウェア開発作業工程を通して一貫している度合を示すものである。開発方法、手順等を規定した作業標準が無いと開発技術が不安定となり、機能の漏れ、不備、不整合等が発生しやすい。このような点から、一貫性の評価尺度として「開発作業標準の適用度」を採用する。

評価尺度２：開発作業標準の適用度

努力係数q_2レンジ：

品質がよい↑	全工程適用（発注側標準）
q_2が高い	部分適用　（　〃　）
	全工程適用（受注側標準）
q_2が低い	部分適用　（　〃　）
品質がわるい↓	適用なし

（３）自己記述性（Q_3）

生産物にソフトウェアの機能あるいは機能間の関係の理解を助ける記述がどれだけされているかの程度を示すものである。記述が不足しているとソフトウェアの変更、改造が難しくなる。一般にこれらの記述は設計ドキュメントとソースプログラム中に記述されているので、自己記述性の評価尺度として「ステップ当りの設計ドキュメント枚数」と「ステップ当りのソースコード中のコメント行数」を採用する。

評価尺度３１：$\dfrac{\text{設計ドキュメント枚数}}{\text{ＤＳＬ}}$

表3　ＩＳＯのソフトウェア品質特性分類[10]に基づくコスト要因の抽出

品質特性			内部特性の概要	コスト要因としての分類		備考
外部特性	関係	内部特性		規模要因	単価要因	
機能性		完全性	要求仕様書に記述された機能が漏れなく実現されていることを示す性質	○		
		追跡可能性	要求仕様書から各種設計書さらにプログラム及び説明書へと機能の関連が追跡できる性質		◎	Q1
		一貫性	ソフトウェアを開発、設計する際の設計方法、コーディング方法、ドキュメントの記述方法が一貫している（設定されている）性質		◎	Q2
		自己記述性	ソフトウェアが有している機能及び、機能と機能との関連をプログラム自身や説明書、ドキュメントが説明している性質		◎	Q3
		無矛盾性	システムやプログラムが有する同一目的の機能が矛盾を含んでいない性質	○		
信頼性		計算正確性	計算結果、出力が要求精度を達成する性質	○		
		データ共通性、通信手順共通性	システム内／システム間で使用するデータ、通信手順、インタフェースが共通している性質	○		
		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、自由に使用できる性質	○		
		アクセス制御性	アクセスＩＤやコマンドをチェックし、ソフトウェアやデータへのアクセスを制御できる性質	○		
		アクセス監査性計測性	ソフトウェアやデータへのアクセス記録を残せる、プログラムの動作状況を観察できる性質	○		
保守性		堅固性、整合性安全性	操作誤りや内部異常の発生に対してデータやプログラムが破壊されない性質	○		
		モジュール性構造性	ソフトウェアが適切に構造化されている性質プログラムの変更が局所的となる性質		◎	Q4
		単純性	機能の理解しやすさに関する性質	○		
		自己包含性	他のプログラムに依存しないで自分自身だけで機能を果たすことができる性質	○		
		拡張性	仕様の追加、変更に対して、容易に対応できるように準備がされている性質	○		
移植性		環境独立性	プログラムが特定のソフトウェア、ハードウェア、データの環境に依存せず動作できる性質		◎	Q5
		製品管理性	製品が正しく管理できる性質	○		
		統一性伝達性	意味、表現、手順が一義的、同一的である性質入出力の形式や内容が統一されている性質	○		
使用性		表現性、階層性比喩性、注目性	入出力や処理の論理、結果をうまく表現する性質		○	
		適時性、適量性	使用者との時間的、量的関係を良好にする性質		○	
		明瞭性、簡潔性	不要な情報がなく情報がコンパクトな性質		○	
		完備性	ソフトウェアの使用を助けるための機能や事物が揃っている性質		◎	Q6
		説明性、選択性誘導性、省力性環境適合性	必要な情報を提供し、使用者の能力や好みに合わせられ、ソフトウェアが持つ論理にうまく使用者を誘導し、投入労力が少なく、カスタマイズが容易な性質	○		
効率性		処理速度	処理結果が得られるまでの速さの性質		◎	Q7
		処理能力	単位時間に処理できる仕事量の性質		◎	Q8
		資源使用量	処理に必要なハードウェア量の性質		◎	Q9

（注）◎：単価要因項目　○：規模要因項目または計測不可能な単価要因項目を示す．

評価尺度３２：$\frac{\text{コメントコード行数}}{\text{ＤＳＬ}}$

努力係数q_{31}，q_{32}レンジ：尺度はいずれも大きいほど品質がよく、q_{31}，q_{32}も高い

（４）モジュール性（Q_4）

プログラム理解を容易にし、保守性や移植性を確保するためモジュール設計の適切さの程度を示すためのものである。モジュール性の評価尺度としてモジュール分割の適切さを表す「モジュール強度」「モジュール結合度」「平均モジュールサイズ」[19]を採用する。

評価尺度４１：モジュール強度（一モジュール内の構成要素のまとまりの強さを示す）

努力係数q_{41}レンジ：強度が強いほど品質がよく、q_{41}が高い（プログラムを構成するモジュールの中で最も強度が弱いモジュールをプログラム全体の強度とする）

品質がよい↑	機能的／情報的強度
q_{41}が高い	連絡的強度
	手順的強度
	時間的強度
q_{41}が低い	論理的強度
品質がわるい↓	暗号的強度

評価尺度４２：モジュール間結合度（モジュール間の情報結合の緩さを示す）

努力係数q_{42}レンジ：結合度が弱いほど品質がよく、q_{42}が高い（プログラム中で最も結合度が強い情報結合を該プログラムの結合度とする）

品質がよい↑	データ結合
q_{42}が高い	スタンプ結合
	制御結合
	外部結合
q_{42}が低い	共通結合
品質がわるい↓	内容結合

評価尺度４３：平均モジュールサイズ

努力係数q_{43}レンジ：尺度の値が小さいほど品質がよく、q_{43}が高い

（５）環境独立性（Q_5）

プログラムの各種動作環境（ハード機種、ＯＳ、言語等）への依存度を定量化するもので、それが小さいほど移植が容易となり、品質が高くなる。

この依存度を測定する尺度としては、ＯＳや言語の標準仕様との一致度合で見る方法もあるが、発注時に定量的な指定が難しい。よって、環境独立性の評価尺度としては「ステップ当りの環境依存ソースコード量」を採用する。

評価尺度５：$\frac{\text{環境依存ＤＳＬ}}{\text{ＤＳＬ}}$

努力係数q_5レンジ：尺度の値が小さいほど品質がよく、q_5が高い

（６）完備性（Q_6）

ソフトウェアの使用を助ける手段の充実度を示すものであり、マニュアルの充実度として捉えることとする。従って、完備性として「ステップ当りのマニュアル枚数」を採用する。

評価尺度６：$\frac{\text{マニュアル枚数}}{\text{ＤＳＬ}}$

努力係数q_6レンジ：尺度の値が大きいほど品質がよく、q_6が高い（ただし、ユーザマニュアルの品質確保のための記述ガイドライン等に従っていること）

（７）処理速度（Q_7）

ソフトウェアの実行効率を測る尺度の一つとしてレスポンスタイムがある。これは所定の処理を実行する速さであり、プログラムの構成、処理アルゴリズム等により左右される反面、動作環境であるハードウェアや同時利用数等の利用特性にも影響を受ける。従って、処理速度の評価尺度としては、絶対値としてのレスポンスタイムではなく、要求条件としての許容レスポンスタイムに対する短縮度合で計量する。

評価尺度７：$\frac{\text{実現レスポンスタイム}}{\text{許容レスポンスタイム}}$

努力係数q_7レンジ：尺度の値が小さいほど品質がよく、q_7が高い

（８）処理能力（Q_8）

単位時間に遂行できる仕事量を測る尺度としてスループットがある。例えば１時間当りのジョブ数である。処理能力についても処理速度と同様に動作環境などの影響を受けるため、絶対値としてのスループットは評価尺度に成り難い。そこで要求条件の許容スループットに対する実現可能なスループット量で計量する。

評価尺度８：$\frac{\text{実現スループット}}{\text{許容スループット}}$

努力係数 q_8 レンジ：尺度の値が大きいほど品質がよく、q_8 が高い

（９）資源使用量（Q_9）

ソフトウェアが目的の処理を実行するうえでの、資源の使用効率の良さを示すものである。資源使用量は少ない方が効率的であるが、ソフトウェア種別等によりそれらの値を一律的に規定することは困難である。そこで、要求仕様の中で規定する資源使用量に対する節約度で計量する。

資源には種々のものがあり、例えば、主記憶使用量とＣＰＵ使用量の場合、下記のように設定する。他の資源についても同様に設定する。

評価尺度９１：$\frac{\text{実際の最大主記憶使用量}}{\text{許容主記憶使用量}}$

評価尺度９２：$\frac{\text{実際の最大ＣＰＵ使用量}}{\text{許容ＣＰＵ使用量}}$

努力係数 q_{91}，q_{92} レンジ：尺度の値がいずれも小さいほど品質がよく、q_{91}，q_{92} が高い

３．３　評価尺度設定の妥当性評価

自己記述性（Q_3）を例に３．２節で選定した評価尺度の妥当性を、類似した５本のプログラムの開発例をサンプルケースにして検証してみる。

自己記述性の評価尺度３１では、「①単位規模当りの設計ドキュメント枚数が多ければ多いほど品質がよく、②ステップ単価は高い。」と仮定している。

図１はサンプルケースについて設計ドキュメント枚数とバグ検出数の関係をみたものであり、上記の①の傾向が表れている。また図２は設計ドキュメント枚数とステップ単価の関係をみたものであり、②の傾向が表れている。

自己記述性についても、今後更に、サンプル数を増やしてこの傾向の確実性を確認する必要があるが、このようにして、他の評価尺度の設定方法の妥当性についても検証していく。

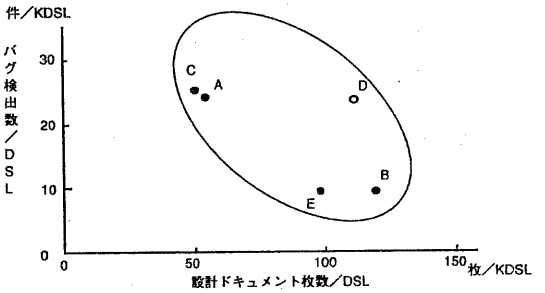

図1 設計ドキュメント枚数とバグ検出数との関係
(サンプル数＝５)

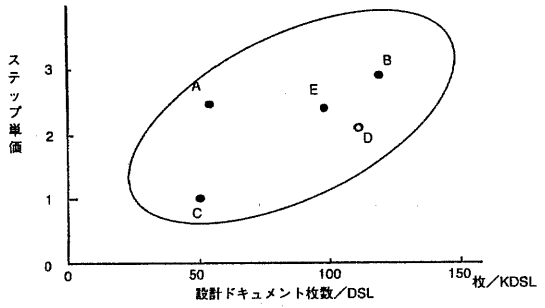

図2 設計ドキュメント枚数とステップ単価との関係
(サンプル数＝５)

４．おわりに

既存のコスト見積りモデルの多くは、ソフトウエアの製造者側すなわち受注者側の視点から提案されているのに対し、本稿では発注者側に立ったモデル化の可能性を追究しようしている。

ＣＯＳＭＯＳで採択しようとするコスト要因は何れも発注に際して明確に規定することが望ましいものであるが、特に単価要因について規定しき

れていない場合もあると思われる。本モデルの導入を機に各要因について発注条件が明確に規定されることになり、希望の品質を備えたソフトウエアの入手が確実になる。また、本モデルをカスタマイズするための貴重なデータが得られてくる。

本モデルを確立するためには、下記の検討課題が残存している。

(1) フィールドデータの計測・分析手段の整備
(2) フィールドデータの分析に基づく、単価要因の努力係数 q_i のレンジ設定
(3) 規模要因分析によるＤＳＬの推定精度の向上

謝辞：本稿の作成にあたってご指導や資料の提供をいただいたソフトウエア研究所　鶴保征城所長、古山恒夫主幹研究員、高橋宗雄主幹研究員、小野佳英主任研究員、三宅武司研究主任、ならびに助言をいただいた大阪大学基礎工学部情報工学科松本健一先生に感謝いたします。

参考文献

[1] B. W. Boehm, "Software Engineering Economics," Prentice-Hall, Inc, 1981
[2] B. W. Boehm, P. N. Papaccio, "Understanding and Controlling Software Costs," IEEE Tr. SE. Vol.14, No.10, 1988
[3] H. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Trans. on Software Engineering, Vol. 4(4), pp. 345-361, July 1978
[4] A. Albrecht and J. Gaffeny, "Software Function Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Tr. SE., Vol.9, No.6, 1983
[5] C. R. Symons, "Function Point Analysis: Difficulties and Improvements"
[6] 宮崎，山田，板倉，"段階的規模見積りモデルの概念とその作成方法，" 情報処理論文誌，Vol.32, No.2, 1991
[7] M. Itakura and A. Takayanagi, "A Model for Estimating Program Size and its Evaluation," Proceedings, 6th International Conference on Software Engineering, pp. 104 - 109, Sep. 1982
[8] 藤野，花田 編，"ソフトウエア生産技術，" 電子通信学会，1985
[9] 鳥居，"ソフトウエア・メトリクスの意義と方法論，" 昭和63年電気・情報関連学会連合大会，29-1, pp. 5-19 - 5-22
[10]平成元年度 ISO/IEC JTC1/SC7 STD-WG5 （ソフトウエア品質特性）分科会報告
[11]三宅，高橋，花田，"Statistically based Program Size Estimation," COMPSAC89, 1989
[12]R. B. Grady, D. L. Caswell, 清水・水野訳，"ソフトウエア・メトリクス，" 日経ＰＢ社，1990
[13]Basili, V., and M. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering, IEEE, 1978, pp. 116-123
[14]C. Jones, "Programming Productivity," Issues for the Eighties, IEEE Computer Society Press, 1981
[15]B. Zimmer, "Software Quality and Productivity Analysis at Hwelett Packard," COMPSAC89, 1989
[16]D. B. Brown, "A Cost Model for Determining the Optimal Number of Software Test Cases," IEEE Tr. SE., Vol.17, No.2, 1989
[17]久保 監修，"富士通におけるソフトウエア品質保証の実際，" 日科技連出版社，1989
[18]花田 編，"ソフトウエアの計画と管理，" 日科技連，1987
[19]G. J. Myers, 久保・国友 訳，"高信頼ソフトウエア――複合設計，" 近代科学社，1976
[20]"ソフトウエア生産性評価の手引き，" 情報サービス産業協会，1989