修士学位論文

題目

類似度を用いたプログラムの再利用性尺度の提案と実現

指導教官
井上 克郎 教授

報告者
藤原 晃

平成 14 年 2 月 13 日

大阪大学 大学院基礎工学研究科
情報数理系専攻 ソフトウェア科学分野

平成 14 年度 修士学位論文

類似度を用いたプログラムの再利用性尺度の提案と実現

藤原 晃

内容梗概

再利用性の高いソフトウェア部品を再利用することは，生産性と品質を改善し，結果としてコストを削減できる．ソフトウェア部品の再利用性を評価する方法はこれまでに数多く提案されているが，その方法は全て，部品そのものの持つ静的な特性を計算して再利用性を評価するものとなっている．しかし，再利用性が高いということは，実際に多くのソフトウェア中に再利用されているという実績が定量的に示されることが必要である．つまり，主観的に再利用性が高いと判断されても，実際の利用実績がなければ意味がないという立場である．現実には，従来手法では再利用性が低いと評価されても，多くのシステムで再利用されているという部品は多く存在すると考えられる．そこで，本論文では，利用実績に基づいたソフトウェア部品の再利用性評価手法について提案する．利用実績に基づく再利用性についての基本的な考え方は，以下の (1)~(3) である．(1) ソフトウェアを構成する部品間には相互に利用関係がある，(2) 一般に，時間が経過して多くのプロジェクト開発で再利用などが行われるに連れて部品の利用関係は変化していく，(3) 十分な時間が経過した状態のもとで，被利用数が多い部品は再利用性が高く，再利用性が高い部品から利用されている部品も再利用性が高い．本論文では，この考え方に基づいて，評価手法を定義した．まず，評価対象のソフトウェア部品の集合に対して，それを構成する部品間の利用関係を抽出する．次に，各部品間で類似度が高い部品を集め，その部品群を一つの部品とみなして，部品群同士の利用関係を抽出する．利用関係はそれぞれ再利用性評価値の重みを持つ．部品の再利用性評価値は，その部品が利用される関係の重みの合計値となる．部品の再利用性評価値を求める計算は，行列の固有値計算に帰着される．提案する手法に基づいて，プログラミング言語 Java で開発されたソフトウェア群から再利用性を計測するシステムを開発し，幾つかのソフトウェアシステムに適用した．また，類似部品群の分類基準などのパラメータを変化させて適用実験を行ない，パラメータが再利用性の評価に与える影響を分析した．

主な用語

類似度 (Similarity)

再利用性 (Reusability)

メトリクス (Metrics)

Java (Java)

目 次

1 まえがき

ソフトウェアの大規模化と複雑化に伴い，高品質なソフトウェアを一定期間内に効率良く開発することが重要になってきている．これを実現するために様々なソフトウェア工学技術が提案されてきている．再利用はそれらの中でも最も有効なものの一つである．

再利用は既存のソフトウェア部品を同一システム内や他のシステムで用いることであると定義されている [7]．一般にソフトウェアの再利用は生産性と品質を改善し，結果としてコスト削減するといわれている．再利用の実際の効果を報告した論文も多く発表されている [4, 19, 22].

個々のソフトウェア部品の再利用性を評価する方法はいろいろ提案されている．Etzkorn らは，レガシーソフトウェア中の部品 (C++のクラス) に対して，様々なメトリクス値を計算し，それらの値を正規化して足し合わせることで，再利用性とすることを提案した [10]．また、山本らは，ソースコードが非開示なソフトウェア部品に対して，インタフェース部分の情報のみを用いて再利用性を評価する方法を提案している [35]．これらの方法は全て，部品そのものの持つ静的な特性を計算して再利用性を評価するものとなっている．また，提案された再利用性の評価値の妥当性については，複数の部品に対して得られた再利用性の値の順位と実際のプログラマによる主観的な再利用性の評価の結果が似ているといった評価が行われている．

しかし，再利用性が高いということは，実際に多くのソフトウェア中に再利用されているという実績が定量的に示されることが必要である．主観的に再利用性が高いと判断されても，実際の利用実績がなければ意味がないという立場である．現実には，従来手法では再利用性が低いと評価されても，多くのシステムで再利用されているという部品は多く存在すると考えられる．

そこで，利用実績に基づいたソフトウェア部品の再利用性評価手法について提案する．再利用性についての基本的な考え方は，以下の (1)〜(3) の通りである．(1) ソフトウェアを構成する部品間には相互に利用関係がある．(2) 一般に，時間が経過し，多くのプロジェクト開発で再利用などが行われるに連れて部品の利用関係は変化していく．(3) 十分な時間が経過した状態のもとで，被利用数が多い部品は重要である（再利用性が高い）．また，重要な部品から利用されている部品も重要である（再利用性が高い）．

このような評価手法は計量社会学においては以前から用いられている．例えば論文の引用解析の分野では，Narin らが多くの論文から引用されている論文は重要であり，重要な論文から引用されている論文もまた重要であるという考えに基づいて論文の重要度を評価する手法を 1970 年代に提案している [25, 28, 29]．また，よく知られているサーチエンジン Google[15] では，多くのページ良質なページからリンクされているページはやはり良質なページである

という再帰的な関係をもとに，あらゆるページの重要度を評価している [3, 18, 27].

本論文では，この考え方に基づいて，評価手法を定義した．まず，評価対象のソフトウェア部品の集合に対して，それを構成する部品間の利用関係を抽出する．次に，各部品間で類似度が高い部品を集め，その部品群を一つの部品とみなして，部品群同士の利用関係を抽出する．利用関係はそれぞれ再利用性評価値の重みを持つ．部品の再利用性評価値は，その部品が利用される関係の重みの合計値となる．部品の再利用性評価値を求める計算は，行列の固有値計算に帰着される．

提案する手法に基づいて，プログラミング言語 Java で開発されたソフトウェア群から再利用性を計測するシステムを開発し，幾つかのソフトウェアシステムにも適用した．

以降，2 では，本論文で用いられているソフトウェア部品，類似部品群，相対的再利用性の各概念について説明する．3 では，提案手法「R^3 法」の定義と計算方法について述べる．4 では，Java ソースコードを対象とした相対的再利用性評価システム「R^3-System」の実装について述べる．5 では，R^3-System を実際のソースコードに適用し，有効性を検証する．

2 準備

ここでは，準備としてソフトウェア部品，類似部品群，相対的再利用性の各概念について説明する．

2.1 ソフトウェア部品

一般にソフトウェア部品 (Software Comopnent) は再利用できるように設計された部品とされ，特に部品の内容を利用者が知る必要のないブラックボックス再利用ができるものを指すこともある [1, 20]．本論文ではより一般的に，ソースコードファイルやバイナリファイル，ドキュメントなどの種類を問わず，開発者が再利用を行う単位をソフトウェア部品，あるいは単に部品と呼ぶ．部品間には互いに利用する，利用されるという利用関係が存在する．

2.2 類似部品群

2.2.1 部品間の類似度

ソフトウェアシステム間の類似度を求める研究としてさまざまな研究がある．

Baxter らは二つの抽象構文木（Abstract Syntax Tree）の類似度を定義している [5]．定義している類似度は一致する節点数を全ての節点数で割った値である．また，[23] では，類似度を二つのプログラムの全行数の中で共通部分の行数の割合と定義している．山本らは 2 つのソフトウェアシステムのソースコードを比較し，対応する行を調べることで類似度 S_{line} を定義している [34]．文献 [34] では，類似度を次のように形式的に定義している．ソフトウェアシステム P はそれを構成する要素の集合と考え，P = $\{p_1, \cdots, p_m\}$ と書く．二つのソフトウェアシステム $P = \{p_1, \cdots, p_m\}, Q = \{q_1, \cdots, q_n\}$ に対し，等価な要素の対応 $R_s \subseteq P \times Q$ が得られるとする．P と Q の R_s に対する類似度 $S(0 \leq S \leq 1)$ を次のように定義する．

$$S(P,Q) \equiv \frac{|\{p_i|(p_i,q_j) \in R_s\}| + |\{q_j|(p_i,q_j) \in R_s\}|}{|P| + |Q|}$$

これは，図 1 のように対応 R_s に含まれる P, Q の要素数を P と Q の総要素数で割ったものである．R_s に関係しない P, Q の要素が増えることによって S は下がる．$R_s = \phi$ では，$S = 0$ となる．また，P と Q が等価である時，$\forall i(p_i, q_i) \in R_s$ となり $S = 1$ となる．

本論文においても，ソフトウェアシステム間の類似度と同様に，二つのソフトウェア部品中で等価な要素数を全要素数で割った値を類似度とする．本論文では，山本らの提案している類似度のメトリクス S_{line} をソフトウェア部品に対し適用する．山本らは複数のファイルからなるソフトウェアシステム間の類似度として S_{line} を提案しているが，ソフトウェア部

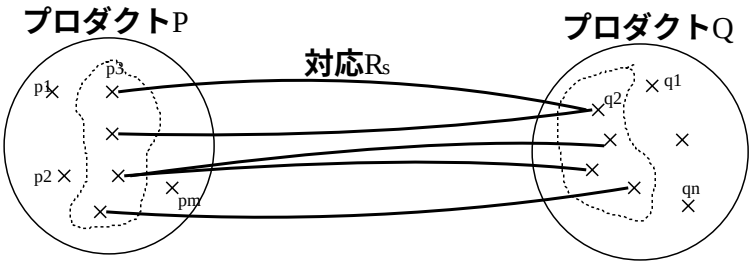

図 1: 要素の対応 R_s

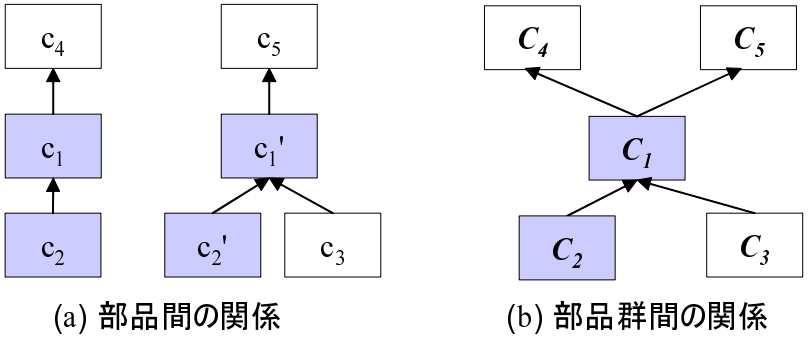

図 2: 類似した部品群

品をソフトウェアシステムに対応させ，要素をソフトウェア部品の行とすることでソフトウェア部品に対しての S_{line} を計測できる．

ソフトウェア部品 P, Q に対し，p_i, q_j を P, Q それぞれの各ファイルの各行とする．直感的には P，Q の各ファイルを連結した一つのファイルをそれぞれ考え，その各行を構成要素とする．等価な行の対応を調べることによって対応 R_s が与えられる．この類似度メトリクスをソフトウェア部品における S_{line} とする．

2.2.2 類似部品群

一般に，部品の集合には，コピーした部品や，コピーして一部変更した部品が多く存在する．そこで，類似した部品をまとめることにより，部品の集合をいくつかの部品群に分類する．以降，類似した部品を集めた部品群を単に部品群と呼ぶ．

図 2 を例に説明する．図 2(a) は部品間の利用関係である．部品 c_1 と c_1'，部品 c_2 と c_2' はそれぞれ類似した部品である．c_3, c_4, c_5 には類似性はない．

部品 c_2 は c_1 を利用し，c_2', c_3 は c_1' を利用しているとする．また，c_1 は c_4 を，c_1' は c_5 をそれぞれ利用しているとする．

図 2(b) では，類似した部品をまとめて部品群としている．部品群に属する部品はそれぞれ $C_1 = \{c_1, c_1'\}$， $C_2 = \{c_2, c_2'\}$， $C_3 = \{c_3\}, C_4 = \{c_4\}$， $C_5 = \{c_5\}$ である．

ある部品群 C_i に属する部品が，他の部品群 C_j に属する部品を利用している場合，その 2 つの部品群間には利用関係があるとする．

例えば，図 2 では，c_1 と c_1' を利用する関係は，C_1 を利用する関係としてまとめる．また，c_1 と c_1' が利用している関係は，C_1 が利用している関係としてまとめる．

上述した考えを用いる場合には，2 つの部品がどの程度類似しているか (類似度 (Similarity)) を定量的に評価する必要がある．そのために部品間の類似度を評価するメトリクスを利用する．まず，類似度に基づいてクラスタ分析を行い n 個の部品の集合を $m(0 \leq m \leq n)$ 個の部品群に分割する．類似度は 0 以上 1 以下の範囲に正規化され，値が高いほど部品は良く類似しているとし，類似度 1 を完全に部品が一致した場合（コピーした部品）とする．基準となる類似度の閾値 $t(0 \leq t \leq 1)$ を与え，部品群間の類似度が t 以下になるように分類する．

2.3 相対的再利用性

2.3.1 従来の再利用性評価

個々の部品の再利用性を評価する手法は多く提案されている．Etzkorn らは, Modularity, Interface Size, Documentation, Complexity の 4 つの視点からオブジェクト指向ソフトウェアの再利用性を評価する手法を提案している [10, 11]．彼らはソースコードから計測される複数のメトリクスを正規化して足し合わせることで再利用性のメトリクスとしている．また，C++を対象として自動的に再利用性メトリクスを計測するシステム「PATRicia」を実装し，適用実験を行なっている．適用実験では，C++で記述されたソースコードを対象とし，そのソースコードに PATRicia を適用して算出した再利用性評価値と，プログラマがソースコードを実際に見て評価した再利用性とを比較し，手法の有効性を検証している．

また，山本らはソースコードが非開示な部品に対して，そのインターフェイスから再利用性を評価する手法を提案している [35]．彼らは理解容易性，利用容易性，テスト容易性，可搬性の 4 つの視点から再利用性メトリクスを定義している．JavaBeans[32] を対象として実際にプログラマがアプリケーションを実装し，使用した部品の再利用性評価値と，実装にかかった時間，コンパイル回数，誤って使用した状態数を比較し，有効性を検証している．

2.3.2 相対的再利用性

従来の再利用性評価手法は全て，部品の構造やインターフェイスなど部品そのものの持つ静的な特性を計算して再利用性を評価するものとなっている．また，それらの有効性の検証にはプログラマによる再利用性評価やプログラマの実装時間などが用いられている．

しかしながら，実際のソフトウェア開発においては，静的な特性からは再利用性が低いと評価されていても，何度も再利用されている部品も存在すると考えられる．たとえば，プログラムの内部構造が複雑で，静的な特性としては再利用性が低くても，開発組織内で長年にわたって利用されている部品であれば頻繁に再利用されるであろう．

そこで本論文では，その部品がどの程度利用されているかという実績に基づいた評価を行うことを目的とする．この再利用性は多数の部品間の利用関係から相対的に決まるものであり，部品の静的特性から決まる再利用性とは区別して「相対的再利用性 (Relative Reusability)」と呼ぶ．

2.3.3 実績に基づく評価手法

実績に基づく評価手法は，計量社会学においては以前から存在し，様々な分野に応用されている．ここでは，他の分野における実績に基づく評価手法を取り上げる．

Influence Weight 論文の引用解析の分野においては，1970 年代には実績に基づいて論文の重要度を評価する手法が提案されている．Narin らは，論文の重要度を論文がどの程度引用されているかという実績に基づいて評価する手法「Influence Weight」を提案している [25, 28, 29]．この手法では，(1) 多くの論文から引用されている論文は重要である，(2) 重要な論文から引用されている論文はまた重要である，という 2 つの考えに基づいて論文の重要度を評価している．

Page Rank 実績に基づく評価手法は，Web ページ検索の分野にも応用されている．Page らは，Web ページの重要度をその Web ページがどの程度リンクされているかという実績に基づいて評価する手法「Page Rank」を提案している [27, 18]．この手法では (1) 多くの Web ページからリンクされている Web ページは重要である，(2) 重要な Web ページからリンクされている Web ページはまた重要である，という 2 つ考えに基づいて Web ページの重要度を評価している．有名な検索サイト「Google」[15] はこの Page Rank による評価を用いた検索サービスを提供している．Google では，検索語の一致によって得られた検索結果を Page Rank の重要度で順位づけして表示し，より正確な検索を可能にしている．

3　提案手法

3.1　概要

ソフトウェア開発者が過去に開発されたソフトウェア部品を利用して新しいソフトウェアを開発する場合を想定する．一般に，開発者は過去に開発されたソフトウェア部品の中で，開発しようとしているソフトウェアに対して再利用性が高いと判断したものを利用する．ここで，開発者が部品を利用するということは，その部品に対して「再利用性が高い」という支持投票をしたとみなす．再利用によるソフトウェア開発が何度も繰り返されれば，再利用性の高い部品は何度も利用され，支持投票数が増加していく．逆に再利用性の低い部品はあまり利用されず，したがって支持投票数は低い値に留まる．このときソフトウェア部品は獲得した票数に応じた再利用性の評価値を持つと考えられるので，以下の式が成り立つ．

$$(\text{部品の評価値}) = (\text{部品への投票数})$$

このとき，単純に獲得票数を数えるだけでなく，どのような部品から利用されたかによって票に重みづけをする．多くの部品から利用されるような優秀な部品（優秀な部品の開発者）に利用されている部品は，再利用性が高いとみなして，同じ一票の支持投票でも，再利用性の評価値の高い部品から投票された場合と，評価値の低い部品から投票された場合では，前者の票の方がより高い重みを持つようにする．

また，ある部品が利用している部品の数も考慮する．ある部品 A が多数の部品を利用している場合には，各利用部品が，A の機能に占める割合が少なくなるので，再利用性の評価は分散してしまうとみなす．つまり，ある部品が複数の部品に投票している時は，票の重みは各利用部品にある配分率をもって配分され，以下の式が成り立つ．

$$(\text{票の重み}) = (\text{投票元の評価値}) \times (\text{投票先に対する配分率})$$

このように，部品の集合において部品同士がお互いに再利用性を評価し，投票しあうことで各部品の評価値が決まっていく．部品が獲得した票の重みの合計値を「相対的再利用性評価値（Value of Relative Reusability）」と呼ぶ．

実際に再利用を繰り返してソフトウェア開発を行う場合，新たに開発したソフトウェアが蓄積されていくため，部品の集合の要素数は増加し，利用関係は変化していく．相対的再利用性評価値は部品の集合における利用関係から求められるため，部品の集合の要素数や利用関係が変化すると，変化前の評価値と変化後の評価値では比較することができない．そこで，評価値そのものではなく，その評価値による部品の順位に着目する．部品の集合の要素数や再利用関係が変化したときも，その変化前後の部品の順位の変動を見ることで，部品の相対的再利用性がどのように変化したのかを理解することができる．

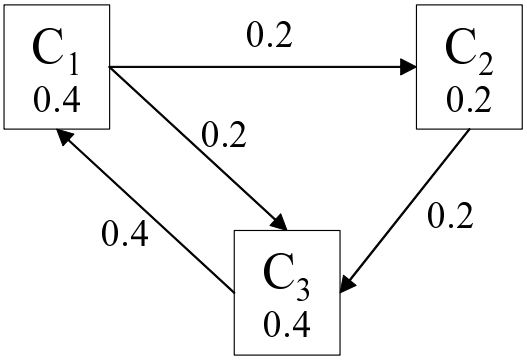

図 3: 再利用性評価の例

2.2.2 で述べたように，実際の部品の集合には多数のコピーや類似部品が存在しているため，提案手法では部品群を部品の単位とし，部品群の相対的再利用性評価値による順位づけを行うことで評価を行う．この手法を「Relative Reusability Ranking 法 (R^3 法)」と呼ぶ．

図 3 を例に説明する．C_1, C_2, C_3 は部品群を表している．利用関係は利用する部品群から利用される部品群への矢印で表している．簡単のため，利用していない部品群への配分率は 0 としている．また，利用している部品には均等に配分する．C_1, C_2, C_3 の評価値を v_1, v_2, v_3 とし，評価値の総和が 1 となるようにしておく．このとき，図 3 の利用関係から各部品群の評価値は $v_1 = 0.4, v_2 = 0.2, v_3 = 0.4$ となる．よって C_1, C_3 は C_2 よりも相対的再利用性が高いと評価される．

3.2 部品の分類

評価対象となる部品が n 個あるとし，それぞれ $c_1, c_2, ..., c_n$ とする．

部品間には方向性を持つ利用関係があり，部品 c_i から c_j への関係を $r(c_i, c_j)$ と表し，

$$\begin{aligned} r(c_i, c_j) \quad = \quad & \textbf{if} \quad (c_i\text{は } c_j\text{を利用している}) \\ & \textbf{then} \quad true \\ & \textbf{else} \quad false \end{aligned}$$

とする．

部品間の類似度を $s(c_i, c_j)$ と表す．類似度は $0 \leq s(c_i, c_j) \leq 1$ に正規化されているとする．

評価対象となる部品全体の集合を

$$C = \{c_1, c_2, ..., c_n\}$$

と表す．部品間の類似度 s に基づいて部品の集合間の類似度が決まるとし，部品の集合 C_i, C_j の類似度を $S(C_i, C_j)$ と表す．類似度は $0 \leq S(C_i, C_j) \leq 1$ に正規化されているとする．

[定義 1] 分類の基準となる類似度の閾値を $t(0 \leq t \leq 1)$ とするとき，部品の集合 C を次の (1)(2) を満たすように分割した部分集合 $C_1, C_2, ..., C_m$ を類似部品群と呼ぶ.

(1)　C_i に属するすべての部品について，類似度は t 以上.

$$\forall c_k, c_l \in C_i \mid s(c_k, c_l) \geq t \tag{1}$$

(2)　異なる集合間の類似度は t より低い.

すなわち，すべての $i, j(1 \leq i, j \leq m)$ について次式が成り立つ.

$$S(C_i, C_j) < t \quad (i \neq j) \tag{2}$$

C を m 個の類似部品群に分割し，それぞれ $C_1, C_2, ..., C_m$ とする．以降，類似部品群を単に部品群と呼ぶ.

部品群間には方向性を持つ利用関係があり，部品群 C_i から C_j への関係を $R(C_i, C_j)$ と表す.

[定義 2] $c_i \in C_i, c_j \in C_j$ とするとき，ある c_i, c_j について c_i から c_j への利用関係があれば，C_i から C_j への利用関係があるとみなし，次式で表す.

$$\begin{aligned} R(C_i, C_j) \quad = \quad & \textbf{if} \quad (\exists c_i, c_j \mid r(c_i, c_j)) \\ & \textbf{then} \quad true \\ & \textbf{else} \quad false \end{aligned} \tag{3}$$

3.3　相対的再利用性評価値の定義

部品群は相対的再利用性評価値を持つとし，部品群 C_i の相対的再利用性評価値を v_i と表す．また，C_i から C_j への利用関係の重みを w_{ij} と表す.

[定義 3] 部品群 C_i の相対的再利用性評価値は，部品群 C_i への利用関係の重み w_{ji} の総和とし，次式で表す.

$$v_i = \sum_{j=1}^{m} w_{ji} \tag{4}$$

部品群 C_i から部品群 C_j への重みの配分率 (Distribution Ratio) を $d_{ij}(0 \leq d_{ij} \leq 1)$ と表す.

[定義 4] 部品群 C_i から C_j への利用関係の重み w_{ij} は，C_i の相対的再利用性評価値を配分率 d_{ij} で配分した値とする.

$$w_{ij} = v_i d_{ij} \tag{5}$$

[定義 5] 部品群 C_i の相対的再利用性評価値はすべての部品 $C_j(1 \leq j \leq m)$ に配分される．

$$\sum_{j=1}^{m} d_{ij} = 1 \tag{6}$$

[定義 6] 利用している部品群への配分率は，利用していない部品群への配分率より高い．すなわち，$R(C_i, C_j) = true, R(C_i, C_k) = false$ のとき，

$$d_{ij} > d_{ik} \tag{7}$$

とする．

3.4 補正

前節までで定義した相対的再利用性評価値を実際のソフトウェア部品にそのまま適用するといくつかの問題が生じるため，若干の補正が必要となる．以下ではその問題点と対策を説明する．

3.4.1 利用していない部品に対する評価

一般に，ソフトウェア開発においては，他の部品を一つも利用せずに開発した部品も存在する．

ある部品群 C_i が他のどの部品も利用していない場合，C_i はどの部品に対しても投票を行っていないことになる．したがってどの部品にも評価値を配分できないと考えて $d_{i0}, d_{i1}, d_{i2}, ..., d_{in}$ をすべて 0 とすると，定義 5 を満たさないことになる．そこで，どの部品にも投票しない場合は「再利用性が非常に低い」という評価をすべての部品に対して投票したと解釈する．

[補正 1] 部品群 C_i がどの部品群も利用していない場合，すべての j について

$$d_{ij} = \frac{1}{m} \tag{8}$$

とする．

3.4.2 評価が全体に循環しない場合

図 4 を例に説明する．ここで，四角は部品群を，矢印は利用関係を表している．図 4 上部では，楕円内の部品群に入る矢印は存在するが，楕円から出ていく矢印は存在しない．従って，再利用性評価の投票はこの楕円内に蓄積され，全体へ評価が循環しないことになる．本提案手法では，部品の集合における票の偏りを分析することで相対的再利用性を評価しているため，部品の集合全体に票が循環しなければ正しい評価を行なうことができない．そこで，部品群を利用しない場合には，非常に低い重みの票を投票すると考える．

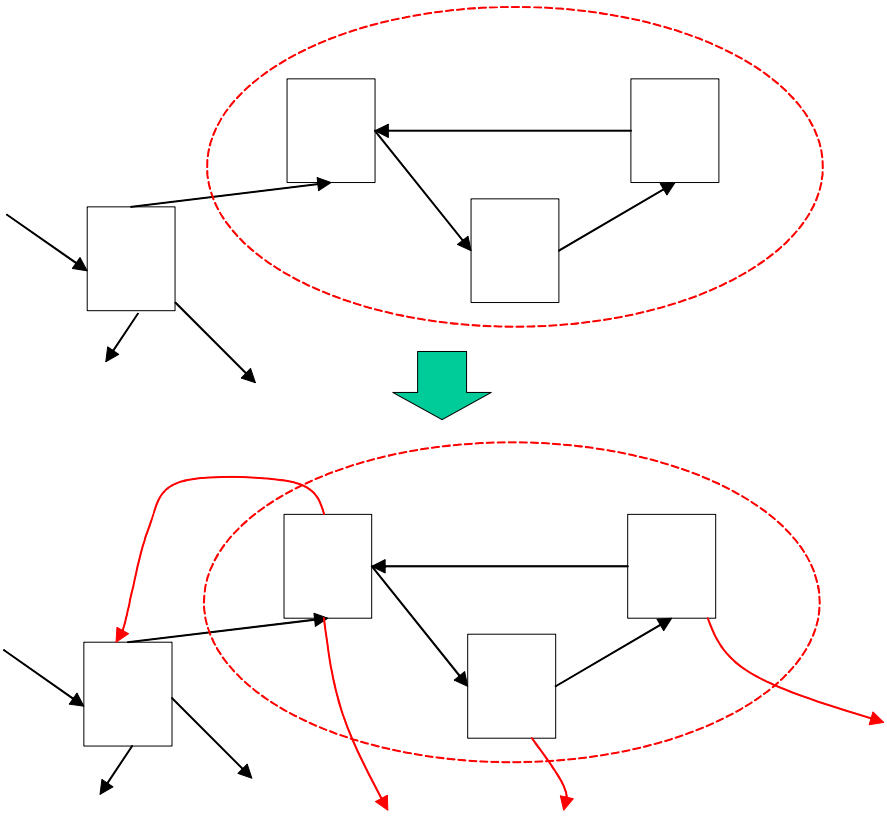

図 4: 評価が全体に循環しない場合

[補正 2] 部品群のもつ評価値のうち $p(0 < p < 1)$ は利用した部品群にのみ配分し，$(1-p)$ はすべての部品に配分する．もとの配分率を d_{ij}，修正後の配分率を d'_{ij} とし，以下のように配分率を修正する．

$$d'_{ij} = pd_{ij} + (1-p)\frac{1}{m} \tag{9}$$

3.5 評価値の計算方法

ここでは，実際に相対的再利用性評価値を求める方法を説明する．

定義 3, 4 より，次式が成り立つ．

$$v_i = \sum_{j=1}^{m} v_j d_{ji} \tag{10}$$

これを $v_i(i = 1, 2, ..., m)$ のすべてについて解けば，すべての部品群の評価値を求めることができる．

すなわち，

$$\begin{aligned} v_1 &= \sum_{j=1}^{m} v_j d_{j1} \\ v_2 &= \sum_{j=1}^{m} v_j d_{j2} \\ &\vdots \\ v_m &= \sum_{j=1}^{m} v_j d_{jm} \end{aligned} \tag{11}$$

の m 個の連立方程式を解けば良い．

これを行列の記法で表す．

m 個の部品群の評価値を表す m 次元列ベクトルを V とする．

$$V = (v_1, v_2, ..., v_m)^t \quad (^t\text{は転置を表す})$$

また，C_i から C_j への配分率を表す $m \times m$ 行列を D とする．

$$D = \begin{pmatrix} d_{11} & d_{12} & ... & d_{1m} \\ d_{21} & d_{22} & ... & d_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & ... & d_{mm} \end{pmatrix}$$

このとき連立方程式 (11) は

$$V = D^t V \tag{12}$$

と表される.

式 (12) を満たすようなベクトル V は行列 D^t の固有値 $\lambda = 1$ の固有ベクトルである.

よって配分率の行列の固有ベクトルを求めることで, 相対的再利用性評価値を決定することができる.

4 再利用性評価システム

4.1 概要

提案した再利用性評価モデルに基づいて，Java ソースコードを対象に相対的再利用性評価システム「R^3-System」を実装した．R^3 法を Java に適用する際の，モデルと Java の概念との対応を表 1 に示す．Java はオブジェクト指向言語であり，クラス単位での利用が行いやすい．また，原則として 1 つのソースファイルには 1 つのクラスを記述する．そこで，Java ソースコードファイルを部品の単位とし，R^3 法を適用する．クラスの継承，インターフェースおよび抽象クラスの実装，メソッドの呼びだしを利用関係とみなす．また，部品間の類似度を評価するメトリクスとして 2 で述べた S_{line} を用いる．S_{line} をソースコードファイルから計測するシステム「SMMT(Similarity Metrics MesuringTool)」も開発されており，本システムでは SMMT を用いて類似度を算出する [34].

4.2 システムの構成

R^3-System の構成図を図 5 に示す．N 個の Java ソースコードファイルを相対的再利用性の評価対象とする．

ファイル間の関係抽出部　Java ソースコードファイルを解析し，クラス間の継承，インターフェイスと抽象クラスの実装，メソッドの呼びだしを利用関係として抽出する．Java ソースコードの構文解析には，ANTLR[2] を利用している．

SMMT　Java ソースコードファイル間の類似度 S_{line} を算出する．

クラスタ分析ツール　SMMT で得られた類似度をもとにクラスタ分析を行い，ファイルの集合を M 個の部品群に分類する．クラスタ分析においては，分類の基準となる閾値 t をパラメータとして与える (定義 1).

部品群間の関係抽出部　クラスタ分析ツールとファイル間の関係抽出部の結果から，部品群間の関係を抽出する．

相対的再利用性計算部　部品群間の関係から，R^3 法で部品群の相対的再利用性評価値による順位づけを行う．なお，行列の固有値計算には，Java で行列演算を行うパッケージ JAMA[21] を利用している．

部品群順位→ファイル順位変換部　部品群の相対的再利用性順位を，ファイルの順位に変換する．

表 1: Java への適用

R^3 法のモデル	Java
部品	Java ソースファイル
類似度	S_{line}
利用関係	継承，実装，呼出

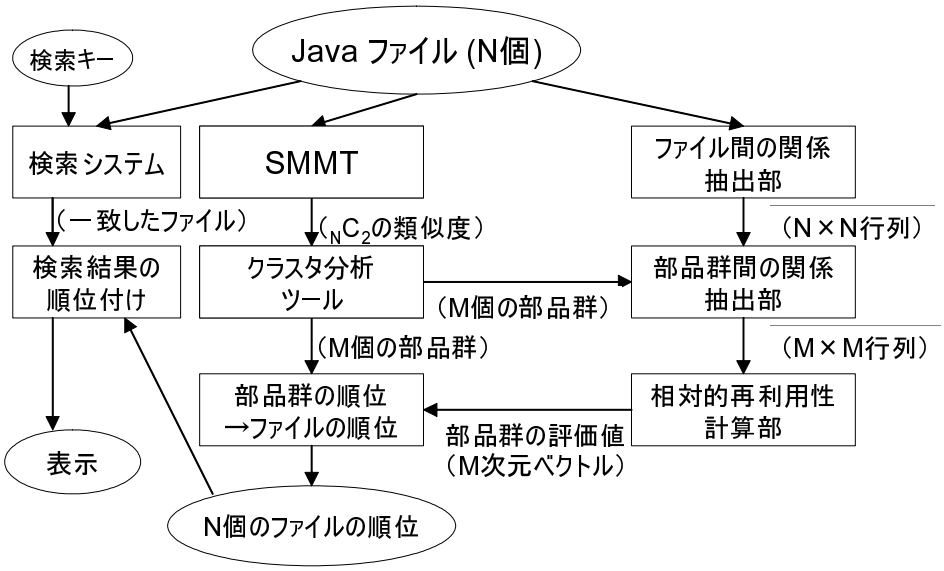

図 5: R^3 システムの構成図

検索システム　検索キーが入力され，検索キーに一致する部分を持つソースコードファイルを返す.

検索結果の順位づけ部　相対的再利用性の順位をもとに，検索結果を順位づけして表示する.

表 2: JDK-1.3.0 への適用結果

順位	クラス名	評価値
1	java.lang.Object	0.16126
2	java.lang.Class	0.08712
3	java.lang.Throwable	0.05510
4	java.lang.Exception	0.03103
5	java.io.IOException	0.01343
6	java.lang.StringBuffer	0.01214
7	java.lang.SecurityManager	0.01169
8	java.io.InputStream	0.01027
9	java.lang.reflect.Field	0.00948
10	java.lang.reflect.Constructor	0.00936
⋮	⋮	⋮
1256	com.sun.image.codec.jpeg.TruncatedFileException	0.00011
1256	sunw.util.EventObject	0.00011
⋮	⋮	⋮
1256	sunw.io.Serializable	0.00011
1256	sunw.util.EventListener	0.00011

5 適用実験

本提案手法では，3. で説明したように再利用性評価値の順位によって各部品の相対的再利用性を評価している．そこで，部品の順位が利用実績を反映していることを示すために，実際のソフトウェア部品に対して R^3 法を適用する実験を行った.

実際に Java ソースコードに R^3 システムを適用した場合の適用例を以下に示す．評価対象として JDK-1.3.0 と我々の研究室内で開発された複数の Java アプリケーションの 2 つを選んだ．調整パラメータとして，定義 1 で述べた部品の分類における類似度の閾値を $t = 0.80$, 補正 2 で述べた票の重みの比率を $p = 0.85$ とした.

また，適正な閾値の範囲を調べるため，JDK-1.3.0 を対象としてパラメータの調整を行なった.

5.1 JDK-1.3.0 への適用

ここでは，評価対象として Sun Microsystems [31] から配布されている Java 2 Software Development Kit, Standard Edition 1.3.0 (以下 JDK) のソースコード (1877 ファイル) を対象とした．JDK へ適用して得られた結果の一部を表 2 に示す.

JDK は Java の基本パッケージであり，Java でアプリケーションを開発するときには JDK が必要となる．相対的再利用性の上位 10 クラスについて見てみると，Object, Class, Throwable など，Java の言語仕様 [16, 17] で利用しなければならないクラスが大半を占めている．Java 言語仕様によれば，java.lang.Object クラスはすべてのクラスのスーパークラスである．したがってすべてのクラスから利用されていることになり，相対的再利用性が 1 位となっている．また，java.lang.Class クラスは実行中のクラスおよびインターフェイスを表すクラスで，このクラスを直接継承するようなクラスはないが，実行時のオブジェクト型の情報を取得するために頻繁に呼び出される．java.lang.Throwable クラスはすべてのエラーと例外のスーパークラスであり，したがって例外やエラーを扱うクラスはすべてこのクラスを間接的に利用していることになる．このように，実際に直接的，間接的に利用される回数の多いクラスが上位を占めている．

最下位は 1256 位で，該当するクラスは 622 あった．これらのクラスは，どのクラスにも利用されていない.

5.2 研究室内ソースコードへの適用

ここでは研究室内で過去に開発された Java アプリケーションのソースコードを適用対象とした (582 ファイル).

適用対象について簡単に説明する.

C-K メトリクス計測ツール 1　Java ソースコードから C-K メトリクス [8] を計測するツール．パッケージ名は cktool．(29 ファイル) ANTLR[2] を使用してソースコードの構文解析を行っている.

C-K メトリクス計測ツール 2　上のツールを若干修正したもの．上と同じく ANTLR を用いている．パッケージ名は cktool_new．(29 ファイル)

R^3-System　本論文で作成した R^3-System．ソースコードの構文解析には ANTLR を使用している．また，行列計算には JAMA[21] を使用している. そのほか Caffe Cappuccino Class Library[26] というライブラリも使用している．パッケージ名は jp.ac.osaka_u.es.ics.iip_lab.metrics．(68 ファイル)

表 3: 研究室内ソースコードへの適用結果

順位	クラス名	評価値
1	antlr.Token	0.10727
2	antlr.debug.Event	0.06189
2	antlr.debug.NewLineEvent	0.06189
4	antlr.collections.impl.Vector	0.05434
5	jp.gr.java_conf.keisuken.text.html.HtmlParameter	0.05246
6	jp.gr.java_conf.keisuken.net.server.ServerProperties	0.03699
7	Jama.Matrix	0.01564
8	jp.gr.java_conf.keisuken.util.IntegerArray	0.01390
8	jp.gr.java_conf.keisuken.util.LongArray	0.01390
10	jp.ac.osaka_u.es.ics.iip_lab.metrics.parser.IdentifierInfo	0.01365
⋮	⋮	⋮
418	jp.gr.java_conf.keisuken.util.PropertiesSerializer	0.00050
418	cktool_new.examples.Main	0.00050
⋮	⋮	⋮
418	cktool_new.examples.WriteLog	0.00050
418	cktool.examples.WriteLog	0.00050

使用パッケージのソース　ANTLR(パッケージ名 antlr, 188 ファイル), JAMA(パッケージ名 Jama, 9 ファイル), Caffe Cappuccino Class Library(パッケージ名 jp.gr.java_conf.keisuken, 245 ファイル) のソースコードも適用対象とした.

その他　(14 ファイル)

適用結果を表 3 に示す.

研究室内で作成されたアプリケーションよりも，利用したパッケージの方が上位に来る傾向が見られる．また，主に機能を提供するようなクラスよりも，データを格納するようなクラスの方が高く評価される傾向にある．全体的に複数のアプリケーションで共通に使用されている ANTLR のクラスが高く評価される傾向にある．

これらの結果から，相対的再利用性評価値による順位づけは，実際の利用実績の傾向を反映していると考えられる．

5.3　パラメータの調整

ここでは相対的再利用性評価値に影響を与えるパラメータの調整を行なう．5.1 で用いた JDK-1.3.0 のソースコードを評価対象とし，3.2 で述べた分類における類似度の閾値 t と，3.4.2 で述べた利用した部品群のみへの配分と全体に対する配分との比率 p の 2 つのパラメータをを変化させながら相対的再利用性評価値を求める．

5.3.1　比率 p の調整

利用した部品群への配分の比率 p の変化と相対的再利用性評価値の変化との関係を調べるため，分類における類似度の閾値を $t = 0.80$ と固定して p を変化させながら相対的再利用性評価値を計測する．p は 0.05 から 0.10 間隔で 0.95 まで変化させる．p の値が大きいほど利用した部品群への配分率と利用していない部品群への配分率の差が大きくなる．また，分類の閾値を固定しているため，部品群に属する部品や，部品群間の関係は一定である．

評価値の計測結果を表 4 に，その順位を表 5 にそれぞれ示す．表 4, 5 は，共に $p = 0.85$ での評価値，順位で整列している．ここでは上位 10 位と一部のクラスについてのみ表示している．また，p の値ごとの評価値についての Pearson の相関係数を表 6 に，Spearman の順位相関係数を表 7 に示す.

表 4 から，上位にあるクラスについては p が大きくなるにつれて評価値が高くなり，逆に下位のクラスについては p が大きくなるにつれて評価値が低くなるという傾向が見られる．この様子をグラフにしたものが図 6 である．横軸は p の値，縦軸は評価値として変化の様子を示している．縦軸の目盛は常用対数で表示している．p を大きくするにつれて，各クラス

間の評価値の差が大きくなる傾向が見られる．評価値の相関係数は全体的に高く，p の差が 0.10 の場合には，相関係数は 1 に非常に近くなっている．表 7 から順位相関係数がほぼ 1 であり，p の値が変化しても，全体的には順位の変動が少ないことが分かる．

また，順位については常に Object クラスが 1 位であり，p の値に関わらず直接利用されているクラスは高く評価されることが分かる．これは，補正 2 における修正によって得られた配分率 d'_{ij} も定義 6 の不等式を満たし，従って修正後の配分率においても利用した部品群への配分率は利用していない部品群よりも大きいためであると考えられる．

一方，Throwable に着目すると，p の値が小さくなるにつれて順位が下がっている．逆に，Exception に着目すると p の値が小さくなるにつれて順位が上がっている．Exception は他のクラスから直接利用されることが多い．また，Exception は Throwable を利用しているため，Throwable は間接的に利用されることが多いことになる．p の値が大きい時には Exception クラスの評価値（票の重み）を多く受けとるため，順位が Exception より高く，逆に p の値が低い場合には，Exception クラスから受けとる評価値（票の重み）が小さくなるため，順位が Exception より低くなっている．このことから，p が小さい場合は直接利用される数のみが重視され，p が大きい場合は直接の利用だけでなく間接的な利用も重視されることが分かる．

5.3.2 閾値 t の調整

ここでは分類における類似度の閾値 t の変化と相対的再利用性評価値の変化との関係を調べるため，利用した部品への配分の比率を $p = 0.85$ と固定して t を変化させながら相対的再利用性評価値を計測する．t は 0.10 から 0.10 間隔で 1.00 まで変化させる．t の値が大きいほど類似部品群内の部品同士の類似度が高くなり，$t = 1.00$ では，内容が完全に一致する部品（コピーした部品）のみで構成される類似部品群になる．逆に t を 0 に近くすると，あまり類似していない部品も同一の類似部品群にまとめられる．$t = 0$ のときは，すべての部品が 1 つの類似部品群としてまとめられるため，相対的再利用性評価値を求めることができない．

評価値の計測結果を表 8 に，その順位を表 10 にそれぞれ示す．表 8, 10 は，共に $t = 0.80$ での評価値，順位で整列している．前節と同様，上位 10 位と一部のクラスについてのみ表示している．

上位 10 位については，Object クラスについてはほぼ変化はないが，$t = 0.70$ と $t = 0.80$ を境に Throwable と Exception の順位が大幅に入れ替わっている．表 9 に $t = 0.70$ における順位と評価値を示す．$t = 0.70$ で分類すると，Exception クラスを含めて 62 個の例外クラスが 1 つの類似部品群にまとまっている．そのため，Throwable や StringBuffer など t が 0.80 以上の時には上位にあったクラスが 65 位以下に落ちている．この変化は相関係数にも現れている．t の値ごとの評価値についての Pearson の相関係数を表 11 に示す．$t = 0.10$ から 0.70

表 4: p と評価値の変化 ($t = 0.80$)

クラス名	p=0.05	p=0.15	p=0.25	p=0.35	p=0.45	p=0.55	p=0.65	p=0.75	p=0.85	p=0.95
java.lang.Object	0.004046	0.012136	0.022078	0.034351	0.049591	0.068623	0.092513	0.122655	0.160853	0.206475
java.lang.Class	0.001104	0.002737	0.005404	0.009528	0.015716	0.024859	0.038271	0.057940	0.086926	0.128235
java.lang.Throwable	0.000862	0.001995	0.004114	0.007526	0.012549	0.019497	0.028649	0.040228	0.054363	0.070018
java.lang.Exception	0.001699	0.004303	0.007378	0.010875	0.014713	0.018771	0.022903	0.026947	0.030732	0.033599
java.io.IOException	0.001081	0.002193	0.003409	0.004690	0.006015	0.007396	0.008910	0.010755	0.013341	0.017220
java.lang.StringBuffer	0.000953	0.001734	0.002563	0.003444	0.004400	0.005487	0.006843	0.008792	0.012071	0.018100
java.lang.SecurityManager	0.000747	0.001099	0.001493	0.001972	0.002609	0.003537	0.004995	0.007428	0.011683	0.019161
java.io.InputStream	0.001000	0.001809	0.002583	0.003347	0.004143	0.005048	0.006192	0.007794	0.010215	0.013859
java.lang.reflect.Field	0.000570	0.000577	0.000644	0.000824	0.001206	0.001933	0.003243	0.005531	0.009461	0.015944
java.lang.reflect.Constructor	0.000580	0.000603	0.000680	0.000859	0.001223	0.001909	0.003157	0.005384	0.009343	0.016197
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.image.ImageObserver	0.000598	0.000659	0.000752	0.000872	0.001012	0.001157	0.001278	0.001321	0.001181	0.000630
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.JOptionPane	0.000576	0.000566	0.000549	0.000523	0.000488	0.000443	0.000384	0.000309	0.000214	0.000087
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.color.ICC_ColorSpace	0.000560	0.000521	0.000479	0.000435	0.000387	0.000335	0.000279	0.000215	0.000143	0.000056
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.plaf.metal.MetalSplitPaneDivider	0.000553	0.000501	0.000450	0.000397	0.000344	0.000289	0.000233	0.000175	0.000112	0.000043

表 5: p と順位の変化 ($t = 0.80$)

クラス名	p=0.05	p=0.15	p=0.25	p=0.35	p=0.45	p=0.55	p=0.65	p=0.75	p=0.85	p=0.95
java.lang.Object	1	1	1	1	1	1	1	1	1	1
java.lang.Class	9	4	3	3	2	2	2	2	2	2
java.lang.Throwable	22	11	5	4	4	3	3	3	3	3
java.lang.Exception	2	2	2	2	3	4	4	4	4	4
java.io.IOException	10	7	7	7	6	6	5	5	5	8
java.lang.StringBuffer	16	15	15	14	13	11	8	6	6	7
java.lang.SecurityManager	34	34	33	29	25	21	16	8	7	6
java.io.InputStream	13	14	14	15	15	14	10	7	8	13
java.lang.reflect.Field	488	366	226	137	89	56	26	17	9	11
java.lang.reflect.Constructor	368	272	177	120	85	57	28	18	10	10
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.image.ImageObserver	200	157	136	118	119	105	100	98	103	135
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.JOptionPane	400	418	433	449	457	472	493	493	499	519
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.color.ICC_ColorSpace	800	810	818	828	836	844	850	853	850	856
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.plaf.metal.MetalSplitPaneDivider	1253	1253	1253	1253	1253	1253	1253	1253	1253	1253

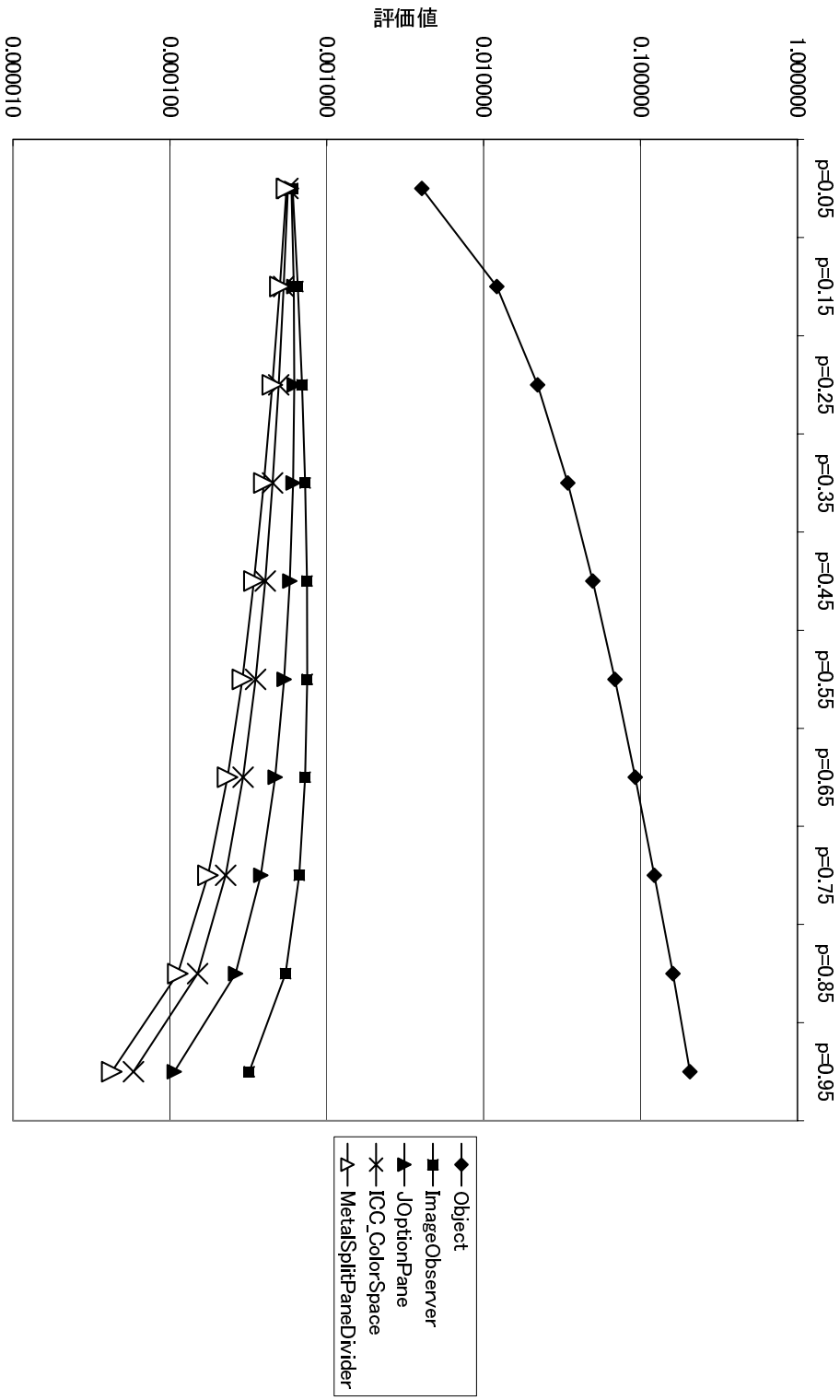
評価値
1.000000
0.100000
0.010000
0.001000
0.000100
0.000010
p=0.05
p=0.15
p=0.25
p=0.35
p=0.45
p=0.55
p=0.65
p=0.75
p=0.85
p=0.95
Object
ImageObserver
JOptionPane
ICC_ColorSpace
MetalSplitPaneDivider

図 6: p と評価値の変化

表 6: 評価値の相関 ($t = 0.80$)

	p=0.05	p=0.15	p=0.25	p=0.35	p=0.45	p=0.55	p=0.65	p=0.75	p=0.85	p=0.95
p=0.05	1.000	0.997	0.988	0.973	0.952	0.926	0.894	0.857	0.815	0.762
p=0.15	0.997	1.000	0.997	0.988	0.972	0.951	0.924	0.892	0.853	0.803
p=0.25	0.988	0.997	1.000	0.997	0.987	0.972	0.950	0.923	0.888	0.841
p=0.35	0.973	0.988	0.997	1.000	0.997	0.987	0.971	0.949	0.920	0.877
p=0.45	0.952	0.972	0.987	0.997	1.000	0.997	0.987	0.971	0.947	0.909
p=0.55	0.926	0.951	0.972	0.987	0.997	1.000	0.997	0.987	0.969	0.936
p=0.65	0.894	0.924	0.950	0.971	0.987	0.997	1.000	0.997	0.985	0.959
p=0.75	0.857	0.892	0.923	0.949	0.971	0.987	0.997	1.000	0.996	0.978
p=0.85	0.815	0.853	0.888	0.920	0.947	0.969	0.985	0.996	1.000	0.991
p=0.95	0.762	0.803	0.841	0.877	0.909	0.936	0.959	0.978	0.991	1.000

表 7: 順位の相関 ($t = 0.80$)

	p=0.05	p=0.15	p=0.25	p=0.35	p=0.45	p=0.55	p=0.65	p=0.75	p=0.85	p=0.95
p=0.05	1.000	0.998	0.996	0.994	0.992	0.989	0.985	0.980	0.975	0.967
p=0.15	0.998	1.000	0.999	0.998	0.996	0.994	0.991	0.987	0.982	0.975
p=0.25	0.996	0.999	1.000	1.000	0.998	0.997	0.994	0.991	0.987	0.980
p=0.35	0.994	0.998	1.000	1.000	1.000	0.999	0.997	0.994	0.991	0.984
p=0.45	0.992	0.996	0.998	1.000	1.000	1.000	0.998	0.997	0.994	0.988
p=0.55	0.989	0.994	0.997	0.999	1.000	1.000	1.000	0.998	0.996	0.991
p=0.65	0.985	0.991	0.994	0.997	0.998	1.000	1.000	1.000	0.998	0.994
p=0.75	0.980	0.987	0.991	0.994	0.997	0.998	1.000	1.000	0.999	0.996
p=0.85	0.975	0.982	0.987	0.991	0.994	0.996	0.998	0.999	1.000	0.999
p=0.95	0.967	0.975	0.980	0.984	0.988	0.991	0.994	0.996	0.999	1.000

の間および $t = 0.80$ から 1.00 の間では相関係数は 0.99 以上であるが，$t = 0.70$ と $t = 0.80$ の境をまたぐ場合は相関係数が 0.45 以下になっている．このことから，$t = 0.70$ と $t = 0.80$ を境として評価が分かれていることが分かる．

t の変化によって評価値がどのように変化するかをグラフにしたものが図 7 である．横軸は t の値，縦軸は評価値として変化の様子を示している．縦軸の目盛は常用対数で表示している．NoSuchFieldException は Exception のサブクラスである．グラフからも，$t = 0.70$ と $t = 0.80$ を境に例外クラスの評価値が大きく変化していることが明らかである．例外クラス以外では，$t = 0.10$ から $t = 0.90$ の範囲では大きな変化は見られない．

t の値ごとの Spearman の順位相関係数を表 12 に示す．順位相関係数は 0.70 以上と全体的に高く，t の差が 0.10 の場合には，順位相関係数は 1 に近くなっている．上述した Exception についての問題はあるが，t が変化しても全体的な順位の傾向はあまり変動しないことが分かる．

$t = 0.70$ 以下で分類するかどうか，すなわち例外クラスをすべて同一の類似部品群として扱うかどうかは，対象とする部品のドメインや評価の目的によって適宜判断する必要がある．また例外クラスを評価対象に入れるかどうかも本提案手法を適用する際には検討すべきであると考えられる．

表 8: t と評価値の変化 ($p = 0.85$)

クラス名	t=0.10	t=0.20	t=0.30	t=0.40	t=0.50	t=0.60	t=0.70	t=0.80	t=0.90	t=1.00
java.lang.Object	0.156049	0.156342	0.157200	0.156984	0.154787	0.154066	0.153380	0.160853	0.156856	0.156509
java.lang.Class	0.084910	0.085542	0.085879	0.085655	0.084420	0.084011	0.083642	0.086926	0.084637	0.084425
java.lang.Throwable	0.028833	0.029417	0.029003	0.028849	0.028544	0.028423	0.028565	0.054363	0.054886	0.054532
java.lang.Exception	0.051902	0.052988	0.052375	0.051997	0.051505	0.051336	0.051730	0.030732	0.030819	0.030642
java.io.IOException	0.051902	0.052988	0.052375	0.051997	0.051505	0.051336	0.051730	0.013341	0.013319	0.013227
java.lang.StringBuffer	0.012963	0.012840	0.012783	0.012583	0.012578	0.012498	0.012420	0.012071	0.011598	0.011545
java.lang.SecurityManager	0.011393	0.012142	0.012138	0.012106	0.011759	0.011695	0.011634	0.011683	0.011313	0.011280
java.io.InputStream	0.010590	0.010597	0.010506	0.010428	0.010349	0.010309	0.010282	0.010215	0.010178	0.010125
java.lang.reflect.Field	0.009328	0.009381	0.009408	0.009390	0.009258	0.009210	0.009167	0.009461	0.009202	0.009177
java.lang.reflect.Constructor	0.009226	0.009266	0.009286	0.009259	0.009128	0.009081	0.009037	0.009343	0.009087	0.009061
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.image.ImageObserver	0.000804	0.001210	0.001271	0.001282	0.001288	0.001294	0.001290	0.001181	0.001121	0.001153
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.JOptionPane	0.000242	0.000237	0.000229	0.000240	0.000238	0.000235	0.000233	0.000214	0.000200	0.000198
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.color.ICC_ColorSpace	0.000185	0.000169	0.000163	0.000160	0.000159	0.000157	0.000156	0.000143	0.000134	0.000132
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.plaf.metal.MetalSplitPaneDivider	0.000146	0.000133	0.000128	0.000126	0.000125	0.000124	0.000122	0.000112	0.000105	0.000104

表 9: $t = 0.70$ での順位と評価値 ($p = 0.85$)

順位	クラス名	評価値
1	java.lang.Object	0.153380
2	java.lang.Class	0.083642
3	java.lang.Exception	0.051730
3	java.lang.Error	0.051730
3	java.io.EOFException	0.051730
3	java.io.FileNotFoundException	0.051730
3	java.io.IOException	0.051730
⋮	⋮	⋮
3	java.util.NoSuchElementException	0.051730
3	java.util.TooManyListenersException	0.051730
3	javax.naming.NoInitialContextException	0.051730
3	javax.naming.OperationNotSupportedException	0.051730
65	java.lang.Throwable	0.028565
66	javax.naming.NamingException	0.015063
67	java.lang.StringBuffer	0.012420
68	java.lang.SecurityManager	0.011634
69	java.lang.VirtualMachineError	0.011115
70	java.io.InputStream	0.010282
⋮	⋮	⋮

表 10: t と順位の変化 ($p = 0.85$)

クラス名	t=0.10	t=0.20	t=0.30	t=0.40	t=0.50	t=0.60	t=0.70	t=0.80	t=0.90	t=1.00
java.lang.Object	1	1	1	1	1	1	1	1	1	1
java.lang.Class	2	2	2	2	2	2	2	2	2	2
java.lang.Throwable	66	66	66	66	66	66	65	3	3	3
java.lang.Exception	3	3	3	3	3	3	3	4	4	4
java.io.IOException	3	3	3	3	3	3	3	5	6	6
java.lang.StringBuffer	68	68	68	68	68	68	67	6	7	7
java.lang.SecurityManager	69	69	69	69	69	69	68	7	8	8
java.io.InputStream	72	71	71	71	71	71	70	8	9	9
java.lang.reflect.Field	73	72	72	72	72	72	71	9	10	10
java.lang.reflect.Constructor	74	73	73	73	73	73	72	10	11	11
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.image.ImageObserver	316	191	172	172	166	166	163	103	101	94
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.JOptionPane	832	671	637	578	563	562	561	499	496	497
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
java.awt.color.ICC_ColorSpace	1046	961	941	931	923	917	914	850	828	833
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
javax.swing.plaf.metal.MetalSplitPaneDivider	1403	1343	1326	1318	1315	1314	1306	1253	1229	1228

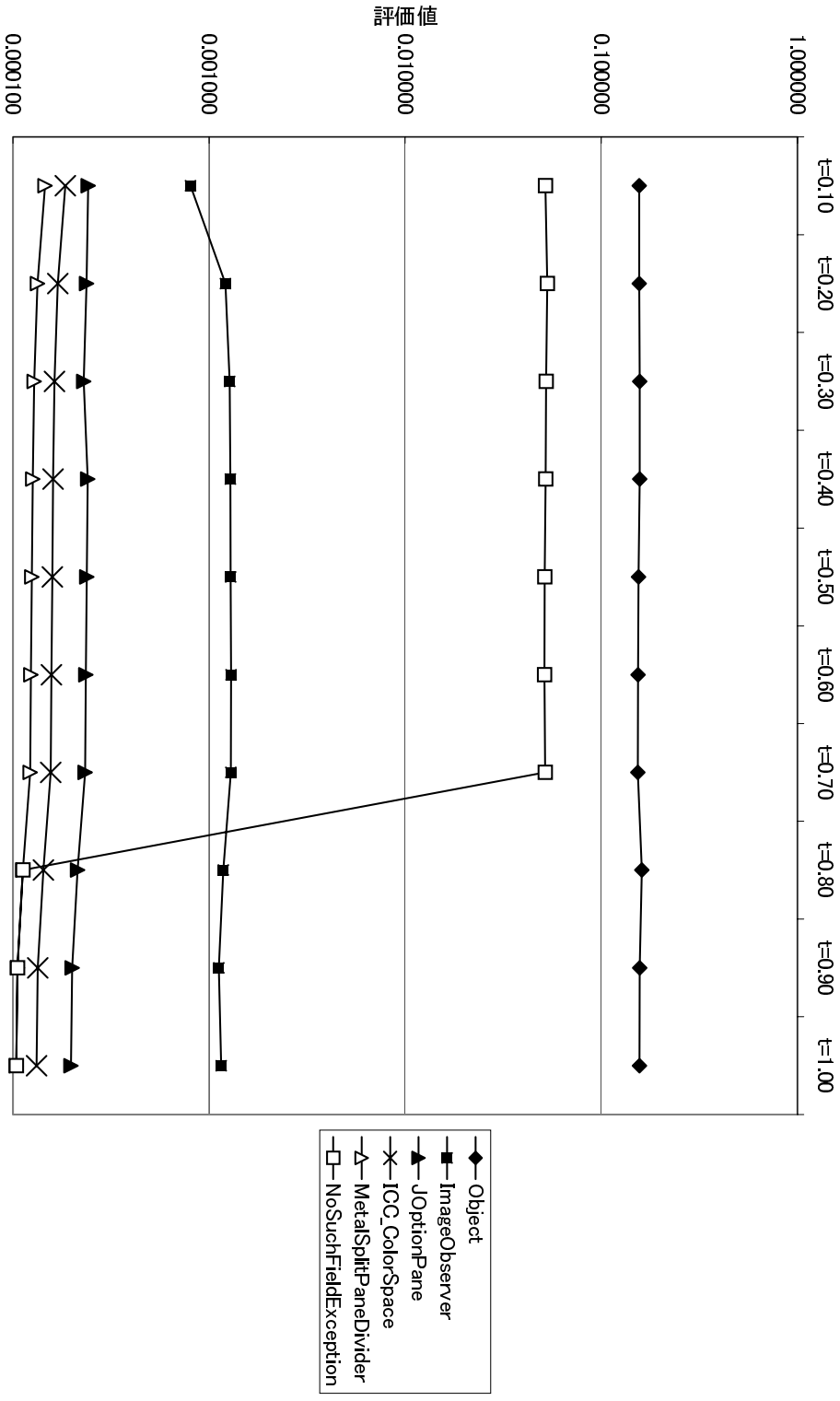
評価値
1.000000
0.100000
0.010000
0.001000
0.000100
t=0.10
t=0.20
t=0.30
t=0.40
t=0.50
t=0.60
t=0.70
t=0.80
t=0.90
t=1.00
Object
ImageObserver
JOptionPane
ICC_ColorSpace
MetalSplitPaneDivider
NoSuchFieldException

図 7: t と評価値の変化

表 11: 評価値の相関 ($p = 0.85$)

	t=0.10	t=0.20	t=0.30	t=0.40	t=0.50	t=0.60	t=0.70	t=0.80	t=0.90	t=1.00
t=0.10	1.000	0.998	0.998	0.998	0.998	0.998	0.991	0.433	0.428	0.428
t=0.20	0.998	1.000	0.999	0.999	0.999	0.999	0.992	0.428	0.423	0.423
t=0.30	0.998	0.999	1.000	1.000	1.000	1.000	0.993	0.434	0.429	0.428
t=0.40	0.998	0.999	1.000	1.000	1.000	1.000	0.993	0.436	0.431	0.430
t=0.50	0.998	0.999	1.000	1.000	1.000	1.000	0.993	0.435	0.430	0.429
t=0.60	0.998	0.999	1.000	1.000	1.000	1.000	0.993	0.435	0.429	0.429
t=0.70	0.991	0.992	0.993	0.993	0.993	0.993	1.000	0.434	0.428	0.428
t=0.80	0.433	0.428	0.434	0.436	0.435	0.435	0.434	1.000	0.996	0.995
t=0.90	0.428	0.423	0.429	0.431	0.430	0.429	0.428	0.996	1.000	0.998
t=1.00	0.428	0.423	0.428	0.430	0.429	0.429	0.428	0.995	0.998	1.000

表 12: 順位の相関 ($p = 0.85$)

	t=0.10	t=0.20	t=0.30	t=0.40	t=0.50	t=0.60	t=0.70	t=0.80	t=0.90	t=1.00
t=0.10	1.000	0.880	0.849	0.837	0.830	0.828	0.823	0.730	0.713	0.710
t=0.20	0.880	1.000	0.968	0.958	0.951	0.949	0.943	0.845	0.824	0.821
t=0.30	0.849	0.968	1.000	0.991	0.984	0.982	0.976	0.877	0.855	0.852
t=0.40	0.837	0.958	0.991	1.000	0.993	0.991	0.985	0.886	0.863	0.861
t=0.50	0.830	0.951	0.984	0.993	1.000	0.998	0.995	0.895	0.872	0.870
t=0.60	0.828	0.949	0.982	0.991	0.998	1.000	0.996	0.897	0.874	0.871
t=0.70	0.823	0.943	0.976	0.985	0.995	0.996	1.000	0.901	0.878	0.875
t=0.80	0.730	0.845	0.877	0.886	0.895	0.897	0.901	1.000	0.976	0.973
t=0.90	0.713	0.824	0.855	0.863	0.872	0.874	0.878	0.976	1.000	0.997
t=1.00	0.710	0.821	0.852	0.861	0.870	0.871	0.875	0.973	0.997	1.000

6 まとめ

本論文では，ソフトウェア部品の相対的再利用性を定義し，その評価方法として R^3 法を提案した．実際に R^3 法に基づいて相対的再利用性を評価するシステム R^3-System を実装し，JDK のソースコードファイルに対して適用した．適用結果では，実際に利用されている回数の多いクラスが上位を占め，本手法の有効性が確認された．

再利用によるソフトウェア開発において，多くのソフトウェア部品の中から開発者が本当に必要としているものを見つけ出すのは困難な作業であるが，再利用する部品を選択する基準として R^3 法を用いることで開発者の負担を軽減できると考えられる．

今後の課題としては以下のような点が挙げられる．

Java における継承，実装，メソッド呼びだしの各関係の重みづけを検討する 本論文では Java への適用の際，簡単のためすべての関係を利用関係として区別せず取り扱っている．関係の種類によって，重みづけを変更することも検討する．

Java における例外クラスの取り扱いを検討する 5.3 で議論した通り，例外クラスを 1 つの類似部品群としてまとめるかどうかによって評価値が大きく変化する．Java を対象に本提案手法を適用する際に例外クラスを類似部品群にまとめるか，あるいは評価対象から除外するかなど検討する．

多くの部品に対して R^3-System を適用する 5 で行った適用実験では JDK や研究室内のソースコードを対象に適用している．今後多くの部品を収集し，大規模な部品の集合に対する R^3 法の有効性を検証する．

部品検索システムの検討と実装を行う 再利用を用いたソフトウェア開発を支援するために，部品検索の様々な手法が提案されている [24, 33]．R^3 法により，部品の検索結果を順位づけして表示することで，部品検索の精度を向上できると考えられる．

謝辞

本研究において，常に適切な御指導および御助言を賜わりました大阪大学大学院基礎工学研究科情報数理系専攻 井上 克郎 教授に心より深く感謝致します.

本研究において，常に適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 楠本 真二 助教授に深く感謝致します.

本研究において，適切な御指導および御助言を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 松下 誠 助手に深く感謝致します.

R^3-System の開発において御協力を頂きました大阪大学大学院基礎工学研究科情報数理系専攻 山本 哲男 様，横森 励士 様に深く感謝致します.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院基礎工学研究科情報数理系専攻井上研究室の皆様に深く感謝致します.

参考文献

[1] 青山, 中所, 向山: コンポーネントウェア, 共立出版, (1998).

[2] “ANTLR Website,” http://www.antlr.org/

[3] 馬場: “Google の秘密 - PageRank 徹底解説,” http://www.kusastro.kyoto-u.ac.jp/˜baba/wais/pagerank.htm

[4] V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page and S. Waligora: “The software engineering laboratory - an operational software experience,” *Proc. of ICSE14*, pp. 370-381 (1992).

[5] I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier, “Clone detection using abstract syntax trees,” *Proc. of the International Conference on Software Maintenance*, pp. 368–378, (1998).

[6] G. Booch: Object-Oriented Analysis and Design with Applications,*The Benjamin/Cummings Publishing* (1994).

[7] C. Braun: Reuse, in John J. Marciniak, editor, *Encyclopedia of Software Engineering*, Vol. 2, John Wiley & Sons, pp. 1055-1069 (1994).

[8] S. R. Chidamber and C. F. Kemerer: “A metrics suite for object-oriented design,” *IEEE Trans. on Software Eng.*, Vol.20, No.6, pp.476-493 (1994).

[9] G. Caldiera, G. Antoniol, R. Fiutem and C. Lokan: “Definition and experimental evaluation of function points for object-oriented systems”, *IEEE, Proc. of METRICS98*, pp.167-178 (1998).

[10] L. H. Etzkorn, W. E. Huges Jr., C. G. Davis: “Automated reusability quality analysis of OO legacy software,” *Information and Software Technology*, Vol. 43, Issue 5, pp. 295-308 (2001).

[11] L. H. Etzkorn, J. Bansiya, C. G. Davis: “Design and code complexity metrics for OO classes”, *Journal of Object-Oriented Programming*, Vol. 12, No. 1, pp. 35-40, (1999).

[12] 藤原, 今川, 楠本, 井上, 大坪, 湯浦: “特定アプリケーション開発におけるフレームワークの評価に関する考察”, 情報処理学会第６２回全国大会講演論文集 (分冊１), pp. 283-284, (2001)

[13] 藤原, 楠本, 井上, 大坪, 湯浦: “複雑度と機能量に基づくアプリケーションフレームワークの実験的評価”, オブジェクト指向最前線 2001, pp.85-90, (2001).

[14] H. Fujiwara, S. Kusumoto, K. Inoue, T. Ootsubo and K. Yuura: “Evaluation of a Business Application Framework Using Complexity and Functionality Metrics”, *3rd International Conference on Product Focused Software Process Improvement(PROFES2001)*, pp 371-380, (2001)

[15] “Google,” http://www.google.com/

[16] J. Gosling, B. Joy, G. Steele and G. Bracha: “The Java Language Spwcification Second Edition,” Sun Microsystems, http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

[17] J. ゴズリン, B. ジョイ, G. スティール, G. ブラーハ, 村上 (訳): Java 言語仕様 第 2 版, ピアソン・エデュケーション, (2000).

[18] T. H. Haveliwala, “Efficient Computation of PageRank”, Stanford Technical Report, (1999).

[19] S. Isoda: “Experience report on a software reuse project: Its structure, activities, and statistical results,” *Proc. of ICSE14*, pp.320-326 (1992).

[20] I. Jacobson, M. Griss and P. Jonsson: Software Reuse, *Addison Wesley*, (1997).

[21] “JAMA : A Java Matrix Package,” http://math.nist.gov/javanumerics/jama/

[22] B. Keepence and M. Mannion: “Using patterns to model variability in product families,” *IEEE Software*, Vol. 16, No. 4, pp. 102-108 (1999).

[23] 長橋: “類似度に基づくソフトウェア品質の評価,” 情報処理学会研究報告 2000-SE-126, vol. 2000, no. 25, pp. 65-72, (2000).

[24] 古賀, 飯田, 松本, 井上: “ソフトウェアコンポーネント利用情報の収集と共有”, 電子情報通信学会技術報告, vol.100, No.472, pp. 1-8, (2000).

[25] F. Narin, G. Pinski, and H. H. Gee: “Structure of the Biomedical Literature,” *Journal of the American Society for Information Science*, Vol. 27, No. 1, 25-45, (1976).

[26] 西本: “Java 言語について,” http://cappuccino.ne.jp/keisuken/java/

[27] L. Page, S. Brin, R. Motwani, T. Winograd: “The PageRank Citation Ranking: Bringing Order to the Web,” http://www-db.stanford.edu/ backrub/pageranksub.ps

[28] G. Pinski and F. Narin: “Citation Influence for Journal Aggregates of Scientific Publications: Theory, with Application to the Literature of Physics,” *Information Processing and Management*, Vol. 12, No. 5, pp. 297-312, (1976).

[29] G. Pinski and F. Narin: “Structure of the Psychological Literature,” *Journal of the American Society for Information Science*, Vol. 30, No. 3, pp. 161-168, (1979).

[30] G. Spanoudakis, P. Constantopoulos: “Measuring Similarity Between Software Artifacts”, *Proceedings of 6th International Conference on Software Engineering and Knowledge Engineering (SEKE ’94)*, pp. 387-394, (1994).

[31] “The Source For Java(TM) Technology,” Sun Microsystems, http://java.sun.com/

[32] “JavaBeans(TM),” Sun Microsystems, http://java.sun.com/products/javabeans/

[33] 鷲崎, 深澤: “有向置換性距離に基づくコンポーネント検索方式の実現と評価”, オブジェクト指向最前線 2001, pp. 77- 84, (2001)

[34] 山本, 松下, 神谷, 井上: “ソフトウェアシステムの類似度とその計測ツール SMMT,” 電子情報通信学会論文誌 D-I(採録決定).

[35] 山本, 鷲崎, 深澤: “再利用特性に基づくコンポーネントメトリクスの提案と検証,” ソフトウェア工学の基礎ワークショップ (FOSE2001), (2001).

[36] 山本, 鷲崎, 深澤: “静的側面から見たソフトウェアコンポーネントの品質”, 電子情報通信学会技術研究報告, vol. 101, No. 98, pp. 33-40, (2001).