情報処理学会論文誌 Vol. 50 No. 12 3095–3107 (Dec. 2009)

開発中のソースコードに基づくソフトウェア部品の自動推薦システム A-SCORE

島田 隆次[†1] 市井 誠[†1] 早瀬 康裕[†1]
松下 誠[†1] 井上 克郎[†1]

ソフトウェアの再利用は，クラスやメソッドなど（ソフトウェア部品）の単位で行われることが多い．このようなソフトウェア部品の再利用を促進するために，コーディング中に自動的にソフトウェア部品を推薦する手法が提案された．しかし，既存手法では変更を加えずに再利用できる部品しか発見できないため，再利用の機会が限られるという問題点がある．そこで本稿では，変更を加えなければ再利用できないソフトウェア部品をも検索できる自動推薦手法を提案する．提案手法ではソースコード中のコメントや識別子の中に類似した単語を含む部品を検索することで，多少の変更を加えれば再利用できるようなソフトウェア部品も推薦することができる．また，提案手法を実装したソフトウェア部品自動推薦システム A-SCORE を Eclipse プラグインとして作成し，実験によって再利用の促進などの効果を評価した．その結果，A-SCORE を利用したほうが再利用したソフトウェア部品の個数が多くなり，また，既存手法では自動推薦が行えなかったような場合においても自動推薦による再利用が行えた事例があったことから，A-SCORE の有効性が示された．

A-SCORE: Software Component Recommendation System Based on Source Code under Development

Ryuji Shimada,[†1] Makoto Ichii,[†1] Yasuhiro Hayase,[†1]
Makoto Matsushita[†1] and Katsuro Inoue[†1]

An automatic software component recommendation approach was proposed to support reusing software components such as classes or modules. However the approach recommends only the components that can be reused without modifications. Consequently, developers using the system miss many reusing opportunities yet. Therefore, this paper proposes a new automatic component recommendation approach that supports various reuse scenarios, e.g., modifying a component before importing or copy-and-pasting in part. Our approach recommends components similar to those a developer is editing on the basis of words in comments and identifiers. The approach is implemented as A-SCORE, an extension of Eclipse IDE. An experimentation is performed for evaluating the extension from the view of reuse efficiency. The subjects of the experiment implemented same tasks with or without the extension, then the number of reused components and defects are measured. The result shows that A-SCORE recommends components in the situations in which the existing systems never recommend any components and increases reused components.

1. はじめに

ソフトウェア開発においては，適切な再利用を行うことで，ソフトウェアの品質や生産性が向上するといわれている[6]．ソフトウェアの再利用は，ソフトウェア部品（以下，単に部品）と呼ばれる，ソフトウェアの構成要素であるクラスやメソッドなどの単位で行われることが多い．部品は，企業で開発したソフトウェアやインターネット上で入手できるオープンソースソフトウェアなどに含まれ，世の中に無数に存在する．

しかし，部品の再利用には，それを妨げるいくつかの要因がある．たとえば，部品には，ドキュメントが整備されていないために利用方法や動作条件などが分かりにくく，そのままでは再利用が難しいものが多く存在する．そのような部品を再利用するためには，開発者が部品のソースコードを読んで再利用できるかどうかを判断しなければならないが，大量の部品の中から開発者が手作業で必要な部品を見つけることは手間がかかり困難である．

そこで，大量の部品から目的にあった部品を効率的に見つけるために，キーワード検索による部品検索システムが用いられている．部品検索システムとは，部品を機能や用途などで分類・管理して蓄積し，ユーザの指定に応じて部品を探し出すシステムである．多くの部品検索システムは，ユーザによって指定されたキーワード（検索語句）に適合する部品を検索する，キーワード検索機能を提供する．

しかし，キーワード検索を用いた再利用には 2 つの問題点がある．第 1 に，開発者が意識しないと検索が行われないという問題である．第 2 に，開発者が適切なキーワードを選定する必要があるという問題である．

これらの問題に対して，Ye らは，開発者が明示的に検索を行わなくてもシステムが自動的に部品を検索する，部品自動検索という概念を提案し，その手法を CodeBroker という

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

ツールとして実装した[9]．部品自動検索では，開発者によるソースコードの編集を監視することで，開発者の手間を増やすことなく，自動的に再利用可能な部品を検索することができる．

しかし，CodeBroker には，検索を行うタイミングが限られているという問題と，変更を加えずに再利用を行える部品しか検索できないという問題がある．CodeBroker はドキュメントコメントの内容に基づいて検索を行うため，ドキュメントコメントを書いたときにしか部品を推薦できない．また，部品の再利用では，再利用先のプログラムに合わせて部品に変更を加えてから利用する場合も多いが，CodeBroker は変更が必要となる部品を発見することができない．

そこで本稿では，プログラム記述中の多くのタイミングで，変更が必要となるような部品をも自動的に推薦できる，新しい自動推薦手法を提案する．提案手法では，ソースコード中のコメントや識別子に基づいて検索を行う．また，検索の際には，自然言語に対する検索手法である潜在的意味インデキシング LSI（Latent Semantic Indexing）[3] を応用して，曖昧さを許容して検索を行う．これにより，ある程度の差異がある部品も検索できるようになるため，変更を加えてから再利用する場合にも対応できるようになる．また，提案手法は，ドキュメントコメント以外にも様々な情報を利用して検索を行うため，メソッドのボディやクラス宣言など，ソースコードの様々な部分を記述しているときにも推薦を行うことができる．

また，提案手法を実装した部品自動推薦システム A-SCORE を作成し，学生を用いた実験によって A-SCORE による再利用の促進などの効果を評価した．実験では被験者にプログラミング課題を与え，再利用した既存部品の個数や完成したプログラムの不具合の数などを測定した．その結果，A-SCORE を利用したほうが再利用した部品の個数が多く，完成したプログラムの不具合も少なくなることが分かり，プログラムの品質が高くなることが明らかになった．また，既存手法では自動推薦が行えない場合において，自動推薦による再利用が行えた事例も存在したことが分かり，このことからも A-SCORE の有用性が示された．

以下，2 章では本研究の背景となる既存の部品検索システムについて述べる．3 章では提案手法の詳細について述べ，4 章で提案手法を実装した部品自動推薦システム A-SCORE について説明する．5 章では A-SCORE の評価実験とその妥当性について述べる．6 章ではまとめと今後の課題について述べる．

2. 部品検索システム

1 章で述べたように，部品検索システムとは，部品を機能や用途などで分類・管理して蓄積し，ユーザの指定に応じて部品を探し出すシステムである．現在までに開発された部品検索システムの例としては，Koders[⋆1] や Google Code Search[⋆2]，SPARS-J[5),⋆3]，Sourcerer[1),⋆4] などが存在し，そのほとんどがキーワード検索機能を提供している[4]．部品検索システムのキーワード検索機能を用いた再利用は，以下のような手順となる．

(1) 開発者がソフトウェアを開発しているときに問題に遭遇し，その問題を解決するために利用できる部品を探そうと思い立つ．
(2) 開発者が検索のためのキーワードを考え，部品検索システムに入力する．
(3) 部品検索システムがキーワードに合致する部品を検索し，開発者に提示する．
(4) 開発者がその部品が問題の解決に利用できるかを判断する．
(5) 利用可能な部品が見つかった場合，開発者はその部品を開発中のソフトウェアに取り入れる．

しかし，キーワード検索を用いた再利用には大きな問題点が 2 つある．1 つ目は，開発者が意識しないと検索が行われないという問題である．上記手順の (1) で，開発者が部品を探そうと思い立たない限り，利用可能な部品は検索されない．2 つ目は，開発者がよく知らない分野に関する部品は検索しにくいという問題である．目的の部品を得るためには，上記手順の (2) で目的とする機能を表す適切なキーワードを選定する必要がある．しかし，問題となっている分野について開発者が詳しい知識を持っていない場合，解決方法を表す標準的な語句が分からず，適切な検索を行うことができない．

2 つ目の問題に対する解決策としては，キーワードによらない検索手法や，キーワード検索システムの検索結果を改善する手法などが提案されている．たとえば，鷲崎らは，部品のプロトタイプを入力として，置き換え可能な既存部品を，置き換えにともなって修正しなければならない箇所の数に基づいて検索するシステム RetrievalJ を実装した[10]．また，我々の研究グループで開発された部品検索システム SPARS-J では，部品の重要度を表す値 ComponentRank を利用することで，キーワード検索による検索結果を改善することに成

⋆1 http://www.koders.com/
⋆2 http://www.google.com/codesearch
⋆3 http://demo.spars.info/
⋆4 http://sourcerer.ics.uci.edu/sourcerer/search/

功した[5]．しかし，これらの手法では，1 つ目の問題は解決できない．

一方，1 つ目の問題に対して，Ye らは開発者の指示なしにシステムが自動的に検索を行う，部品自動検索という概念を提案した[9]．部品自動検索では，開発者によるソースコードの編集中に，システムが自動的に必要な情報を収集して検索を行い，利用可能な部品を開発者に提示する．部品自動検索の利点としては，次の 2 つがあげられる．

(1)　検索のタイミングをシステムが自動的に決定するため，再利用可能な部品があった場合には開発者が意識していなくても部品が提示される．

(2)　検索条件をシステムが自動的に決定するため，開発者がうまくキーワードを書けなくても検索が行える．

さらに Ye らは，部品自動検索を実現するための手法を提案し，それを実装したシステム CodeBroker を作成した．CodeBroker は Java のメソッドを部品として扱い，開発者がメソッドの宣言を書いたときに自動的に検索を行う．ここでいうメソッドの宣言とは，メソッドに付随するドキュメントコメント（機能などを説明するコメント）とシグネチャ（引数と戻り値の型）のことを表す．開発者がメソッドの宣言を書くと，CodeBroker はそれに含まれるドキュメントコメントをもとに自然言語に対する解析手法である LSA（Latent Semantic Analysis）[7] を用いて類似メソッドを検索する．その後，検索結果からシグネチャが一致するメソッドだけを抽出し，再利用可能な部品として開発者に提示する．

しかし，CodeBroker には，変更なしで再利用できる部品しか検索できないという問題点がある．なぜならば，CodeBroker は再利用にあたって修正しなければならない箇所が少ない部品を提示するために，シグネチャ（入出力となる引数や戻り値）が一致する部品のみを検索結果として表示するためである．しかし，実際の再利用では再利用先のプログラムに合わせて部品に変更を加えてから利用を行う場合が多々ある[8]．

また，再利用の方法には，コピー&ペーストによって小さなコード片だけを再利用する（コード片の再利用）方法もあるが，CodeBroker はコード片の再利用にも対応できない．なぜならば，コード片の再利用には，その中で利用しているメソッドなどの情報が必要となるが，CodeBroker の検索ではそれらの情報は用いられないためである．また，CodeBroker はメソッドの宣言を書いたときにのみ検索を行うが，宣言を書いた時点ではメソッドの本体が存在しないため，コード片の検索に必要な検索条件を生成することもできない．

3. 部品自動推薦手法

本章では，開発者が Java プログラムを記述しているときに，自動的に部品を推薦する手法を提案する．再利用はクラス単位で行われる場合が多いことに鑑み，本手法はクラスを部品として扱う．本手法では，まず，部品から特徴と呼ばれる情報を抽出し，検索に必要な索引を作成する．部品の特徴とは，コメントや識別子を構成する単語である．そして，開発者によるソースコードの編集に応じて，自動的に索引を用いて曖昧さを許容した検索を行う．開発者は提案手法を用いることで，部品全体をそのまま利用する場合だけではなく，部品に変更を加えて再利用する場合や，コード片を再利用する場合に適した部品も入手することができる．

次に，本手法の処理手順を説明する．本手法は検索に用いる索引を作成する処理（**索引作成処理**）と，開発者の編集に応じて自動的に検索を行う処理（**推薦処理**）からなる．処理手順の概要を以下に記す．

索引作成処理（図 1）

(1)　部品データベースに格納されているソースコードから特徴を抽出する

(2)　得られた特徴から索引を作成する

推薦処理（図 2）

(1)　開発者によるソースコードの編集を監視し，検索のタイミングを決定する

(2)　編集中のソースコードから特徴を抽出し，検索クエリを生成する

(3)　索引を用いて検索クエリに合う類似部品を検索する

(4)　検索された部品を開発者に推薦する

なお，検索にはベクトル空間モデルの応用である潜在的意味インデキシング LSI（Latent Semantic Indexing）[3] を用いる．LSI では文章とそれに含まれる単語の出現回数を共起行列として表現するが，本手法では文章に部品を，単語に特徴を対応させる．LSI を用いることで，表記のゆれや類義語などの曖昧さを許容した検索が行われる．

以下では各手順について説明する．

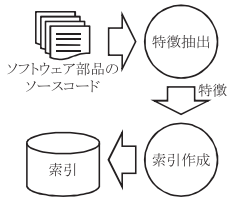

図 1　索引作成処理の流れ
Fig. 1　Outline of indexing process.

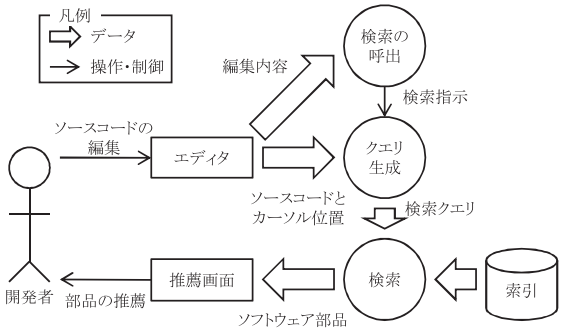

図 2　推薦処理の流れ
Fig. 2　Outline of recommendation process.

3.1　索引作成 1：特徴の抽出

ここでは，検索対象である部品群のソースコードを解析して特徴を抽出する．まず，特徴について説明し，次に各部品から特徴を抽出する方法について述べる．

本手法で扱う部品の特徴とは，3 章冒頭で述べたとおり，ドキュメントコメントや通常のコメント，識別子を構成する単語である．特徴を抽出される識別子は，宣言しているクラス，メソッド，フィールド，ローカル変数名，仮引数の名前と，呼び出されているメソッドの名前である．識別子は部品のソースコード全体に偏在しており，その役割や機能に関連した名前をつけられるため，検索の重要な手がかりとなる．また，ドキュメントコメントや通常のコメントは開発者がその部品に関する情報を記したものであるため，機能や用途などコメント外には明示されない重要な情報を持っている．

具体的な特徴の抽出処理は以下のとおりである．まず，部品のソースコードを構文解析し，クラスごとにコメントと識別子から単語を抽出する．次に，抽出した単語のうち，検索に有用でない語を除去する．最後に，単語を正規化する．

コメントや識別子からの単語の抽出は以下のように行う．

コメントからの抽出　コメントに含まれる文字列をスペース・ピリオド・カンマ区切りで語に分割する．ドキュメントコメントから抽出する場合には，HTML タグや実体参照（& など），@記号から始まる@タグを除去する．

識別子からの抽出　最初に，部品の特徴を表すと考えられる識別子を取り出す．クラス，メソッドおよび変数の宣言から，宣言されているクラス名，メソッド名，メソッドの仮引数名，変数名の識別子を取り出す．また，メソッド呼び出しから，呼び出されているメソッド名の識別子を取り出す．

次に，取り出した識別子を CamelCase [⋆1] に従っているか，アンダーバーによって区切られた単語の列と見なして，各単語に分割する．

たとえば，`ClassName` という識別子は `Class` と `Name` に，`CONSTANT_NAME` という識別子は `CONSTANT` と `NAME` に分割する．

抽出した特徴は，部品と対応付けられる．まず，識別子や通常のコメントから抽出した特徴は，それを直接包含するクラスの特徴とする．ドキュメントコメントは直後の要素を説明するものであるため，直後にクラスがある場合にはそのクラスの特徴とし，直後にメソッドやフィールドがある場合にはそれが属するクラスの特徴とする．ただし，無名クラスの特徴は，その無名クラスを包含する外部クラスの特徴として扱うこととする．

次に，抽出した語から検索に有用でないと考えられる語を除去する．除去する語は以下のとおりである．

(1)　1 文字だけの語
(2)　1 つのクラスにしか現れない語
(3)　半数以上のクラスに現れる語
(4)　不要語リスト[⋆2]に登録されている語

最後に，単語の正規化として，小文字化とステミング処理を行う．ステミングは語から語幹の部分のみを抽出する処理で，複数形や現在分詞などの語形変化を取り除くことを目的とする．

特徴抽出の例として，**図 3** に示す部品のソースコードから特徴を抽出した場合を**表 1** に示す．下線が引かれた識別子から特徴が抽出される．この例では，メソッド名 `selectAll` に含まれる `all` は不要語リストに含まれていたため除去される．また，メソッドの仮引数名 `e` は 1 文字だけの語なので除去される．performed はステミングによって末尾の ed が接尾辞として取り除かれ，perform となる．

3.2　索引作成 2：索引の作成

抽出された特徴から，検索に用いる索引を作成する．索引は得られた特徴を LSI [3] の手法に従って変換したものである．LSI では部品と特徴の共起関係を，潜在的なトピックを用

⋆1 CamelCase は Java の言語標準で推奨されている命名規則で，複数の単語からなる名前において，各単語の先頭だけを大文字にする方法である．ただし，頭字語はすべて大文字で記述する．
⋆2 以下の URL から入手できる不要語リストに，Java の予約語を加えたものを用いた．
ftp://ftp.cs.cornell.edu/pub/smart/english.stop

```
1: import java.awt.event.*;
2:
3: class SelectAllAction implements ActionListener {
4:   public void actionPerformed(final ActionEvent e) {
5:     JEditCommanderTable target =
6:             JEditCommanderPlugin.leftTable;
7:     target.selectAll();
8:   }
9: }
```

図 3　部品のソースコードの例
Fig. 3　Source code of an example component.

表 1　図 3 から抽出した特徴
Table 1　Characteristics extracted from Fig. 3.

特徴	出現頻度
select	2
action	2
perform	1
target	1

いて「部品—トピック」の関係と「トピック—特徴」の関係に分解し，これらを索引として扱う．

索引作成処理では，まず各部品の特徴を基に，「部品—特徴」の共起行列 A を作成する．この行列の列は部品に，行は特徴に対応する．特徴の数を N，部品の数を M とすると，A は $N \times M$ 行列となる．行列の ij 要素は j 番目の部品における i 番目の特徴の出現頻度である．また，この行列の N 次元列ベクトルを部品ベクトルと呼ぶ．この共起行列を，LSI の手順に従って変換することで索引を得る．

3.3　推薦 1：検索の開始

開発者によるソースコードの編集を監視し，特定の時点で検索処理を開始する．

本手法では特徴が変化しない限り検索条件が変化しないため，特徴が変化したときにのみ検索を行う．また，検索の頻度が高すぎると，検索結果の画面が頻繁に更新されて開発者の邪魔になる可能性があるため，あまりに頻繁に検索を行うべきではない．以上をふまえ，検索を行うタイミングを次のように定めた．

- 文の区切りを表すセミコロンが入力されたとき
- 文の変更後に別の文へカーソルが移動したとき
- コメントの終了を表す */ が入力されたとき
- コメントの変更後にコメント外へカーソルが移動したとき

これらをソースコードの編集がある程度完了したときと考え，以下では**編集の区切り**と呼ぶ．編集の区切りが検出されたら検索クエリの生成を行う．

3.4　推薦 2：検索クエリの生成

編集中のソースコードから特徴を抽出し，検索条件を指定する検索クエリを生成する．検索クエリは，組<特徴，重み>の集合である．重みは，対応する特徴が重要なほど大きな値をとる実数値であり，ソースコード上の現在編集中の位置（カーソル位置）と各特徴の現れる位置が近いほど大きな値をとる．これにより，カーソル位置に近い特徴を重視した検索を行う．

検索クエリの元になる特徴は，編集中のソースコードを 3.1 節の索引作成 1 と同様に解析することによって得る．ただしこのとき，各特徴にはカーソル位置からの距離に応じた重みを付ける．複数回出現する特徴の重みは，各出現に対する重みの総和とする．特徴の出現に付けられる重みは，次式で表される．

$$W(l_f, l_c) = max\{1.0 - abs(l_f - l_c)/\alpha, 0.1\}$$

ただし，l_f は特徴が出現した行番号，l_c はカーソル位置の行番号であり，α は検索に用いる範囲を指定する定数である．この式により，カーソル位置に近い行から抽出された特徴には大きな重みが与えられ，α 行よりも遠い特徴には最も小さい重みである 0.1 が与えられる．以下の記述ではすべて $\alpha = 5$ とした．

例として，表 1 のソースコードでカーソルが 7 行目にある場合の検索クエリを以下に示す．

$$\{<\text{select}, 1.2>, <\text{action}, 0.6>, <\text{perform}, 0.4>, <\text{target}, 0.6>\}$$

3.5　推薦 3：部品の検索

3.2 節の索引作成 2 で作成した索引を用いて，LSI の手法に従って検索クエリに合う部品を検索し，順位付けを行う．

まず検索クエリを擬似部品ベクトルに変換する．擬似部品ベクトルとは，検索クエリに含まれる特徴の重みを部品ベクトルと同じ順で並べた M 次元ベクトルである．

次に，索引から，擬似部品ベクトルとの類似度が高い順に部品ベクトルを数個取り出す．

最後に，取り出された部品ベクトルの順位を，利用関係に基づく部品の重要度である ComponentRank [5] を利用して改善する．具体的には，擬似部品ベクトルとの類似度の順位と ComponentRank 値による順位を，Borda の手法[2] によって統合する．

3.6　推薦 4：開発者への推薦

検索された部品の一覧を開発者に提示する．開発者は，一覧から必要に応じて部品の詳細を調べ，再利用を行うかどうかを判断する．

4.　部品自動推薦システム A-SCORE

提案手法を実装した部品自動推薦システム A-SCORE（Automatic Software Component Recommendation Environment）を作成した．A-SCORE の動作画面を**図 4** に示す．開発者が右上部のエディタでソースコードを編集すると，それに応じて右下部の推薦画面に部品が提示される．また，本システムは部品の吟味と開発中プロジェクトへの適用を支援するための機能も持っている．

4.1　システムの構成

システムの構成と検索処理におけるデータの流れを**図 5** に示す．システムはクライアントとサーバからなっており，部品はサーバ上にある部品データベースで一元的に管理されている．サーバはウェブサービスフレームワークである Apache Axis2 [*1] を用いてウェブサービスとして実装した．クライアントは統合開発環境 Eclipse [*2] を A-SCORE プラグインによって拡張したものである．

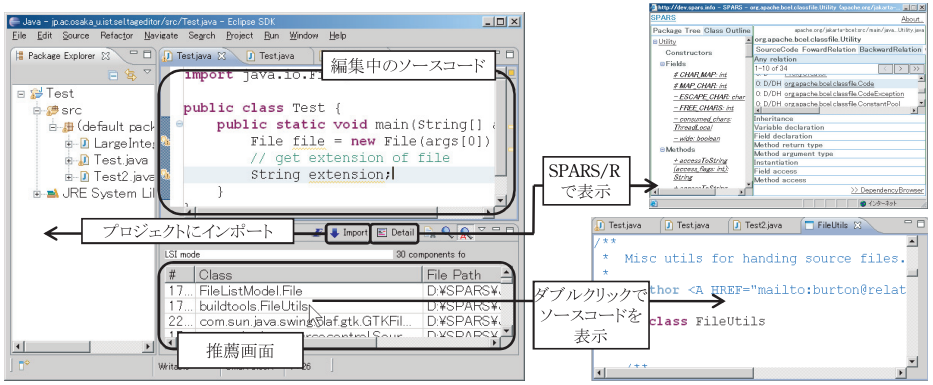

図 4　A-SCORE の動作画面
Fig. 4　Screenshot of A-SCORE.

A-SCORE は開発の効率化や推薦結果の改善のために，内部で部品検索システム SPARS/R を利用している．SPARS/R は SPARS-J の後継として開発されているシステムであり，A-SCORE と同様に Java で開発されているため，SPARS/R の機能を A-SCORE から簡単に利用することができる．以降，SPARS/R を単に SPARS と呼ぶ．

表 2 に A-SCORE の規模を示す．行数には利用したライブラリや SPARS のソースコードは含まれていない．A-SCORE の開発は第 1 著者 1 人で行い，約 1 年を要した．

4.2　A-SCORE の持つ自動推薦以外の機能

A-SCORE はユーザの利便性を高めるため，推薦された部品を表示する以外に以下のような機能を持っている．

インポート機能　この機能は，推薦された部品を現在開発中のプロジェクトに簡単に取り込むためのものである．部品を選択して Import ボタン（図 4 左の矢印）を押すことで，

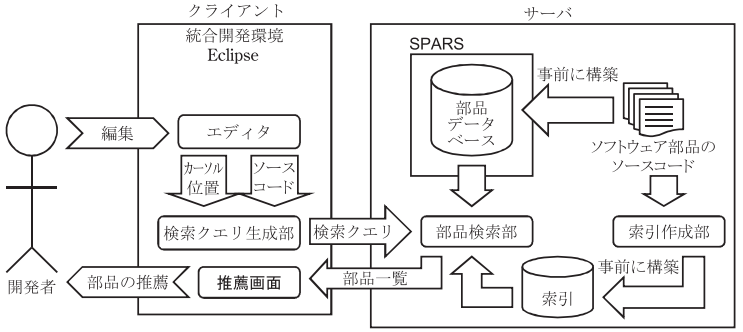

図 5　A-SCORE の構成とデータの流れ
Fig. 5　Architecture and data flow of A-SCORE.

表 2　A-SCORE の規模（行数）
Table 2　Size of A-SCORE (in LOC).

モジュール名	規模
部品検索部	3,439
索引作成部	716
検索クエリ生成部	987
推薦画面	1,722
全体	6,864

⋆1 http://ws.apache.org/axis2/
⋆2 http://www.eclipse.org/

選択した部品がプロジェクトに取り込まれる．

SPARS リンク　この機能は，推薦された部品に関する詳細な情報を，簡単に SPARS で閲覧するためのものである．部品を選択して Detail ボタン（図 4 右上の矢印）を押すことで，SPARS 上で部品の詳細情報が表示される．この機能を利用することで，推薦された部品のサンプルコードを見つけられる．また，自動推薦で得られた情報をもとに，より適切な部品を探すために利用することもできる．

部品のソースコード表示　この機能は，部品のソースコードを閲覧するためのものである．推薦された部品をダブルクリックすると，部品のソースコードが構文強調された状態で表示される．この機能を用いることで，部品をインポートする前に，部品の詳細な実装などを確認することができる．

簡易クロスリファレンサ　この機能は，上記のソースコード表示機能で表示したソースコード上で，ソースコード内で使用されているクラス名から，そのクラスを宣言しているソースコードに簡単にアクセスするためのものである．この機能により，ソースコードをより簡単に閲覧し，理解することができる．クラスを宣言しているソースコードは，A-SCORE の部品データベースに登録されている必要がある．

5. 実　　験

本章では A-SCORE の有効性を評価するために行った実験について説明する．提案手法は一般の検索システムと異なり，自動的にクエリを生成することで，ユーザの操作によらず頻繁に検索を行う．このため，一般の検索システムの評価に用いられる適合率や再現率といった尺度を使用して，一般のシステムと提案手法を比較評価することはできない．

そこで，本実験では，A-SCORE を使用することによって開発効率と再利用頻度，完成したプログラムの品質がどのように変化するかを調査した．実験では，A-SCORE の有無による影響を比較するため，被験者は**表 3** に示すように分かれ，指定した課題を行った．

5.1 実 験 内 容

実験にあたって，3 つの課題（うち 1 つは練習課題）を用意した．すべての課題は，Java で書かれた半完成プログラムに，指定した機能を実装する形式となっている．なお，A-SCORE ではソースコード中の識別子がクエリとして使われるため，半完成プログラムに含まれるクラス名やメソッド名は意味のないものにしてある．A-SCORE 以外に被験者が作業中に参照できるのは，以下に示した情報のみとした．

- 課題の説明資料

表 3　課題割当て
Table 3　Task assignment for each participant.

被験者	A	B	C	D
1 番目の課題	課題 α A-SCORE あり	課題 β A-SCORE あり	課題 α A-SCORE なし	課題 β A-SCORE なし
2 番目の課題	課題 β A-SCORE なし	課題 α A-SCORE なし	課題 β A-SCORE あり	課題 α A-SCORE あり

- Java の API 資料である Javadoc
- キーワードによる部品検索システム SPARS

以下，課題の内容，被験者，課題の作業環境について詳細に説明する．

課題の内容　被験者は各課題で，欠陥を含まない半完成プログラムを与えられ，未実装部分に適切な記述を行う．それぞれの課題にはいくつかのテストケースが用意されており，すべてのテストケースに合格することを課題終了条件とした．各課題の作業内容は以下のとおりである．

練習課題　画面全体のスクリーンショットをファイルに保存する部品を作成する．

課題 α　氏名と試験の成績が記録された CSV ファイルを読み込み，成績上位者の名前を取り出す部品を作成する．読み込む CSV の仕様として，コンマ文字を含む列を表現するためには，列全体を二重引用符で括ることとした．

課題 β　ファイルコピー機能を持つ部品を作成する．作成する部品に渡される引数は，コピー元とコピー先のファイル名を表す `java.io.File` 型のオブジェクトである．コピーを行う際には，ファイルの内容だけではなく，更新日時も元のファイルと同じものに設定しなければならない．コピー先にすでにファイルが存在するときには，そのファイルを上書きするものとした．

被験者　被験者は，ソフトウェア工学分野の大学院生および学部生の計 4 人である．被験者らは学部 3 年次の演習もしくは研究で Java を使用しており，言語に対する知識を持っている．また，作業環境に対する慣れの影響を減らすため，被験者は事前に A-SCORE の使用方法について講習を受けた．

検索対象　被験者が用いる A-SCORE の索引と SPARS のデータベースは以下の部品集合から構成した．

- JDK1.6 に付属の標準ライブラリのソースコード

- The Apache Software Foundation [⋆1]で公開されているライブラリのソースコード
- SourceForge.net [⋆2]で公開されているアプリケーションのソースコード

この部品集合は約 27 万個の部品からなり，課題 α と課題 β の解決に直接利用できる部品を，少なくともそれぞれ 5 個と 10 個含んでいる．この部品集合に対する A-SCORE の索引作成処理に必要な時間は約 1 日であり，推薦処理に必要な時間は約 1 秒であった．

課題の作業環境　被験者の作業状況は逐次記録されており，実験終了後に作業手順を確認することができる．具体的には，以下の情報が時刻付きで保存される．

- すべての検索クエリおよび検索結果
- 閲覧された部品のソースコード
- 5 秒ごとの画面全体のスクリーンショット

5.2 実験手順

実験は以下に示す手順で行った．

(1) 最初にすべての被験者が A-SCORE を利用して練習課題を行う．この作業は，被験者を A-SCORE と SPARS に慣れさせることを目的としており，作業中にツールの利用方法や部品の効率の良い探し方などについて適宜指導を行った．

(2) 表 3 に示す順に各被験者が課題を行う．

(3) 被験者にアンケートをとり，記録された作業状況や完成したソースコード，アンケート結果をもとに評価を行う．

5.3 実験結果

実験結果を**表 4**，**表 5**，**表 6**，**表 7** に示す．表 4 は，各課題の成果物の規模を行数で表したものである．作業前の列は課題の開始時に渡した半完成プログラムの規模を表し，括弧内は開始時からの増減を表す．テストケースのプログラムは行数に含まれていない．表 5 は，各課題に要した時間と作業ごとの内訳である．時間は，課題のソースコードを開いてからテストケースに合格するまでを測定した．表 6 は，完成したプログラムに含まれていた不具合

表 4　実験結果（成果物の規模（行））（括弧内は作業前からの増減）
Table 4　Size of output source (in LOC).

課題	作業前	被験者 A	被験者 B	被験者 C	被験者 D
α	24	205(+181)	111(+87)	93(+69)	96(+72)
β	54	74(+20)	97(+43)	130(+76)	98(+44)

⋆1 http://projects.apache.org/
⋆2 http://sourceforge.net/

表 5　実験結果（作業時間（秒））
Table 5　Consumed time (in second).

A-SCORE の使用			有		無	
課題 α	被験者		A	D	B	C
	作業時間	A-SCORE のソースコード閲覧時間	940	290	—	—
		SPARS で部品を検索していた時間	255	0	0	0
		Javadoc の閲覧時間	80	505	1,225	160
		その他	4,775	10,160	7,830	3,695
		合計	6,050	10,955	9,055	3,855
課題 β	被験者		B	C	A	D
	作業時間	A-SCORE のソースコード閲覧時間	940	265	—	—
		SPARS で部品を検索していた時間	75	0	610	0
		Javadoc の閲覧時間	790	80	220	1,525
		その他	4,530	1,270	1,035	4,505
		合計	6,335	1,565	1,865	6,030
合計時間の平均			6,226.25		5,201.25	

表 6　実験結果（不具合の有無）
Table 6　Output source code includes faults or not.

A-SCORE の使用			有		無	
課題 α	被験者		A	D	B	C
	不具合の種類	close 忘れ	有	有	有	有
		例外処理忘れ	有	有	有	有
課題 β	被験者		B	C	A	D
	不具合の種類	close 忘れ	無	無	有	無
		例外処理忘れ	無	有	有	有
		バイナリファイルに非対応	無	有	有	有

表 7　実験結果（再利用部品数）
Table 7　Number of reused components.

A-SCORE の使用			有		無	
課題 α	被験者		A	D	B	C
	再利用の種類	A-SCORE で部品をインポート	1	0	–	–
		部品の一部をコピー&ペースト	0	0	0	0
		再利用する部品の手がかりが得られた	1	0	0	0
		サンプルコードとして再利用した	0	1	0	0
課題 β	被験者		B	C	A	D
	再利用の種類	A-SCORE で部品をインポート	0	0	–	–
		部品の一部をコピー&ペースト	2	0	0	0
		再利用する部品の手がかりが得られた	0	0	0	0
		サンプルコードとして再利用した	1	0	0	0

の有無である．各不具合は，完成したソースコードを目視で検査して判定した．表 7 は，課題の作業中に再利用を行った部品の数である．以下，表 6，表 7 について詳しく説明する．

表 6 は，完成したプログラムに含まれていた不具合の種類を表す．含まれる不具合が少ないほど，成果物の品質は高い．以下，不具合の種類について説明する．**close 忘れ**は，入出力処理の後にファイルを閉じる処理が欠けていることを示す．**例外処理忘れ**は，例外発生時にオブジェクトを適切に破棄するための処理が欠けていることを示す．**バイナリファイルに非対応**は，ファイルコピーにおいて，テキストモードで処理を行っているためバイナリファイルを正しく処理できないことを示す．

表 7 は，課題の作業中に再利用を行った方法とその回数を表す．再利用の回数が多いほど，再利用が活発に行われたことを示す．表中の再利用の種類は，それぞれ以下の意味である．**A-SCORE で部品をインポート**は，A-SCORE のインポート機能を用いて部品をプロジェクトにインポートし，そのまま，または改変して再利用を行った部品の数を示す．**部品の一部をコピー&ペースト**は，部品のソースコードから必要な機能のみをコピー&ペーストして再利用を行った部品の数を示す．**再利用する部品の手がかりが得られた**は，部品のソースコードを閲覧することで，再利用すべき部品の存在や名前に気付いた件数を示す．**サンプルコードとして再利用した**は，再利用したい部品を利用している別の部品を発見し，再利用したい部品の利用方法を知ることができた件数を示す．

5.4　分析と考察

実験結果の表 6 および表 7 から，A-SCORE を利用したほうが再利用部品数は多く，不具合の数は少なくなるため，成果物の品質は良くなる傾向にあることが分かった．特に A-SCORE を利用しなかった場合はまったく部品が再利用されておらず，A-SCORE による再利用促進の効果が示された．一方，表 5 からは，A-SCORE を利用した場合に作業時間が長くなる傾向が読み取れる．しかし，A-SCORE を利用しなかった場合の成果物には不具合が多く存在したため，本来は不具合の修正のためにより長い作業時間が必要であったと考えられる．

5.4.1　作業手順の分析

また，再利用の手順を詳細に観察したところ，A-SCORE が CodeBroker よりも多くの再利用を支援できることが分かった．表 7 の A-SCORE を用いて再利用が行われた部品のうち，CodeBroker が推薦可能であった事例は被験者 B が課題 β で行ったコピー&ペーストによる再利用の 1 件のみであった．

A-SCORE による影響を詳しく調べるために，さらに作業プロセスを詳細に調査した．その結果，被験者は部品を調べるための情報源（Javadoc，SPARS，A-SCORE）のうち SPARS をほとんど利用していないことが分かった．そこで被験者になぜ SPARS を利用しなかったのかをインタビューしたところ，以下のような回答が得られた．

- 自分の望む形で利用できる既存部品があるとは思わなかった．
- 検索するよりも部品を自作したほうが早いと判断した．

この結果から，部品検索システムが十分に活用されていないことが分かる．このことと，A-SCORE を用いた場合には再利用が行われたことから，A-SCORE の自動推薦が有用な情報を提供できたことが分かる．

一方で，インポートして再利用した部品は 1 つしかなく，インポート機能が有効に活用されていないことが分かった．そこで作業プロセスを調査したところ，部品をいったんインポートした後に，依存関係を解決できなかったため部品の利用を諦めた事例があった．依存関係にある部品を一括してインポートする機能を作成することで，より簡単に再利用を行えるようになると考えられる．

5.4.2　検索結果の分析

別の観点として，部品の推薦がうまく働く場合と働かない場合を調べるため，A-SCORE の検索結果を調査した．

まず，A-SCORE を使用した場合の作業履歴をすべて観察し，推薦がうまく働かなかった場合がないかを調査した．その結果，被験者 D が課題 α の作業中に，課題の達成に有用でない自然言語処理の部品が非常に多く推薦され[⋆1]て推薦リストの上位を占めてしまい，有用な部品が見つからないという問題が，長い期間にわたって起こっていた．**図 6** は最初にその問題が起きたときのソースコードであり，カーソルは 16 行目にあった．このソースコードに対して推薦された部品のうち語幹抽出の部品が占めた割合は 57%であり，上位 10 件に限ると 80%（3 位〜10 位）を占めていた．これらの部品は自動生成されたものであったため互いに非常に類似しており，同時に推薦されてしまったものと考えられる．この問題の解決法として，類似した部品を 1 つのグループとして推薦することや，システムの利用者が不要と判断した部品を推薦から取り除くことができるフィルタリング機能を実装することが考えられる．

次に，推薦された部品を直接利用できた被験者 A と B の作業履歴を観察し，どの程度の

⋆1 org.tartarus.snowball.ext パッケージに所属するステミング（語幹抽出）を行う部品．言語ごとに別のクラスとして実装されていた．

```
 1: package exp.first;
 2:
 3: import java.io.BufferedReader;
 4: import java.io.File;
 5: import java.io.FileInputStream;
 6: import java.io.FileReader;
 7: import java.io.IOException;
 8: import java.io.InputStreamReader;
 9: import java.util.regex.Pattern;
10:
11: public class CsvAnalyzer {
12:   /* Get name and scores from csv file,
13:    * Analyze csv file,
14:    * Counts score,
15:    * Select high score name
16:    */
17:   public static String[] getHighScoreAnalyzingCsvFile(File csvFile, int
    count) throws IOException {
18:     BufferedReader br = new BufferedReader(new InputStreamReader(new
    FileInputStream(csvFile)));
19:     Pattern border = Pattern.compile(",");
20:     String line = null;
21:     while ((line=br.readLine()) > -1) {
22:
23:     }
24:     return null;
25:   }
26:   /**
27:    * Split line to String by comma.
28:    * @return
29:    */
30:   private String[] getSplittedWord() {
31:
32:   }
33: }
```

図 6　被験者 D で推薦がうまく働かなかったときのソースコード
Fig. 6　Snapshot of source code when A-SCORE failed to recommend reusable components for participant D.

記述を行った時点で再利用した部品が推薦されたかを調べた．その結果，それぞれ，被験者 A はメソッドのシグネチャとファイルを開く処理を書いた時点で，被験者 B はメソッドのシグネチャのみを書いた時点で，再利用した部品が推薦されていた．ただし，被験者 A は推薦された部品をインポートした直後に，より有用な部品が推薦されたため，前者の部品を捨てて後者の部品を利用することとしていた．このことから，適切な記述を行った場合には，十分に早い段階で有用な部品の推薦が行われることが分かった．**図 7** に被験者 B の例を示す．推薦が行われたのは，`copyFileSimply` メソッドの第 1 引数の名前を変更した直後であり，カーソルは 7 行目にあった．このとき，推薦の第 3 候補に，ファイルコピー機能を有する部品 phex.utils.FileUtils が推薦された．

```
 1: package exp.second;
 2:
 3: import java.io.File;
 4: import java.io.IOException;
 5:
 6: public class FileCopy {
 7:   public static void copyFileSimply(File sourceFile, File param2) throws
    IOException {
 8:     //copy file simply
 9:   }
10:
11:   public static void copyFileExcludeComments(File srcFile, File param2)
    throws IOException {
12:   }
13: }
```

(1)　org.apache.commons.vfs.tasks.AbstractSyncTask
(2)　org.romaframework.core.io.virtualfile.physical.PhysicalFile
(3)　phex.utils.FileUtils
(4)　org.apache.commons.jci.monitor.FilesystemAlterationListener
(5)　org.smartfrog.examples.orchdws.filetester.FileTester
…

図 7　被験者 B が再利用を行ったときのソースコードと推薦部品（下線は再利用に適した部品）
Fig. 7　Snapshot of source code and recommended components when A-SCORE recommended reusable components for participant B (The underlined component was reused).

今回の実験では早い段階で部品の推薦が行われていたが，プログラムの書き方によっては，ほとんどの機能を実装した後にしか部品が推薦されない場合も考えられる．このような場合であっても，再利用を行ったほうが成果物の欠陥が少ないため，積極的に推薦された部品を利用したほうが良いと考えられる．

5.4.3　キーワード検索を用いた場合の結果

実験では SPARS の検索結果を用いた再利用が行われなかったため，キーワード検索との直接的な比較が行われていないと考えられる．そこで，SPARS でキーワード検索を行った

場合の上位 10 件における適合率を計測した．

それぞれの課題に直接関係するキーワードで検索を行った結果を記す．課題 α に対応する検索を行った場合，「CSV Reader」というキーワードに対しては適合率 0.2 であり，「CSV Parser」というキーワードに対しては適合率 0.1 だった．一方，課題 β に対応する検索結果は，「copy file」に対しては適合率 0.2 であり，「copyfile」という 1 語で検索を行った場合は適合率 0.8 であった．

このように，キーワード検索では入力するキーワードによって結果が大きく変化する．そのため，良いキーワードを考えつくことができれば良い部品を入手できるが，そうでない場合には良い部品にたどりつくまでに長い時間を要してしまう．

5.5　アンケート結果

ツールの使い勝手などについて定性的な評価を行うために，実験後に行ったアンケートの結果を**表 8** に示す．自動推薦機能は評価値の最頻値が 5 であり，高く評価されたといえる．他に高評価だった機能としては，ソースコード表示と簡易クロスリファレンサがあった．これらの機能が高評価を得た理由としては，編集中のソースコードに対する Eclipse の同様の機能と使い勝手が似ている点や，手軽に利用できる点が考えられる．

また，ツールの良い点と改善すべき点について自由記述のアンケートも行った．良い点としては，「今まで知らなかった部品が推薦されて便利」「自動で部品が出てくるため検索の手間が省ける」などがあげられた．一方，改善すべき点としては，「コーディング開始後しばらくは欲しい部品がなかなか出てこない」などの問題が指摘された．この問題の原因は，コーディング開始直後にはソースコードに特徴が少ないため，部品の機能と関連の低い一部の特徴の影響が大きく現れ，類似部品検索がうまく働かないためだと考えられる．ソースコード表示機能については，「手軽に閲覧できて便利」というコメントが得られた一方，「ハイライト機能[*1]やアウトライン表示など Eclipse 標準のエディタと同じ機能を使えるようにしてほしい」という改善点が指摘された．

表 8　アンケート結果（5 を最良とする 5 段階評価）
Table 8　Inquiry into usability of A-SCORE.

被験者		A	B	C	D	最頻値
UI は使いやすかったか		4	2	4	4	4
推薦された部品は役に立ったか		4	4	3	4	4
各機能は便利だったか	自動推薦	4	5	5	3	5
	メトリクス表示	1	1	4	1	1
	ソースコード表示	4	4	4	4	4
	SPARS 連携	2	2	3	4	2
	簡易クロスリファレンサ	3	5	4	4	4
	インポート	5	1	1	4	1

再利用を行わなかった被験者 C が，「推薦された部品は役に立ったか」という質問に対し，3 という評価を下しており，疑問となった．この点について本人に理由を問い合わせたところ，「推薦された部品のリストを眺めていると勉強になったためである」という回答が得られた．

5.6　妥当性の検討

ここでは実験結果の妥当性に影響を及ぼす可能性のある問題について述べる．

最初に，A-SCORE 以外の原因で，再利用数増加や品質向上が起こった可能性について考える．まず，課題の順序の影響が考えられるが，本実験では初めに A-SCORE を使用する被験者と後で使用する被験者に分割した（表 3）ことで順序の影響を排除した．また，被験者が利用可能なツールを指定したことで，積極的に A-SCORE を利用するバイアスが働いた可能性がある．この点については，他に利用可能であった SPARS に比べても長い時間使用されていた（表 5）ことから，問題ないと考えられる．また，使用可能なツールが不十分であったため，作業結果に影響を及ぼした可能性がある．しかし，一般にソフトウェア開発で用いられる情報源であるリファレンスマニュアルと仕様書，検索システムは実験でも提供されている．SPARS は一般のキーワード検索システムよりも検索結果の精度が高いことが分かっており[5]，再利用可能な部品も十分に提供されている．また，統合開発環境として広く用いられている Eclipse を使用しているため，ツールに不足はないものと考えられる．

一方，被験者が A-SCORE に慣れていないため操作ミスなどを起こし，A-SCORE を利用した場合の結果が悪くなった可能性もある．この点については，被験者全員に対して事前に A-SCORE の使い方について講習を行い，さらに他の課題に先立って A-SCORE を使用する練習課題を行うことで影響を軽減する措置をとった．

また，実験設定の一般性については，被験者の熟練度に偏りがある可能性が考えられる．具体的には，被験者が課題の分野（入出力処理や例外処理）に慣れていないため，再利用を行わずに実装した場合に不具合が多くなった可能性がある．また，被験者が再利用を重視した開発に慣れていないため，再利用部品数が少なくなった可能性もある．しかし，本実験の被験者は，Java プログラミングに関する知識や部品の再利用をともなう開発の経験を持っていたことや，A-SCORE を用いた場合には再利用が行われていたことから，問題は大きく

*1 変数やメソッドなどを選択すると，同じ変数やメソッドが出現する箇所がハイライトされる機能

ないと考えられる．ただし，大規模プロジェクトのような複数人での開発の経験を持つ被験者はいなかったため，今後の課題として経験者による実験を行う必要があると考えられる．

6. ま と め

本稿では，部品を変更せずに再利用する場合だけではなく，変更を加えて再利用する場合やコード片を再利用する場合にも対応した部品の自動推薦手法を提案した．本手法は，ソースコード中に数多く現れるコメントや識別子を利用し，LSI によって曖昧さを許容する検索を行う．また，本手法を実装した部品の自動推薦システム A-SCORE を作成した．

さらに学生を被験者として A-SCORE の評価実験を行った．その結果，A-SCORE を用いることで部品の再利用が促進され，プログラムの品質が向上することが分かった．

今後の課題としては，まず評価実験の充実があげられる．複雑なフレームワークを利用する場合の実験として，Eclipse プラグイン開発に関する課題も用意することを考えている．また，実験結果の信頼性を向上させるため，被験者の人数を増やし，様々な熟練度の被験者を対象に実験を行うことも考えている．ほかには，推薦した部品のソースコードを閲覧する際の利便性の向上があげられる．たとえば，アンケートで指摘された，ソースコード表示におけるアウトライン機能の実装などを考えている．

謝辞　本研究は，日本学術振興会科学研究費補助金萌芽研究（課題番号：18650006）の助成を得た．本研究の一部は，文部科学省グローバル COE プログラム（アンビエント情報社会基盤創成拠点）の補助によるものである．ここに記して謝意を表す．

参 考 文 献

1) Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P. and Lopes, C.: Sourcerer: A search engine for open source code supporting structure-based search, *Proc. Dynamic Languages Symposium*, pp.681–682 (2006).
2) de Borda, J.C.: Memoire sur les Elections au Scrutin, *Histoire de l'Academie Royale des Sciences*, Paris (1781).
3) Deerwester, S., Dumais, S., Furnas, G., Landauer, T. and Harshman, R.: Indexing by latent semantic analysis, *Journal of the American Society for Information Science*, Vol.41, No.6, pp.391–407 (1990).
4) Hummel, O., Janjic, W. and Atkinson, C.: Code Conjurer: Pulling Reusable Software out of Thin Air, *IEEE Software*, Vol.25, No.5, pp.45–52 (2008).
5) Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE Trans. Softw. Eng.*, Vol.31, No.3, pp.213–225 (2005).
6) Krueger, C.: Software reuse, *ACM Computing Surveys*, Vol.24, No.2, pp.131–183 (1992).
7) Landauer, T. and Dumais, S.: A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge, *Psychological Review*, Vol.104, No.2, pp.211–240 (1997).
8) Selby, R.: Enabling Reuse-Based Software Development of Large-Scale Systems, *IEEE Trans. Softw. Eng.*, Vol.31, No.6, pp.495–510 (2005).
9) Ye, Y. and Fischer, G.: Reuse-Conducive Development Environments, *Automated Software Engineering*, Vol.12, No.2, pp.199–235 (2005).
10) 鷲崎弘宜，深澤良彰：有向置換性類似度に基づくコンポーネント検索方式の実現と評価，情報処理学会論文誌，Vol.43, No.6, pp.1638–1652 (2002).

(平成 21 年 3 月 31 日受付)
(平成 21 年 9 月 11 日採録)

島田　隆次（正会員）
平成 19 年大阪大学基礎工学部情報科学科卒業．平成 21 年同大学大学院情報科学研究科博士前期課程修了．修士（情報科学）．在学中，ソフトウェア部品検索の研究に従事．現在東芝ソリューションに勤務．

市井　　誠（正会員）
平成 16 年大阪大学基礎工学部情報科学科卒業．平成 21 年同大学大学院情報科学研究科博士後期課程修了．博士（情報科学）．在学中，ソフトウェア部品検索の研究に従事．現在日立製作所組込みシステム基盤研究所に勤務．

早瀬　康裕（正会員）

平成 14 年大阪大学基礎工学部情報科学科卒業．平成 19 年同大学大学院博士後期課程修了．現在，同大学特任助教．博士（情報科学）．オープンソースソフトウェア開発，ソフトウェア保守の研究に従事．IEEE-CS 会員．

松下　　誠（正会員）

平成 5 年大阪大学基礎工学部情報工学科卒業．平成 10 年同大学大学院博士後期課程修了．同年同大学基礎工学部情報工学科助手．平成 14 年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助手．平成 17 年同専攻助教授．平成 19 年同専攻准教授．博士（工学）．ソフトウェア開発環境，リポジトリマイニングの研究に従事．日本ソフトウェア科学会，ACM 各会員．

井上　克郎（フェロー）

昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院博士課程修了．同年同大学基礎工学部情報工学科助手．昭和 59〜61 年ハワイ大学マノア校情報工学科助教授．平成元年大阪大学基礎工学部情報工学科講師．平成 3 年同学科助教授．平成 7 年同学科教授．平成 14 年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教授．平成 20 年国立情報学研究所客員教授．同年情報処理学会フェロー．同年電子情報通信学会フェロー．工学博士．ソフトウェア工学の研究に従事．日本ソフトウェア科学会，電子情報通信学会，IEEE，ACM 各会員．