社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2005-73(2005-12)

保守請負時を対象とした労力見積りのためのメトリクスの提案

早瀬 康裕† 松下 誠† 楠本 真二† 井上 克郎† 小林 健一††
吉野 利明††

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻 〒560-8531 大阪府豊中市待兼山町 1-3
†† (株) 富士通研究所 IT コア研究所 ソフトウェアイノベーション研究部
E-mail: †{y-hayase,matusita,kusumoto,inoue}@ist.osaka-u.ac.jp,
††{kenichi,yoshino.toshi}@jp.fujitsu.com

あらまし コストなどの問題によって保守が困難になったソフトウェアプロダクトの保守作業を，元の業者に代わって請け負うサービスが一般化している．このようなサービスでは，事前に保守作業にかかる費用の見積りを行う必要がある．しかし，見積りに適用できる客観的な基準がなかったため，熟練者が経験に基づいて見積りを行ってきた．そこで我々は，見積り時に用いることができるソースコードと影響波及解析の手法とを用いて，保守作業の労力見積りに用いることができるメトリクスを提案する．

キーワード メトリクス，労力見積り，保守，影響波及解析

Metrics for effort estimation on contracting maintenance service

Yasuhiro HAYASE†, Makoto MATSUSHITA†, Shinji KUSUMOTO†, Katsuro INOUE†,
Kenichi KOBAYASHI††, and Toshiaki YOSHINO††

† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka 560-8531, Japan
†† Fujitsu Laboratories Limited
E-mail: †{y-hayase,matusita,kusumoto,inoue}@ist.osaka-u.ac.jp,
††{kenichi,yoshino.toshi}@jp.fujitsu.com

Abstract A software vendor sometimes contracts the maintenance of a software product that other software vendors gave up maintaining due to costs etc. In such case, the vendor must estimate costs of maintenance before contracting. However, due to lack of an objective measure, an expert estimates the cost on the basis of his/her experience. We will propose some metrics for estimating maintenance effort. These metrics are computed by applying change impact analysis to the source code which is available before contracting.

Key words Metrics, Effort Estimation, Maintenance, Impact Analysis

1. はじめに

ソフトウェア保守とは，“納入後，ソフトウェアプロダクトに対して加えられる，フォールト修正，性能または他の性質改善，変更された環境に対するプロダクトの適応のための改訂”と定義されている[1]．また，長期間に渡って使用されるソフトウェアでは，総所有コストのうち保守コストの占める割合が半分以上になると言われている．

一般に，ソフトウェア保守作業は以下のような手順で行なわれる．まず，どのような作業を行うかを決め，その作業にどれだけの費用がかかるかを見積もる．見積りが承認された場合には，ソフトウェア全体から実際に変更される部分を特定し，実際の変更を行う．そして，回帰テストを行い，変更によって欠陥が作りこまれていないことを確認し，作業を終える．

現在，保守作業の見積りには，新規開発の見積りと同様の手法が用いられている．しかし，保守作業は既存のソフトウェアプロダクトを対象とするため，新規開発向けの見積り手法を適用することは適切ではない．

そこで我々は，保守の見積りを行う際に利用可能なメトリクスを提案する．メトリクスの計算のために，プロダクトを有向グラフでモデル化した．また保守作業の手順を 3 つの工程に分割して考える．それぞれの工程のコストを有向グラフを用い

て計算し，得られたコストを足し合わせることで「保守ポイント」と「一般化保守ポイント」の 2 つのメトリクス値を算出する．このメトリクスは，既存のメトリクスが反映していなかった保守作業の範囲を考慮しているため，より保守の見積りに適していると考えられる．

以降，2. 節で保守作業について説明し，3. 節でそのモデル化を行う．そして，4. 節で提案するメトリクスとその計算方法について説明する．5. 節ではメトリクスの評価方法について検討し，6. 節で関連研究について述べる．最後に，7. 節でまとめと今後の課題について説明する．

2. ソフトウェア保守作業

ソフトウェアプロダクトには様々な変更が行なわれる．これらの保守作業は，目的ごとに次の 4 つに分類される．

修正保守 (corrective maintenance)　発見された問題を修正するために，納入後に実施される，ソフトウェアプロダクトの対処的改変．

予防保守 (preventive maintenance)　ソフトウェアプロダクトのなかに存在する潜在的なフォールトが実際の障害を起こす前に，それを検出し修正するために納入後に実施される，ソフトウェアプロダクトの改変．

適合保守 (adaptive maintenance)　変化した，または変化しつつある環境において，ソフトウェアプロダクトを続けて使用可能なように維持するために，納入後に実施される，ソフトウェアプロダクトの改変．

完全化保守 (perfective maintenance)　性能または保守性を改善するため，納入後に実施される，ソフトウェアプロダクトの改変．

長期間使用されるソフトウェアプロダクトでは，保守を重ねることでソフトウェアの構造が複雑になり，障害の発生を招いたり，保守作業のコストが増大したりすることが問題となっている．このようにして保守を続けることが困難となったソフトウェアプロダクトを，それまで作業を行っていた業者に代わって保守するサービス (保守請負サービス) が行なわれている．

通常，保守請負サービスの契約する前には，図 1 のような作業を行う．保守請負サービスの実施者は，まず，依頼者からプロダクトを受け取り，プロダクトの価値と，サービスを実施する場合にかかるコストを見積る．サービス依頼者は，実施者によりプロダクトの価値と保守コストの見積りを見て，保守サービスを発注するかどうかを決定する．

ここで，もしプロダクトの評価に失敗すると，コスト見積りに大きなずれが生じてしまう．例えば，過去の類似プロダクトの作業実績を基に，保守コストの見積りを行ったとする．しかし，見積り対象のプロダクトの構造が、過去の類似プロダクトに比べて非常に複雑であった場合には，過少な見積り値を算出してしまうことになる．

このように，保守作業の見積りでは，現在のプロダクトの状態を考慮することが重要である．

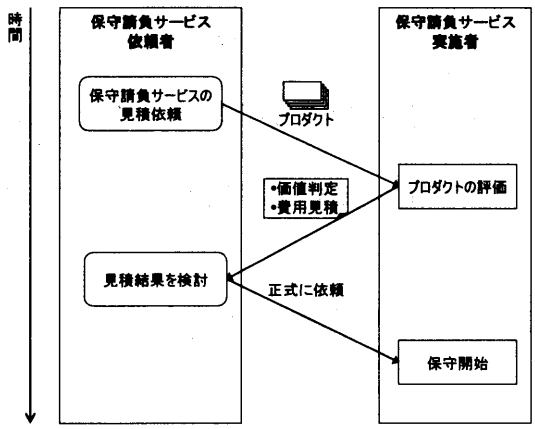

図 1　保守請負サービス開始までの流れ

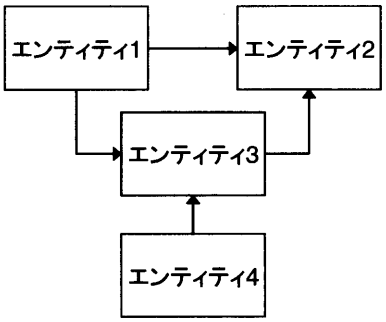

図 2　プロダクトモデルの例

3. 保守作業のモデル化

本節では，見積りに用いるメトリクスの計算を行うために必要な，プロダクトとプロセスのモデル化を行う．

3.1 プロダクトモデル

保守対象のシステム P は，ソースコード，オブジェクト，マニュアル，設計文書，その他の中間生成物などの様々な種類のエンティティの集合である．各エンティティに対する保守要求の R は，その保守作業で変更されるエンティティの集合である．あるエンティティ e1 に対して作業を行なった場合，別のエンティティ e2 に対して作業が発生する場合，e1 は e2 に「影響する」と言う．この関係を影響波及関係と呼ぶ．P の各エンティティを頂点，影響波及関係を有向辺とした有向グラフを，P のプロダクトモデル $G_p = (N, E)$ (N は頂点の集合，E は有向辺の集合) と呼ぶ．図 2 にプロダクトモデルの例を示す．

プロダクトモデルでは，エンティティとしていろいろな種類のプロダクトを仮定しているが，以降の議論では，簡単化のためソースコードに限定し，モジュールと呼ぶ事にする．ただし，ドキュメントや設計文書など，他の種類のエンティティに対して，以降の議論を拡張する事は可能である．

辺は影響波及の度合を表す値を持っており，頂点はそのエンティティの複雑さを表す値を持っている．詳細については，それぞれ 3.3 節と 3.4 節で説明する．

モジュールの粒度としては，プログラミング言語により様々なものが考えられる．例えば Java の場合，メソッド，クラス，パッケージ，jar アーカイブといった階層があり，モジュール

の候補となりうる．何をモジュールとするかは，主に対象とするプロダクトの大きさを基準として，メトリクスの計測者が決定する．小さなプロダクトの場合はメソッドやクラスなどをモジュールに用い，大きなプロダクトの場合はパッケージや jar アーカイブなどの単位を用いるのが適当である．

モジュール間の影響波及関係の計算には，呼び出し関係の解析と，データフロー解析を用いる．また，他の種類のエンティティに対しても，影響波及関係が定義可能である．ソースコードと文書の間の影響波及関係や，文書と文書の間の影響波及関係は，自然言語の類似度を用いて解析 [2] する．

以上に加えて，CVS リポジトリ等から細粒度の変更履歴が入手可能な場合には，同時更新傾向に基づいた影響波及関係の推測手法である Logical Coupling [3] を利用する．Logical Coupling とは，依存関係を持つエンティティが，同じ変更者によって同時に変更されることが多いことを利用して，ソフトウェアの変更履歴から，同時に更新される頻度の高いエンティティの組み合わせを抽出することで，プログラムや自然文の意味的な解析に頼らずに依存関係を計算する手法である．

3.2 プロセスモデル

保守作業のプロセスを，図 3 のように，s1, s2, s3 の 3 つの工程から構成されているとする．入力は，保守対象のシステム P と，変更要求 R である．R は，障害や機能追加案件，または，完全化要求などの保守案件に対して，直接の変更作業を行なう必要のあるモジュールの集合である．出力は，変更，テスト済みのシステム P' である．

3.3 エンティティの複雑さ

エンティティの複雑さとは，そのモジュールを理解したり，テストしたりするために必要なコストを表す値である．

このコストを表す指標としては，具体的には，ソースコードの行数や，McCabe のサイクロマチック数 [4]，CK メトリクス [5]，Halstead の複雑度メトリクス [6] などを用いる．

3.4 辺の重み

プロダクトモデルの辺の重みは，影響波及関係の強さを表している．重みは，0 以上 1 以下の実数値であり，影響が強いほど大きな値を取る．

例えば，モジュール間の呼び出しの数を c，データフローを起こす変数の数を d としたとき，$1-\alpha^c\beta^d$ (α と β は 0 より大きく 1 より小さい定数) と表すことができる．この式は，変数 c および d の定義域を 0 以上の整数とした場合，値域は 0 以上 1 未満となり，c と d それぞれについて単調増加な関数である．つまり，$c=0$ かつ $d=0$ のとき，すなわちモジュール間で呼出しや変数の共有が行われていない場合には，この式の値は 0 となる．また，c と d が無限に増加してゆけば，この式の値は無限に 1 に近づく．

プロダクトの規模が大きいなどの理由で，このような計算を行うのが難しい場合には，辺の重みを定数にすることもできる．

4. 保守ポイント

4.1 保守ポイント

保守作業では，ソフトウェア全体を対象として作業を行うのではなく，最初に小さな作業範囲を特定し，その範囲内で作業を行なうことが分かっている [7]．そのため，保守作業に必要な労力は，作業範囲内の複雑さによって決まると考えるのが妥当である．

そこで，上記のプロダクトモデルとプロセスモデルを用いて，保守作業のコストの指標である「保守ポイント」$C(G_p,R)$ を以下のように定義する．

$$C(G_p,R)=C_{s1}(G_p,R)+C_{s2}(G_p,R)+C_{s3}(G_p,R)$$

ここで C_{s1}，C_{s2}，C_{s3} はそれぞれ，s1，s2，s3 における作業コストの指標である．(図 4)

実際の保守作業は各工程それぞれを行なう必要があるが，保守コストを見積もる場合は，主要なコスト要因となる工程のみを見積り，他の工程に関しては計算しない場合がある．一般にはテスト工程が大きなコスト要因と言われているので，この工程のみの見積りをする場合が多い．以降の議論では，主にテスト工程 s3 のコスト見積りを中心にして議論するが，他のステップでも同様な議論は可能である．

C_{s3} は，以下のように定義される．

$$C_{s3}(G_p,R)=\sum_{r\in R}\mathrm{cx}(r)+\sum_{r'\in A(G_p,R)}w(G_p,R,r')\mathrm{cx}(r')$$

$A(G_p,R)$：グラフ G_p 上で R に含まれる頂点から到達可能な頂点の集合

$w(G_p,R,r')$：グラフ G_p 上で，R に含まれるいずれかの頂点から r' までの経路のうち，重み最大となる経路の重み

$\mathrm{cx}(r)$：モジュール r に対する作業コストを表すポイント

モジュールに対する作業コストには，3.3 節で説明したエンティティの複雑さを用いる．経路の重みには，経路に含まれる辺の重みをすべて掛け合わせた値を用いる．

4.2 一般化保守ポイント

保守ポイントを求めるためには，G_p と R が必要である．しかし，保守請負時においては，どのような作業が発生するかが分からないため，具体的な R を求めるのは非常に困難である．そこで，R の代わりに保守要求 r の確率分布 R_d を用いてコスト指標モデル $C_{\mathrm{gen}}(G_p,R_d)$ を以下のように定義できる．

$$C_{\mathrm{gen}}(G_p,R_d)=\sum_{i=1}^{n}\{a_iC_{s3}(G_p,\{r_i\})\}$$

ただし，$R_d=\{a_1,a_2,\ldots,a_n\}$ (n は G_p の頂点数)，$0\leq a_i\leq 1$，$a_1+a_2+a_3+\cdots+a_n=1$ である．

R_d を均等な $1/n$ にすれば，簡単に試算することができる．このモデルを一般化保守ポイント $C_{\mathrm{ave}}(G_p)$ と呼ぶ．

$$C_{\mathrm{ave}}(G_p)=C_{\mathrm{gen}}(G_p,\{1/n,\ldots,1/n\})=\frac{\sum C_{s3}(G_p,\{r_i\})}{n}$$

また，R_d を計算するために，保守ポイントを用いることも

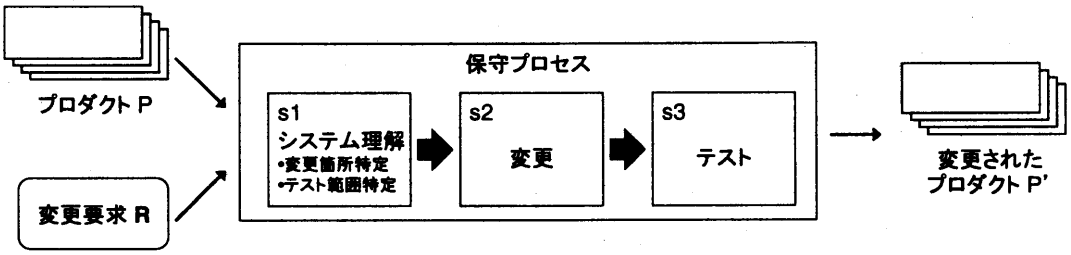

図 3　保守作業の工程

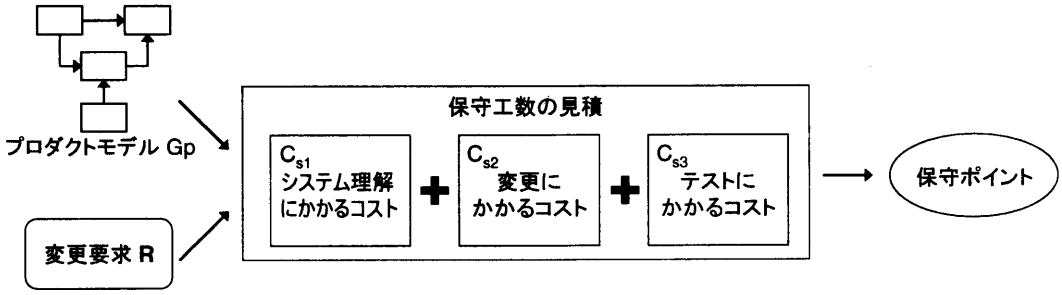

図 4　保守ポイントの計算モデル

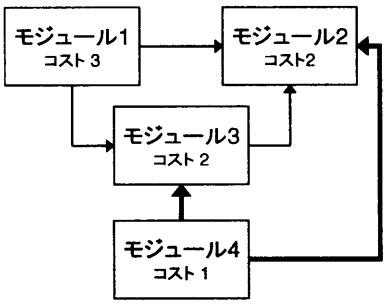

図 5　保守ポイントの計算例

可能である.

$$R_d = \{C_{s3}(G_p, \{r_1\})^\alpha, C_{s3}(G_p, \{r_2\})^\alpha, \ldots, C_{s3}(G_p, \{r_n\})^\alpha\}$$

(ただし $\alpha \geq 0$ の定数)

4.3 計　算　例

図 5 のプロダクトモデル上で，保守ポイントと一般化保守ポイントの計算例を示す．簡単のため，経路の重みは，全て 1 であるとする.

まず保守ポイントの例を示す．図のプロダクトモデルに対し，変更要求としてモジュール 4 が与えられた場合を考える．モジュール 4 が変更された場合に影響を与えるのは，モジュール 3 とモジュール 2 である．よって，モジュール 4 を変更するときの保守ポイントは，モジュール 4，モジュール 3，モジュール 2 のコストの和 $1+2+2=5$ と計算される.

次に一般化保守ポイントの例を示す．上のモジュール 4 を変更するときの保守ポイントの計算例と同様に計算すると，モジュール 1，モジュール 2，モジュール 3 を変更するときの保守ポイントは，それぞれ，7，2，4 となる．よって，このプロダクトモデルの一般化保守ポイントは，$(7+2+4+5)/4 = 4.5$ となる.

5. 実験計画

保守ポイントが，保守作業のコストと相関を持っているかを調べるために，ある程度の規模のプログラムに対して，保守作業にかかった実験と，保守ポイントとの相関を調べる実験を行う予定である.

1 つめの実験では，修正保守の作業コストと保守ポイントの値との相関を調べる．図 6 に示すように，被験者に欠陥を含むプロダクト P と，不具合情報を渡す．被験者は 3.2 節で説明した手順「s1 システムの理解」「s2 変更」「s3 テスト」で，不具合の修正作業を行う．このとき，作業にかかった時間を計測しておく．プログラムには複数の欠陥が埋め込まれており，被験者は，欠陥を一つずつ修正する．また，複数の被験者が同じ作業を行う.

プロダクト P と障害の修正により変更されるモジュールから保守ポイントを計算し，作業にかかった時間と比較することで，修正保守と保守ポイントとの相関を評価する.

2 つめの実験では，適合保守あるいは完全化保守の作業コストと保守ポイントの値との相関を調べる．図 7 に示すように，被験者にプロダクト P と，機能追加要求とを渡す．被験者は 3.2 節で説明した手順「s1 システムの理解」「s2 変更」「s3 テスト」で，機能追加を行う．このとき，作業にかかった時間を計測しておく．複数の機能追加要求を用意しておき，被験者は，一つずつ機能追加を実施する．また，複数の被験者が同じ作業を行う.

プロダクト P と機能追加により変更されるモジュールから，保守ポイントを計算し，作業にかかった時間と比較することで，評価を行う.

6. 関連研究

Leitch ら [8] は，リファクタリングによる依存関係の変化を，

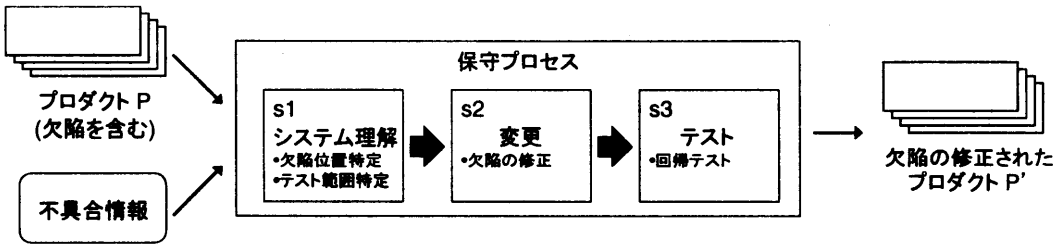

図 6　実験 1: 欠陥の修正

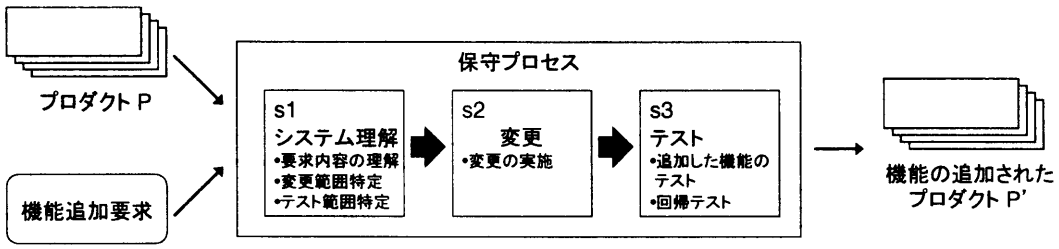

図 7　実験 2: 機能追加

変更前後の依存関係と COCOMO II による作業量見積りを用いて評価する方法を提案した．

実績に基づいて，ソフトウェア部品の再利用性を計算する手法であるコンポーネントランク法 [9] は，メソッドの利用関係を推移的にたどり，値を計算する点で類似している．コンポーネントランク法は部品の利用関係を辿るが，我々の手法は依存関係を辿る．また我々の手法は，保守の困難さを計測するという点で異なる．

Ko ら [7] は，保守に特化した統合開発環境を作るために，保守作業を観察した．その結果，保守作業では，最初に小さな作業範囲を特定し，その範囲内で作業を行なっていることがわかった．我々のメトリクスは，機械的に計算可能な影響範囲を用いて作業範囲を近似することで，保守作業の工数を見積もっていると考えられる．

保守作業の見積りに用いる新しいメトリクスが，いくつか提案されている．Lindvall ら [10] は，モジュール結合度に基づいて，設計と実装との乖離を計測するメトリクスを提案し，実際の保守作業の前後に計測した．Chen ら [11] らは，オブジェクト指向ソフトウェアの凝集度を，属性とメソッドの間の依存関係を用いて計測する方法を提案した．Tran-Cao ら [12] は，データ操作やアルゴリズムの複雑さを考慮した機能量を計算する方法を提案した．Arisholm ら [13] は，オブジェクト指向ソフトウェアの結合度を，実行時情報を用いて計測する方法を提案した．これらのメトリクスは，モジュール単体，あるいは直接利用しているモジュールまでを考慮したものであるが，我々の手法は影響範囲を推移的に辿る点で，実際の保守作業をより反映していると考えられる．

Bianchi ら [14] は，ソフトウェア開発の設計・実装など各段階を構成するコンポーネントの段階間の依存関係を用いて，ソフトウェアの劣化具合を測定するメトリクスを提案した．それぞれのモジュールの複雑さを考慮しない点や，値の利用方法が大きく異なる．

その他に，ソフトウェアの保守を難しくしている部分や，保守を進めるうちに劣化した部分を検出するために，以下のような研究が行なわれている．片岡ら [15] は，リファクタリングよる結合度の変化を事前に予測することで，効果的なリファクタリングを推薦する方法を提案した．Marinescu [16] は，オブジェクト指向プログラムのデザイン上の欠陥を見付けだすための手法を提案した．Grosser ら [17] は，Java で書かれたソフトウェア設計の安定性を計測するために，事例学習システムを用いることを提案した．

見積りに用いるべきメトリクス値と，見積りに用いる計算手法について，以下のような研究が行なわれている．Jørgensen [18] は，実際の保守作業で様々なメトリクス値を収集し，適切なメトリクス値と見積り手法を調べた．Fioravanti ら [19] は，適応保守の見積りに有用なメトリクスを提案し，実際に見積りを行なった．Bouktif ら Ahn ら [20] は，ファンクションポイントを用いて保守工数の見積りを行ない，調整係数の有効性を評価した．これらの手法は，過去の実績を元に見積りを行うため，過去の情報が利用できない保守請負時に適用するのは困難である．

7. ま と め

この論文では，まず保守作業の労力見積りに用いるメトリクスを定義するために，ソフトウェアプロダクトを有向辺と頂点から成るグラフとしてモデル化した．グラフの頂点はプロダクトを構成するエンティティであり，有向辺はエンティティ間の影響波及関係を表す．

そして，このモデル上で 2 つのメトリクスを定義した．

1 つめのメトリクスである保守ポイントは，保守作業で変更されるモジュールが与えられたときに，その影響範囲にあるエンティティの複雑さから計算されるメトリクスであり，個々の保守作業の労力見積りに用いることができる．

2 つめのメトリクスである一般化保守ポイントは，エンティティが平均的に変更されると仮定することで，具体的にどのエンティティが変更されるかが分からない場合であっても，労力

を推定することができるメトリクスである．このメトリクスは，保守請負時の見積りに用いることができる．

今後の課題は，提案したメトリクスが労力とどのような関係を持つかを評価する実験を行い，既存のメトリクスとの比較を行うことである．

文　献

[1] Salvatore Mamone. The ieee standard for software maintenance. *SIGSOFT Softw. Eng. Notes*, Vol. 19, No. 1, pp. 75–76, 1994.

[2] Wei Zhao, Lu Zhang, Yin Liu, Jing Luo, and Jiasu Sun. Understanding how the requirements are implemented in source code. In *APSEC*, pp. 68–77. IEEE Computer Society, 2003.

[3] Harald Gall, Karin Hajek, and Mehdi Jazayeri. Detection of logical coupling based on product release history. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, p. 190, Washington, DC, USA, 1998. IEEE Computer Society.

[4] Arthur H. Watson Thomas J. McCabe. Software complexity. *Crosstalk, Journal of Defense Software Engineering*, Vol. 7, No. 12, pp. 5–9, December 1994.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6, pp. 476–493, 1994.

[6] Maurice H. Halstead. *Elements of Software Science*. Operating and Programming Systems Series. Elsevier Science Inc., New York, NY, USA, 1977.

[7] Andrew J. Ko, Htet Aung, and Brad A. Myers. Eliciting design requirements for maintenance-oriented ides: a detailed study of corrective and perfective maintenance tasks. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pp. 126–135, New York, NY, USA, 2005. ACM Press.

[8] Robert Leitch and Eleni Stroulia. Assessing the maintainability benefits of design restructuring using dependency analysis. In *IEEE METRICS* [21], pp. 309–.

[9] 横森励士, 藤原晃, 山本哲男, 松下誠, 楠本真二, 井上克郎. 利用実績に基づくソフトウェア部品重要度評価システム. 電気情報通信学会論文誌, Vol. J86-DI, No. 9, pp. 671–681, September 2003.

[10] Mikael Lindvall, Roseanne Tesoriero Tvedt, and Patricia Costa. Avoiding architectural degeneration: An evaluation process for software architecture. In *IEEE METRICS*, pp. 77–86. IEEE Computer Society, 2002.

[11] Zhenqiang Chen, Yuming Zhou, Baowen Xu, Jianjun Zhao, and Hongji Yang. A novel approach to measuring class cohesion based on dependence analysis. In *ICSM* [22], pp. 377–384.

[12] De Tran-Cao, Ghislain Lévesque, and Alain Abran. Measuring software functional size: Towards an effective measurement of complexity. In *ICSM* [22], pp. 370–376.

[13] Erik Arisholm, Lionel C. Briand, and Audun Føyen. Dynamic coupling measurement for object-oriented software. *IEEE Trans. Software Eng.*, Vol. 30, No. 8, pp. 491–506, 2004.

[14] Alessandro Bianchi, Danilo Caivano, Filippo Lanubile, and Giuseppe Visaggio. Evaluating software degradation through entropy. In *METRICS '01: Proceedings of the 7th International Symposium on Software Metrics*, p. 210, Washington, DC, USA, 2001. IEEE Computer Society.

[15] Yoshio Kataoka, Takeo Imai, Hiroki Andou, and Tetsuji Fukaya. A quantitative evaluation of maintainability enhancement by refactoring. In *ICSM* [22], pp. 576–585.

[16] Radu Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *ICSM*, pp. 350–359. IEEE Computer Society, 2004.

[17] David Grosser, Houari A. Sahraoui, and Petko Valtchev. An analogy-based approach for predicting design stability of java classes. In *IEEE METRICS* [21], pp. 252–262.

[18] Magne Jørgensen. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Trans. Software Eng.*, Vol. 21, No. 8, pp. 674–681, 1995.

[19] Fabrizio Fioravanti and Paolo Nesi. Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *IEEE Trans. Software Eng.*, Vol. 27, No. 12, pp. 1062–1084, 2001.

[20] Yunsik Ahn, Jungseok Suh, Seungryeol Kim, and Hyunsoo Kim. The software maintenance project effort estimation model based on function points. *Journal of Software Maintenance*, Vol. 15, No. 2, pp. 71–85, 2003.

[21] *9th IEEE International Software Metrics Symposium (METRICS 2003), 3-5 September 2003, Sydney, Australia.* IEEE Computer Society, 2003.

[22] *18th International Conference on Software Maintenance (ICSM 2002), Maintaining Distributed Heterogeneous Systems, 3-6 October 2002, Montreal, Quebec, Canada.* IEEE Computer Society, 2002.