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
SS2006-12 (2006-04)

進行中のプロジェクトに有用なソフトウェアコンポーネントの推薦方法

亀井 靖高† 角田 雅照† 柿元 健† 大杉 直樹† 門田 暁人† 松本 健一†

†奈良先端科学技術大学院大学情報科学研究科 〒630-0192 けいはんな学研都市

E-mail: †{yasuta-k, masate-t, takesi-k, naoki-o, akito-m, matumoto}@is.naist.jp

あらまし 品質の高いソフトウェアを効率よく開発するため，開発プラットフォームベンダから，数多くのソフトウェアコンポーネントが提供されている．しかし，提供されるコンポーネントの数が膨大であるため，開発者が有用なコンポーネントを見つけにくいという問題が生じている．本稿では，この問題を解決すため，進行中のプロジェクトに有用なコンポーネントを，協調フィルタリングを用いて推薦する方法を提案する．提案方法では，まず，進行中のプロジェクトとコンポーネントの利用状況が類似する過去のプロジェクトを見つける．次に，類似プロジェクトで利用されており，進行中のプロジェクトで利用されていないコンポーネントを開発者に推薦する．SourceForge.net に登録されている オープンソースソフトウェア開発プロジェクト 33 件分のデータを用いて，推薦の正確さを評価する実験を行った．Half-Life Utility を評価基準として用いた場合，単純な方法（よく使われるコンポーネントから優先的に推薦する）より提案方法の方が，約 10.95%程度正確に推薦を行えることを確認した．

キーワード 情報フィルタリング，プロジェクト，類似度，類似度計算，リファクタリング

A Recommendation Method of Useful Software Components for Ongoing Project

Yasutaka KAMEI† Masateru TSUNODA† Takeshi KAKIMOTO†
Naoki OHSUGI† Akito MONDEN† Ken-ichi MATSUMOTO†

† Graduate School of Information Science, Nara Institute of Science and Technology
Kansai Science City, 630-0192 Japan

E-mail: †{yasuta-k, masate-t, takesi-k, naoki-o, akito-m, matumoto}@is.naist.jp

Abstract Many software components have been provided by development platform vendors, for achieving efficient development of high quality software; however, some practitioners cannot find useful components because number of the provided components is too large. For solving this problem, we propose a method based on collaborative filtering for recommending useful components to each ongoing project. In the proposed method, at first, some past projects similar to given ongoing project are retrieved by calculating similarity with number of common components used in the ongoing project and each past project. Then, practitioners of the ongoing projects can get some currently unused components pastly used in the similar projects. We evaluated recommendation accuracy of the proposed method with data containing 33 open source software development projects on the SourceForge.net. When we employed Half-Life Utility as a evaluation criterion, the proposed method could make better recommendation than a simple recommendation. The average difference between them was 10.95%.

Keyword information filtering, projects, similarity, similarity computation, refactoring

1. まえがき

近年，開発プラットフォームベンダから，数多くのソフトウェアコンポーネント（以降，単にコンポーネントと記す）が提供されている．提供されているコンポーネントを利用することで，品質の高いソフトウェアを効率よく開発できる[12].

しかし，提供されるコンポーネントの数が増えすぎ，有用なコンポーネントを見つけにくいという問題が生じている．特に，まだ経験が浅い技術者はごく一部のコンポーネントしか利用していないと考えられる．例えば，Java 2 SDK, Standard Edtion (J2SE)のバージョン 1.4.1_02 の基本クラス群である rt.jar には，5568 個のクラスやインタフェースが含まれている．代表的なオープンソースソフトウェアのデータを基に，rt.jar で提供されているクラス中で実際に用いられたものを調べると，約 4%（約 224 クラス）程度しか用いられていない（図 1）.

有用なコンポーネントを発見する手段として，

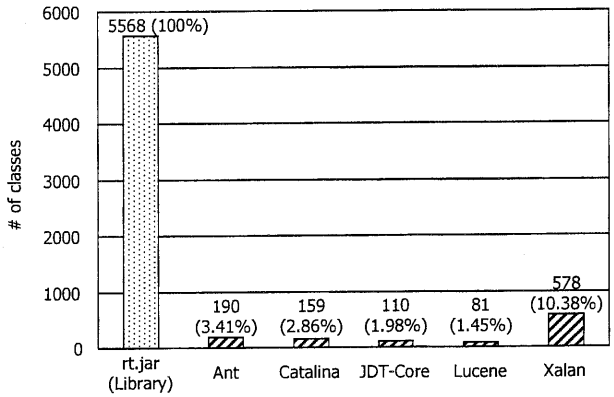

図 1. プロジェクトにおけるクラスの利用数

Koders[8]や，SPARS-J[4]などのコンポーネントの検索エンジンが存在する．コンポーネントの検索エンジンでは，Web の検索エンジンと同様に，検索にはキーワードを入力する必要がある．しかし，開発者自身が知らないコンポーネントを発見するための検索キーワードを開発者が類推することは非常に難しい．また，検索結果の中から有用なコンポーネントを発見することも困難である．そのため，開発者にとって有用なコンポーネントの発見を支援する仕組みが必要である．

本稿では，有用な Java のクラスを推薦する方法を提案する．提案方法では，検索キーワードを入力するのではなく，プロジェクトで利用されているコンポーネントの使用回数を用いて，プロジェクトで未使用の有用だと思われるコンポーネントを推薦する．推薦には，協調フィルタリングを用いる．協調フィルタリングとは，ユーザの嗜好を推測し，好みにあったアイテムを推薦するための手法である．

以降，2 章では提案するコンポーネント推薦方法の詳細について，3 章では評価実験について，4 章では，実験結果について考察する．5 章で関連研究について紹介し，6 章でまとめと今後の課題について述べる．

2. コンポーネント推薦方法

2.1. 推薦方法の概要

本稿で提案するソフトウェアコンポーネント推薦方法は，進行中のプロジェクトに対して，プロジェクトで未使用のコンポーネンを推薦する．提案方法は，進行中のプロジェクトの実行ファイルを入力とするため，プロトタイププログラミングやアジャイル開発など，改良を繰り返しながら開発を進めていく場合や，プロジェクトの更改し，次期バージョンを開発するなどにおける利用を想定している．また，本稿では，プログラミング言語として Java を対象とし，プロジェクトの実行ファイルは jar ファイル，推薦するコンポーネントは Java クラスファイルとする．

先行研究として，我々は，単独の Java クラスファイルに対する Java クラスの推薦を提案している[14]．従来の方法では，単独の Java クラスファイルを入力として，入力されたクラスファイルで未使用のクラスを推薦するため，開発者が他で使用している既知のクラスが推薦されるという問題があった．一方，提案手法では，進行中のプロジェクトの jar ファイルを入力とすることで，プロジェクトにとって未知の Java クラスファイルのみを推薦する．そのため，単独のファイルを入力とするよりも，開発者が既知のコンポーネントが推薦されることが軽減されると考えられる．また，従来の方法では，クラスを使用/未使用を用いて推薦したが，提案方法では，クラスの使用回数を用いて推薦する．

提案方法は以下の手順で推薦を行う．

1. ユーザから進行中のプロジェクトの jar ファイルが入力される．
2. 入力された jar ファイルを解析し，クラスの使用回数を調べる．クラスの使用回数の解析には，J-birth[13]の Used Class Engine を用いた．
3. クラスの使用回数から協調フィルタリングを用いて，入力されたプロジェクトで未使用の Java クラスファイルを推薦する．
4. 推薦結果をユーザに提示する．

2.2. 協調フィルタリング

協調フィルタリングは大量のアイテムの中から，ユーザの好みに合うアイテムを選び出し，推薦するシステムの実装に用いられる．一般に，協調フィルタリングを用いた推薦は次の手順で行われる．

手順1. 各ユーザは使用したアイテム(書籍・楽曲・映画など)に対する評価を行う．
手順2. 推薦対象のユーザ（対象ユーザ）と評価の傾向が似ているユーザ（類似ユーザ）を探す．
手順3. 対象ユーザが未使用のアイテムに対する評価を，類似ユーザの評価に基づいて予測する．
手順4. 対象ユーザが高く評価すると予測されたアイテムを対象ユーザに推薦する．

提案方法では，ユーザはプロジェクトに，アイテムは Java のクラス，評価はクラスの使用回数となる．

2.3. 協調フィルタリングによる推薦

協調フィルタリングよる推薦は，類似度計算，予測値算出の手順で行われる．推薦手順の説明に用いる $m \times n$ 行列を図 2に示す．図 2の $m \times n$ 行列の $p_i \in \{p_1, p_2, \ldots, p_n\}$は i 番目のプロジェクトを示し，$l_j \in \{l_1, l_2, \ldots, l_n\}$は j 番目のコンポーネントを示す．$u_{i,j} \in \{u_{1,1}, u_{1,2}, \ldots,$

	l_1	l_2	…	l_j	…	l_b	…	l_n
p_1	$u_{1,1}$	$U_{1,2}$	…	$u_{1,j}$	…	$u_{1,b}$	…	$u_{1,n}$
p_2	$u_{2,1}$	$U_{2,2}$	…	$u_{2,j}$	…	$u_{2,b}$	…	$u_{2,n}$
…	…	…		…		…		…
p_i	$u_{i,1}$	$u_{i,2}$	…	$u_{i,j}$	…	$u_{i,b}$	…	$u_{i,n}$
…	…	…		…		…		…
p_a	$u_{a,1}$	$u_{a,2}$	…	$u_{a,j}$	…	$u_{a,b}$	…	$u_{a,n}$
…	…	…		…		…		…
p_m	$u_{m,1}$	$u_{m,2}$	…	$u_{m,j}$	…	$u_{m,b}$	…	$u_{m,n}$

図 2. 推薦に用いられる $m \times n$ 行列

$u_{m,n}\}$はプロジェクト p_i におけるコンポーネント l_j の使用回数を示す．

類似度計算には Adjusted Cosine Similarity 法を用いた．類似度は推薦対象である進行中のプロジェクトと似ているプロジェクトを選択するために算出する．多くの類似度計算アルゴリズムが提案されている[1][2][9][10]が，本稿では，事前実験において最も高い精度が得られた Adjusted Cosine Similarity 法を用いた．

Adjusted Cosine Similarity 法は，情報検索の分野における２つの文章間の類似度を評価するために提案されたアルゴリズムである Cosine Similarity 法の１つである．Cosine Similarity 法における類似度は，文章を単語頻度のベクトルとして扱い，２つの単語頻度ベクトルのなす角度の余弦として算出することによって求められる[11]．Cosine Similarity 法では，推薦対象プロジェクト p_a とそれ以外のプロジェクト p_i 間の類似度 $sim(p_a, p_i)$は，式(1)として定義されている．式(1)では，プロジェクト，コンポーネント，コンポーネントの使用回数は，文章，単語，単語の頻度として置き換えられている．類似度 $sim(p_a, p_i)$は，０から１の値を取る．

$$sim(p_a, p_i) = \frac{\sum u_{a,j} \times u_{i,j}}{\sqrt{\sum (u_{a,j})^2} \sqrt{\sum (u_{i,j})^2}} \qquad (1)$$

Adjusted Cosine Similarity 法では，$u_{a,j}$ から $avg(u_j)$を減算し，$u_{a,j}$ の重心(平均値)を０に移した点が異なる．これにより，コンポーネントの使用数が平均よりも大きい場合は正の値をとり，小さい場合は負の値をとるため，大きくコンポーネントの使用数が離れたプロジェクトは類似度が低くなる．

$$sim(p_a, p_i) = \frac{\sum (u_{a,j} - avg(u_j))(u_{i,j} - avg(u_j))}{\sqrt{\sum (u_{a,j} - avg(u_j))} \sqrt{\sum (u_{i,j} - avg(u_j))}} \qquad (2)$$

予測値算出には，Weighted Sum 法[10]を用いた．推薦対象プロジェクトで未使用のコンポーネントに対して，予測値を算出する．類似度計算のアルゴリズムと同様に，多くの推薦スコアのアルゴリズムが提案されている[1][2][9][10]が，本稿では，事前実験において最も高い精度が得られた Weighted Sum 法を用いた．推薦スコアを求めるために，すべてのプロジェクトを用いず，k 個の類似するプロジェクトを用いる．

推薦対象プロジェクト p_a の b 番目のコンポーネントの使用回数 $u_{a,b}$ の予測値 $R_{a,b}$ は式(2) として定義されている．予測値は，類似プロジェクト p_i の b 番目のコンポーネントの使用回数 $u_{i,b}$ の加重平均で算出される．重みは，推薦対象のプロジェクト p_a とそれぞれのプロジェクト p_i の類似度である．*k-nearestProjects* は，k 個の類似プロジェクトの集合である．

$$R_{a,b} = \frac{\sum_{K \in k-nearestProjects} sim(p_a, p_K) \times u_{K,b}}{\sum_{K \in k-nearestProjects} sim(p_a, p_K)} \qquad (3)$$

式(6)の k は事前実験において最も高い精度が得られた k=3 とした．

3. 評価実験

3.1. 実験の概要

提案方法を評価するために，単一バージョンを用いた実験と複数バージョンを用いた実験の二種類の方法で実験を行った．

単一バージョンを用いた実験では，推薦されるコンポーネントが進行中のプロジェクトにおいて有用であるかを評価する．プロジェクトで実際に使用されているコンポーネントの１つを未使用と仮定し，そのコンポーネントの使用回数を予測し，実際の使用回数と比較する．

複数バージョンを用いた実験では，推薦するコンポーネントが次期バージョンを開発するために有用であるかを評価する．プロジェクトの旧バージョンを用いて，旧プロジェクトで使用されているコンポーネントの１つを未使用と仮定し，そのコンポーネントの使用回数を予測し，また，旧バージョンで未使用のコンポーネントで，最新バージョンで使用しているコンポーネントについても使用回数を予測し，それぞれ最新バージョンでの使用回数と比較する．

3.2. 実験に使用したデータ

実験に使用したデータは，オープンソースソフトウェア開発環境を提供している SourceForge.net において開発されているプロジェクトのうち，Java 言語を用い

て開発されているプロジェクトである．

単一バージョンを用いた実験では，SourceForge.netにおいてプロジェクトの活発さを表す指標であるActivityの上位33件のプロジェクトを用いた．また，33件のプロジェクトで用いられているコンポーネントのうち，利用頻度が著しく高く多くの開発者にとって既知であると考えられるコンポーネントを除く3107個のコンポーネントを用いた．

複数バージョンを用いた実験では、単一プロジェクトを用いた実験で用いた33件のプロジェクトのうち，過去のバージョンが公開されているプロジェクト29件を用いた．また，29件のプロジェクトの過去，あるいは最新バージョンで用いられているコンポーネントのうち，利用頻度が著しく高いコンポーネントを除く2558個のコンポーネントを用いた．

3.3. 実験の手順

単一バージョンを用いた実験，および，複数バージョンを用いた実験の手順を図 2の行列を用いて説明する．

単一バージョンを用いた実験では，図 2の $p_i \in \{p_1, p_2, ..., p_n\}$は，33件のプロジェクトを，$l_j \in \{l_1, l_2, ..., l_n\}$は，プロジェクトで使用されている3107個のコンポーネントを示す．単一プロジェクトを用いた実験手順を以下に示す．

1. 推薦対象のプロジェクトとして，i番目のプロジェクトp_iを選択する．
2. プロジェクトp_iのj番目のコンポーネントl_jの使用回数$u_{i,j}$を未使用とし，l_jに対する推薦スコア$R_{i,j}$を他のプロジェクトを用いて協調フィルタリングで算出する．
3. 手順2をプロジェクトp_iで使用されている全てのコンポーネントで実行する．
4. 手順1，2，3を全てのプロジェクトにおいて実行する．

複数バージョンを用いた実験では，図 2の $p_i \in \{p_1, p_2, ..., p_n\}$は，29件のプロジェクトの最新バージョンを，$l_j \in \{l_1, l_2, ..., l_n\}$は，旧バージョン，あるいは最新バージョンで使用されている2558個のコンポーネントを示す．複数バージョンを用いた実験の手順を以下に示す．

1. 推薦対象のプロジェクトとして，i番目のプロジェクトp_iを選択し，p_iを旧バージョンに置き換える．
2. 旧バージョンのプロジェクトp_iのj番目のコンポーネントl_jの使用回数$u_{i,j}$を未使用と仮定し，l_jに対する推薦スコア$R_{i,j}$を他のプロジェクトの最新バージョンを用いて協調フィルタリングで算出する．
3. 手順2をプロジェクトp_iの旧バージョン，あるいは最新バージョンで使用されている全てのコンポーネントで実行する．
4. 手順1，2，3を全てのプロジェクトにおいて実行する．

また，提案方法の比較対照とするために，平均値に基づく手法でも予測値を算出した．平均値に基づく手法は，推薦対象プロジェクトp_aのb番目のコンポーネントの使用回数I_bの予測値$R_{a,b}$を式(4)で算出する手法である．式(4)のN_cは全プロジェクト数を，N_lはコンポーネントl_bを使用しているプロジェクト数である．

$$R_{a,b} = \frac{N_l}{N_c} \tag{4}$$

3.4. 評価基準

評価基準として，推薦システムを評価する指標の1つである Half-Life Utility[3]と絶対誤差平均[7]を用いた．

Half-Life Utility は，式(5)，(6)で定義され，推薦順序を評価する指標である．式(5)，(6)の，$s_{a,d}$はプロジェクトp_aに対してd番目に推薦されたコンポーネントの実際の使用回数である．αは，ユーザが推薦結果を確認する確率が50%となる順位である．本稿では，αを10とした．H_a^{max}は，システムが完全に推薦できた場合のH_aの値である．Half-Life Utility は，値が大きいほど正しく推薦できていることを表す．

$$H_a = \sum_d \frac{s_{a,d}}{2^{(d-1)/(\alpha-1)}} \tag{5}$$

$$H = 100 \times \frac{\sum_a H_a}{\sum_a H_a^{\max}} \tag{6}$$

プロジェクトp_aの絶対誤差平均($|r_a|$)は式(7)で定義される．式(7)のz_iは実際のコンポーネントの使用数を，y_iは予測されたコンポーネントの使用数を，N_aはプロジェクトp_aの全コンポーネント数である．絶対誤差平均は，値が小さいほど推薦精度が高いことを表す．

$$|r_a| = \frac{\sum_{i=1}^{N_i} |z_i - y_i|}{N_i} \tag{7}$$

3.5. 実験結果

単一バージョンを用いた実験，および複数バージョ

表1 実験結果

		Half-Life Utility	絶対誤差平均
単一バージョンを用いた実験	平均値を用いた手法	60%	7.242
	提案方法	71.5%	7.535
		Half-Life Utility	絶対誤差平均
複数バージョンを用いた実験	平均値を用いた手法	16.1%	3.888
	提案方法	26.5%	3.107

ンを用いた実験における，提案方法と平均値を用いた手法の結果を表 1に示す．単一バージョンを用いた実験，および複数バージョンを用いた実験ともに，提案方法の方が平均値を用いた手法よりもHalf-life Utilityの値が大きくなり，正しく推薦が行えていると言える．絶対誤差平均では，単一バージョンを用いた実験では，提案方法よりも，平均値を用いた手法の方が高い精度が得られたが，複数バージョンを用いた実験では，提案手法の方が平均値を用いた手法よりも高い精度が得られた．

また，Half-life Utility，絶対誤差平均ともに，単一バージョンを用いた実験と比較すると，複数バージョンを用いた実験では精度が低下した．

複数バージョンを用いた実験は，単一バージョンを用いた実験と比較すると，平均値を用いた手法，提案方法ともに精度が低下した．

4. 考察

提案方法は，Half-Life Utilityにおいて平均値を用いた手法よりも高い結果が得られていることから，コンポーネントを推薦する順序の精度が高い，特にHalf-Life Utilityの性質上，上位のコンポーネントを推薦する順序の精度が高いと考えられる．一方，提案手法は，単一バージョンを用いた実験の絶対誤差平均において平均値を用いた手法よりも精度が低い結果が得られている．そのため，提案方法は，推薦順序が下位のコンポーネントでは精度が低いと考えられる．一般的に，推薦では上位のコンポーネントを推薦する順序の精度がより重要であるので，提案方法は有効であると考えられる．

また，複数バージョンを用いた実験は，単一バージョンを用いた実験と比較すると，提案方法，平均値を用いた手法ともに精度は低下した．このことは，複数バージョンを用いた実験で推薦対象とする旧バージョンは，最新バージョンと比較するとコンポーネント数が少ないため，他のプロジェクトの最新バージョンと共通して使用されているコンポーネントが少なく，類似度算出において差が小さくなってしまっていることが考えられる．そこで，アイテム数が非常に多く，ユーザによる評価が全体の 1%以下しか評価が行われていないデータセットに対して適用されている協調フィルタリングのアルゴリズムであるアイテムベース手法[10]が有効である可能性が高い．

5. 関連研究

有用なコンポーネントを発見する手段として，コンポーネント検索エンジンが多数提案されている．

ソースコードを検索対象とした検索エンジンとして，SPARS-J[4]や Koders[6]がある．SPARS-J は Javaソースコード検索システムであり，ソースコードを性的解析して抽出した索引語を用いてキーワード検索を行う．Kodersは，C や COBOL などの多言語に対応したソースコード検索エンジンであり，指定したキーワードを含むオープンソースソフトウェアのソースコードを検索できる．

オブジェクト指向特有の関係を用いた検索エンジンとして，Jarhoo[5]や Prospector[8]がある．Jarhooは，Java のクラスとそのクラスが含まれる jar ファイルの関係を調べるための検索エンジンであり，クラスやインタフェース名を入力すると，属する jar ファイルが検索結果として出力される．また，jar ファイル名を入力すると，その jar ファイルが用いられているアプリケーションが出力される．Prospector は，目的のオブジェクトを生成するために必要なライブラリの検索エンジンであり，利用可能なオブジェクトと目的のオブジェクトを入力とし，どのようなライブラリが必要かを出力とする．

検索エンジンにおける出力は，ユーザから入力されるキーワードに大きく影響される．そのため，入力キーワードの類推が容易な熟練者にとって役立つかもしれないが，初心者にとっては，敷居が高く，有用な検索結果を得ることは困難である．また，熟練者にとっても自分の知識外の結果を得ることは困難である．一方，提案方法は，進行中のプロジェクトのファイルを入力として，有用なコンポーネントを出力として推薦するため，初心者にとっても容易に利用できると考え

られる．

6. むすび

本稿では，進行中のプロジェクトのファイルを入力として，協調フィルタリングを用いてコンポーネントを推薦する方法を提案し，オープンソースソフトウェアを用いた実験により評価を行った．実験の結果，平均値を用いた手法に比べ，より有効な推薦を行えることがわかった．

今後の課題として，次期バージョンにおいて有用なコンポーネントの推薦の精度を向上させることが挙げられる．また，さらに多くのプロジェクトのデータを用いて実験することで，結果の信頼性を向上させることが挙げられる．

7. 謝辞

本研究の一部は、文部科学省「eSociety 基盤ソフトウェアの総合開発」の委託に基づいて行われた．

文　献

[1] J. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," in *Proc. Conf. on Uncertainty in Artificial Intelligence*, Madison, WI, pp. 43-52, Jul. 1998.

[2] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol.35, no.12 pp. 61-70, Dec. 1992.

[3] J. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl, "Evaluating collaborative filtering recommender systems," ACM Transactions on Information Systems (TOIS), Vol.22 , No.1, pp.5-53, 2004.

[4] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, S. Kusumoto, "Component Rank: Relative Significance Rank for Software Component Search," In Proc. of the 25th International Conference on Software Engineering (ICSE2003), pp14-24, Portland, Oregon, U.S.A., May, 2003.

[5] Jarhoo, http://www.jarhoo.com/

[6] Koders, http://www.koders.com/

[7] E. Mendes, I. Watson, C. Triggs, N. Mosley, and S. Counsell, "A Comparative Study of Cost Estimation Models for Web Hypermedia Applications," Empirical Software Engineering, Vol.8, Issue 2, pp.163-196, 2003.

[8] Prospector, http://snobol.cs.berkeley.edu/prospector/index.jsp

[9] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," In *Proc. ACM Conf. on Computer Supported Cooperative Work*, Chapel Hill, NC, pp.175-pp.186, Oct. 1994.

[10] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms," In *Proc. International World Wide Web Conference*, Hong Kong, China, pp. 285-295, May, 2001.

[11] G. Salton, and M. McGill, *Introduction to Modern Information Retrieval*. New York: McGraw-Hill, 1983.

[12] C.Szyperski, "Component Software," Addison-Wesley, New York, 1998.

[13] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, "Java birthmark –Detecting the software theft," *IEICE Transactions on Information and Systems, E88-D*, 9, pp.2148-2158 Sept. 2005

[14] M. Tsunoda, T. Kakimoto, N. Ohsugi, A. Monden, and K. Matsumoto, "Javawock: A Java Class Recommender System Based on Collaborative Filtering," In Proc. 17th International Conference on Software Engineering and Knowledge Engineering (SEKE2005), pp.491-497, Taipei, Taiwan, Jul. 2005.