2008－SE－162 (5)
2008／11／19

ソースコードの編集内容を用いたソフトウェア部品の自動推薦手法

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

クラスやメソッドなどソフトウェアの構成要素であるソフトウェア部品を再利用することで，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．開発者はソフトウェア部品検索システムを用いたキーワード検索により再利用可能なソフトウェア部品を得ることができる．しかしキーワード検索を用いた再利用には，開発者が意識しないと検索が行われないなどの問題点がある．それに対して，開発者の指示なしに自動的にソフトウェア部品を検索する手法が提案されているが，その手法では入出力仕様が完全に一致するものを検索するため，入出力仕様に変更を加えれば再利用できるような部品は発見できない．そこで本稿では，そのようなソフトウェア部品も検索できるソフトウェア部品の自動推薦手法を提案する．提案手法では，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，自然言語に対する検索手法である LSI を応用して曖昧さを許容する検索を行うことで，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の自動推薦システムを作成した．

Automatic software component recommendation using Coding Context

Ryuji SHIMADA ,[†1] Makoto ICHII ,[†1] Yasuhiro HAYASE ,[†1]
Makoto MATSUSHITA [†1] and Katsuro INOUE[†1]

Reusing software components like classes and modules improves software quality and productivity. Software developers often retrieve available components by keyword search using software component retrieval system. However the developers who will not search for components can not retrieve reusable components. To address this issue, software component retrieval technique without developer's instruction was proposed. However developers cannot discover available components that need change using the technique because it searches components having same input and output specification to requested one. In this paper, we propose the automatic software component recommendation technique that can retrieve such software component. Proposed technique vaguely retrieves reusable components including the ones that requires small modification to reuse based on the LSI technique using comments and identifiers in source code. In addition, we have developed the automatic software component recommendation system that implements the proposed technique.

1. はじめに

ソフトウェア開発においては，ソフトウェア部品を適切に再利用することでソフトウェアの品質や生産性が向上すると言われている[1)]．ソフトウェア部品とは，ソフトウェアの構成要素であるモジュールやクラスなどのことである．

ソフトウェアの生産活動の成果としてソフトウェア部品は世の中に無数に存在しているが，それだけでは再利用による恩恵は得られない．ソフトウェア部品を効率的に再利用するためには，ソフトウェア部品が入手可能な状態で蓄積されていることと，再利用が行いやすいように整理されていることが必要である．しかし実際には，再利用に適した形で整理されていないソフトウェア部品も存在する．入手可能ではあるものの，ドキュメントも整備されておらずそのままでは再利用が難しいソフトウェア部品も数多く存在する．

そういったソフトウェア部品群から再利用可能なソフトウェア部品を効率的に見つけるために，開発者はキーワード検索によるソフトウェア部品検索システムを用いることが多い．しかし，キーワード検索を用いた再利用には開発者が意識しないと検索が行われない

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

などの問題点があるため，Ye らによってソフトウェア部品の自動検索という概念が提案されている[2]．本稿ではそのソフトウェア部品の自動検索に対する新しい手法を提案する．

以下，2節ではソフトウェア部品の再利用について述べる．3節では提案手法の詳細について述べ，4節で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について説明する．5節では提案手法の適用例について述べる．6節では関連研究について述べる．7節でまとめと今後の課題について述べる．

2. ソフトウェア部品の再利用

ソフトウェア部品の再利用とは，ソフトウェア部品のソースコードを新しいソフトウェアの開発に利用することである．1節で述べたように必要なソフトウェア部品を見つけるのは難しいため，開発者はキーワード検索によるソフトウェア部品検索システムを用いて蓄積されているソフトウェア部品群から再利用可能なソフトウェア部品を検索することが多い．キーワード検索は検索したいものに関する語句（キーワード）を指定して検索する検索手法であり，Web 検索などで広く利用されている．ソフトウェア部品検索システムはソフトウェア部品検索に特化した検索システムで，現在までに Koders[3] や Google Code Search[4]，SPARS-J[5),6] などが開発されている．キーワード検索を用いた再利用は以下のようなプロセスとなる．

(1) 開発者がソフトウェアを開発しているときに使えそうなソフトウェア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る
(3) ソフトウェア部品検索システムがキーワードに合致するソフトウェア部品を検索し，開発者に提示する
(4) 開発者がそのソフトウェア部品が使えるかを判断する
(5) 開発者がそのソフトウェア部品を再利用する

しかし，キーワード検索を用いた再利用には問題点が大きく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ものである．上記プロセスの(1)で開発者が能動的に検索を行わないと，利用可能なソフトウェア部品は検索されない．2つ目は，開発者がよく知らないソフトウェア部品は検索しにくいことである．目的のソフトウェア部品を得るためには上記プロセスの(2)で適切なキーワードを選定する必要がある．しかし欲しい機能は分かっていてもその機能を表す言葉が分からないなど，うまくキーワードを書けない場合もある．

これらの問題に対して 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ソフトウェア部品自動検索という概念を提案している[2]．ソフトウェア部品自動検索では，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，利用可能なソフトウェア部品を開発者に提示する．ソフトウェア部品自動検索の利点として，次の二つが挙げられる．

(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，再利用可能なソフトウェア部品があった場合には開発者が意識していなくてもそれが提示されること．
(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書けなくても検索が行えること．

また Ye らはソフトウェア部品自動検索を実現するための検索手法も提案しており，その手法の実装として CodeBroker というシステムを作成している．CodeBroker はメソッドをソフトウェア部品として扱い，メソッドの宣言を書いたときに検索を行う．メソッドの宣言は，メソッドに付随するドキュメントコメント（機能等を説明するコメント）とシグネチャ（引数と戻値の型）を含む．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元に自然言語に対する解析手法である潜在的意味解析法 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ものだけを抽出し，再利用可能なソフトウェア部品として開発者に提示する．

しかし CodeBroker では変更なしで再利用できるソフトウェア部品しか検索できない．CodeBroker はソフトウェア部品の再利用の中でも，特にソフトウェア部品を変更せずにそのまま再利用する場合を対象としている．しかしソフトウェア部品の再利用には，再利用先のプログラムに合わせて変更を加えてから再利用する場合や小さなコード片だけを再利用する場合もあり，大規模なプログラムでは無変更の再利用よりも変更を加えて再利用するほうが多く行われているという報告もある[8]．

これには2つの理由が挙げられる．1つ目はシグネチャの一致による絞込みを行っていることである．CodeBroker は無変更の再利用を想定しているため，シグネチャが一致する，つまり入出力がそのまま使えるものを検索するのは妥当である．しかしソフトウェア部品に変更を加えて再利用する場合には，この絞込みは適切ではない．2つ目は自動検索を行うタイミングの問題である．CodeBroker ではメソッドの宣言を書いた時点で検索を行うが，再利用を行いたい場面はメソッドの宣言を書いたときだけではない．特にコード片の再利用ではメソッドの本体を書いているときに自動検索を行うことが適切であろう．そのためにはメソッド宣言の情報だけではなく，メソッドの本体に関する情報も必要となる．

そこで本稿では，変更を加えて再利用する場合や，コード片の再利用にも対応したソフトウェア部品の自

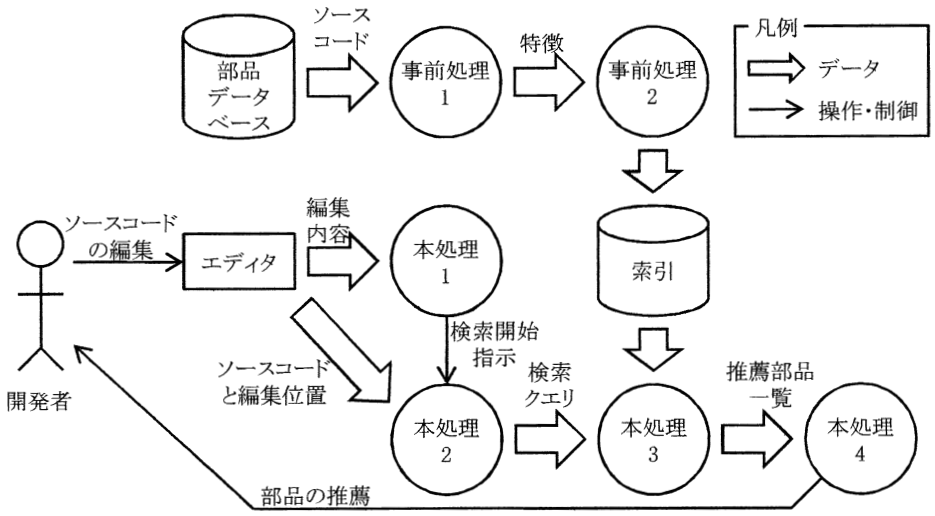

図 1 提案手法の流れ

動推薦手法を提案する．提案手法では，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，曖昧さを許容する検索を行う．

3. 提案手法

本節では提案手法の詳細について述べる．**図 1** に本手法の流れを示す．本手法では Java で書かれたクラスをソフトウェア部品として扱っており，ソフトウェア部品の特徴を元に，ソフトウェア部品のソースコードが格納された部品データベースから編集中のソースコードに類似したソフトウェア部品を検索し，利用可能ソフトウェア部品として推薦を行う．本手法では頻繁に検索を行うので，検索の高速化のために索引を用いた検索を行う．そのため本手法は検索に用いる索引を作成する事前処理と，開発者の編集に応じて自動的に検索を行う本処理からなっている．本手法の処理手順は以下のとおりである．

事前処理 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

事前処理 2 得られた特徴から索引を作成する

本処理 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

本処理 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

本処理 3 索引を用いて検索クエリに合う類似ソフトウェア部品を検索する

本処理 4 検索されたソフトウェア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)[9] を用いる．LSI を用いることで，表記のゆれや類義語など曖昧さを許容した検索が期待できる．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，本手法では文章にソフトウェア部品を，単語に特徴を対応させている．

以下では各手順について説明する．

3.1 事前処理 1: 特徴の抽出

検索の実行に先立って索引を作成するために，検索対象であるソフトウェア部品群のソースコードを解析して特徴を抽出する．特徴とは，ソフトウェア部品のソースコードに含まれる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．識別子はクラス名やメソッド名，フィールド名，ローカル変数名を含む．これらの特徴はソフトウェア部品の機能などを表していると考えられる．

まずソフトウェア部品のソースコードを解析し，クラスごとに以下の要素を抽出する．

(1) ドキュメントコメント
(2) 通常のコメント
(3) クラス宣言
(4) メソッド宣言
(5) フィールド宣言
(6) ローカル変数宣言
(7) メソッド呼出

Java では 1 つのファイルに複数のクラス（ソフトウェア部品）を記述することができるため，各要素がどのクラスに属するのかを決めなければならない．メソッドやフィールドなどはそれらを直接包含するクラスに属することは明白である．ドキュメントコメントは直後の要素を説明すると定義されているため，直後のクラス，または直後のメソッド等が属するクラスに

```
import java.awt.event.*;

class SelectAllAction implements
                    ActionListener {
  public void actionPerformed(final
       java.awt.event.ActionEvent e) {
    JEditCommanderTable table =
        JEditCommanderPlugin.leftTable;
    table.selectAll();
  }
}
```

図 2 ソフトウェア部品のソースコードの例 （下線部が抽出される要素）

表 1 図 2 から抽出した特徴

特徴	出現頻度
select	1
all	2
action	3
performed	1
table	1

属する．しかし通常のコメントにはそのような明確な基準がない．本手法では通常のコメントはソースコード上でそのコメントを直接包含するクラスに属するとした．また，無名クラスに属する要素は，その無名クラスの外部クラスに属するものとして扱う．

次に抽出した各要素を単語に分割し，各単語を特徴とする．コメント類（上記 1〜2）は，含まれるコメント文字列をスペース区切りで語に分割し，数字や記号のみからなる語や普遍的すぎる単語[*1] を除去する．その他のノード（上記 3〜7）は，宣言された識別子や呼び出しているメソッド名をキャメルケース[*2]として，小文字に続く大文字の直前で単語に分割する．ただし大文字が連続している部分は頭字語であるので 1 つの単語とする．例えばメソッド名 `getUIClassID` は `get`，`UI`，`Class`，`ID` の 4 つの単語に分割する．なお，各特徴は分割後に全て小文字に変換しておく．

図 2 に示す部品のソースコードから特徴を抽出した例を**表 1** に示す．

3.2 事前処理 2: 索引の作成

抽出した特徴を元に，検索に用いる索引を作成する．

まず各ソフトウェア部品の特徴を基に，ソフトウェア部品—特徴の共起行列を作成する．この行列は列にソフトウェア部品，行に特徴をとる．行列の ij 要素は j 番目のソフトウェア部品における i 番目の特徴の出現頻度である．また，この行列の列ベクトルを部品ベクトルと呼ぶ．この行列を入力として LSI を行う．LSI の詳細については文献 9) を参照されたい．

3.3 本処理 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，特定のタイミング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，特徴が変化するタイミングでのみ検索を行いたい．そこで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
- 文の変更後に別の文へカーソルが移動した
- コメントの終了を表す*/が入力された
- コメントの変更後にコメント外へカーソルが移動した

これらをソースコードの編集がある程度完了したタイミングと考え，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次の検索クエリの生成を行う．

3.4 本処理 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．重みはその検索においてその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．これによって現在編集中の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 事前処理 1 と同様に解析することによって得る．ただしこの時，各特徴にはカーソル位置からの距離に応じた重みをつける．なお，同じ特徴が複数回現れた場合，重みはその合計値を使う．

3.5 本処理 3: ソフトウェア部品の検索

3.2 事前処理 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ソフトウェア部品を検索する．

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ものである．擬似部品ベクトルと索引に含まれる部品ベクトルのコサイン尺度を類似度とし，類似度の高い順に数個を検索結果とする．

3.6 本処理 4: ソフトウェア部品の推薦

検索されたソフトウェア部品の一覧を開発者に提示する．また，開発者が必要に応じてソフトウェア部品の詳細を調べ，再利用を行うかどうかを判断できるようにする．

---

*1 以下の URL から入手できる不要語リストを用いた．
`ftp://ftp.cs.cornell.edu/pub/smart/english.stop`

*2 複数の単語からなる名前を，最初の単語は全て小文字，以降の単語は先頭だけ大文字にする命名規則．

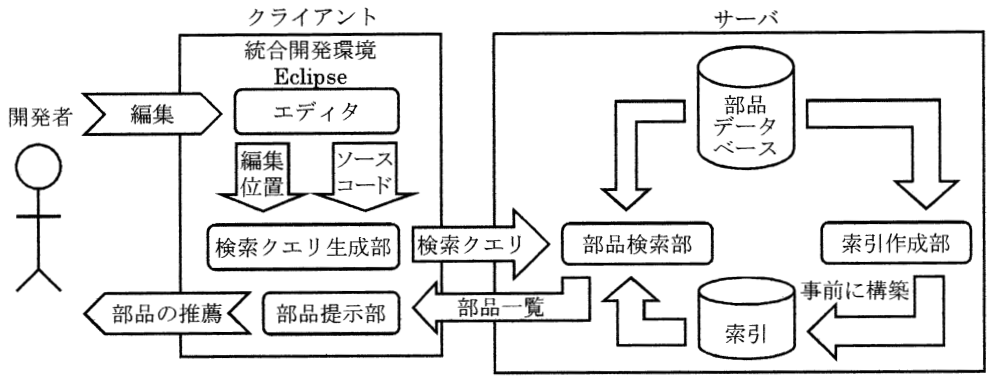

図 3 A-SCORE の構成とデータの流れ

4. ソフトウェア部品自動推薦システム A-SCORE

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 (Automatic Software Component Recommendation Environment) を作成した．本節では A-SCORE のシステム構成と実装の詳細，各部の提案手法の流れとの対応について説明する．

システムの構成と検索処理におけるデータの流れを**図 3** に示す．システムはクライアントとサーバから成っており，ソフトウェア部品はサーバの部品データベースで一元的に管理できるようになっている．クライアントは統合開発環境 Eclipse[10] をプラグインによって拡張したものである．ソフトウェア部品のソースコードの構文解析には Eclipse JDT の AST パーサを利用している．LSI に必要な行列演算には SVDPACKC[11] を Java に移植したものと JLAPACK[12] を使用した．サーバはウェブサービスフレームワークである Apache Axis2[13] を用いてウェブサービスとして実装した．

図 3 の各部について説明する．

部品データベース 蓄積されたソフトウェア部品のソースコードを格納している．

索引 部品データベースに含まれるソースコードを解析して作成した，LSI による検索に用いる索引．

索引作成部 提案手法の事前処理を実装している．部品データベースを解析して索引を作成する．

検索クエリ生成部 本処理 1〜2 を実装している．開発者によるソースコードの編集を監視して編集の区切りを検出する．編集中のソースコードから特徴を抽出して検索クエリを生成する．検索クエリをサーバの部品検索部に送信する．

部品検索部 本処理 3 を実装している．索引を用いて検索クエリに合致する部品を検索する．検索結果のソフトウェア部品を部品データベースから取得する．ソフトウェア部品一覧を部品提示部に送信する．

部品提示部 本処理 4 を実装している．得られたソフトウェア部品一覧を開発者に推薦する．また，ソフトウェア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として，ソフトウェア部品のソースコードを Eclipse 上で閲覧したり，そのソフトウェア部品をプロジェクトにインポートすることができるようになっている．ソフトウェア部品検索システム SPARS-J が持つ該当ソフトウェア部品の詳細情報ページにアクセスする機能も備えている．

A-SCORE の動作画面例を**図 4** に示す．ソースコードは Eclipse 標準のエディタで編集を行う．自動推薦機能を有効にしておけば，エディタでの編集に応じて自動的に画面下部のリストにソフトウェア部品が推薦される．推薦されたソフトウェア部品をダブルクリックすると，そのソフトウェア部品のソースコードが Eclipse のエディタ上で開かれる．また，詳細情報表示のボタンを押すと，SPARS-J が持つそのソフトウェア部品の詳細情報ページが開かれる．開発者はこれらを見てその部品を再利用するかどうかを判断することができる．再利用することに決まったら，インポートボタンを押すとそのソフトウェア部品をサーバからダウンロードして現在のプロジェクトに追加することができる．

5. 適 用 例

本システムの適用例として，Java の標準ライブラリに含まれている，ファイルパスを表す `File` クラスのオブジェクトからそのファイルの拡張子を取得する例を紹介する．`File` クラスにはファイル名を得るメソッドや親ディレクトリ名を得るメソッドは存在する

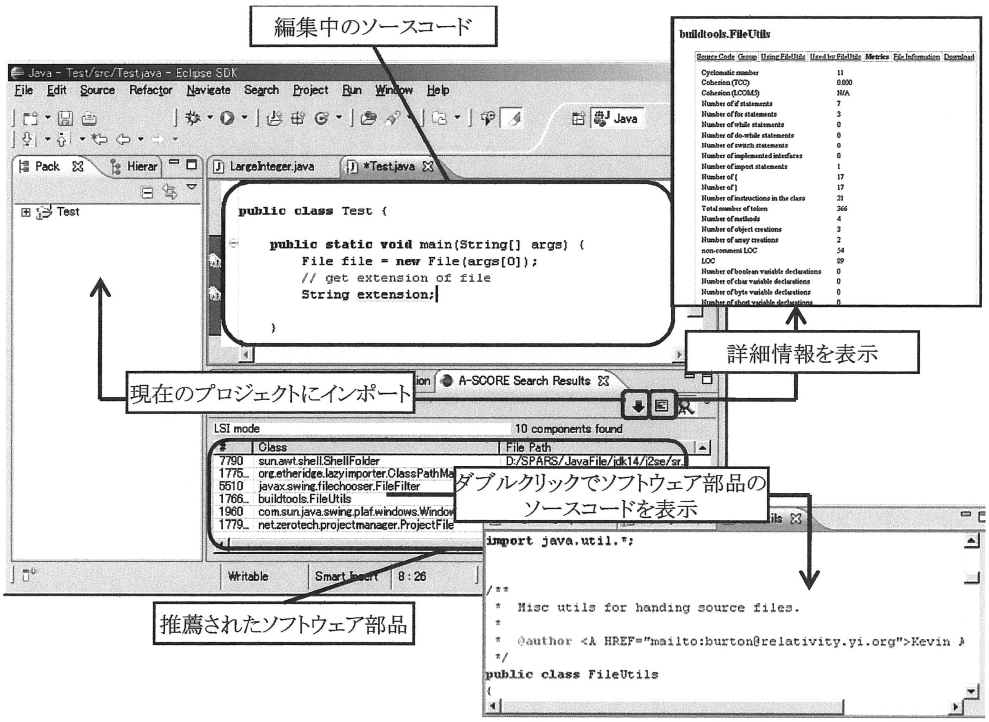

図 4 A-SCORE の画面

が，拡張子を得るメソッドは存在しないため，そのような機能がほしい場合は自分で作成するか既存のソフトウェア部品を再利用しなければならない．

そこで自動推薦を有効にして**図 5** に示すソースコードを入力した．このソースコードでは，`File` クラスの変数 `file` があるとして，`String` 型の変数 `extension` に拡張子を取得しようとしている．入力を行うと，カーソル位置のセミコロンを入力した時点で自動推薦が行われる．推薦されたソフトウェア部品の名前を見て，8 番目にあった `buildtools.FileUtils` クラスを選んだ．このソフトウェア部品はオープンソースソフトウェア jEdit のものであった．一覧で `buildtools.FileUtils` が表示されている行をダブルクリックすると**図 6** のソースコードが表示される．このソフトウェア部品には `String` 型の引数から拡張子を取得するメソッドが含まれており，これが再利用できそうだと判断した．この場合，目的は `File` 型のオブジェクトから拡張子を取得することなので，このソフトウェア部品はそのままでは再利用することはできない．しかしこのメソッドを `File` 型の引数を取るように変更するのは容易なので，そのように変更を加えて再利用を行えばよい．インポートボタンをクリックしてこのソフトウェア部品をプロジェクトにインポートし，変更を加えて再利用を行った．

```
import java.io.File;

public class Test {
  public static void main(String[] args)
  {
    File file = new File(args[0]);
    // get extension of file
    String extension;#
  }
}
```

図 5 編集中のソースコード（#はカーソル位置を示す）

CodeBroker ではこの場合，入力したソースコードにはドキュメントコメントの無い `main` メソッドだけしかないため自動検索が行えない．また，もし `getExtension` というメソッドを入力して自動検索を行おうとしても，入力が一方は `String` 型，もう一方と `File` 型と異なるためこのソフトウェア部品を見つけることができない．

```
/**
 *  Misc utils for handing source files.
 */
public class FileUtils {
  ……省略……
  public static String getExtension(
                          String file) {
    int begin = file.lastIndexOf(".");
    if (begin < 0) {
      return null;
    } else {
      return file.substring(begin + 1,
                          file.length());
    }
  }
  ……省略……
}
```

図 6 図 5 に対して推薦された部品の例

6. 関連研究

ソフトウェア部品の自動検索を行った研究としては，Ye らがソフトウェア部品自動検索システム CodeBroker を開発している[2]．CodeBroker は，Java で書かれたメソッドをソフトウェア部品として，メソッドに付随するドキュメントコメントとシグネチャを元に検索を行う．本手法は Java で書かれたクラスをソフトウェア部品として，クラス内に含まれるドキュメントコメント，通常のコメント，識別子を元に検索を行う．

キーワード検索を用いない検索やコード片の再利用を行った研究としては，Holmes らがコード例検索システム Strathcona を開発している[14]．Strathcona は，開発者の要求を表す小さなコード片を入力として，その要求を満たす実行可能なコード例を構造的な類似度を元に検索する．

川口らは，ソースコード中の識別子をそのソフトウェアの特徴と考え，LSA によって類似ソフトウェアを検索することでソフトウェアのクラスタリングを行っている[15]．本手法では識別子とコメントの両方をそのソフトウェア部品の特徴と考え，類似部品の検索を行っている．

キーワード検索による検索結果を改善する研究として，SPARS-J ではソフトウェア部品の重要度を表す値として ComponentRank とその計算方法を提案し，ComponentRank をキーワード検索と組み合わせることで検索結果の順位付けを改善することに成功している[5),6),16]．本手法による推薦も，ComponentRank を LSI による検索と組み合わせることで改善できると期待できる．

7. ま と め

本稿では，ソフトウェア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ソフトウェア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，本手法を実装したソフトウェア部品の自動推薦システム A-SCORE を作成した．

今後の課題として，まず大規模な部品集合への対応が挙げられる．次に手法の有用性を評価するための大規模な部品集合を用いた評価実験が挙げられる．具体的には，学生に本システムを使用してもらい，作業時間の短縮やアンケート結果で評価を行うことを考えている．その他には，特徴の抽出の仕方による影響や，どの種類の特徴が検索に一番役立っているのかを調査することも考えている．

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．

参 考 文 献

1) Krueger, C.: Software reuse, *ACM Computing Surveys (CSUR)*, Vol.24, No.2, pp.131–183 (1992).
2) Ye, Y. and Fischer, G.: Reuse-Conducive Development Environments, *Automated Software Engineering*, Vol.12, No.2, pp.199–235 (2005).
3) : Koders, `http://www.koders.com/`.
4) : Google Code Search, `http://www.google.com/codesearch`.
5) Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking Significance of Software Components Based on Use Relations, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, pp. 213–225 (2005).
6) : SPARS-J, `http://demo.spars.info/`.
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 TRANSACTIONS ON SOFTWARE ENGINEERING*, pp.495–510 (2005).
9) Deerwester, S., Dumais, S., Furnas, G., Landauer, T. and Harshman, R.: Indexing by latent semantic analysis, *J.Am.Soc.Inf.Sci*, Vol.41, No.6, pp.391–407 (1990).

10) : Eclipse, http://www.eclipse.org/.
11) Berry, M. W., Do, T., O'Brien, G. W., Krishna, V. and Varadhan, S.: SVDPACKC (Version 1.0) User's Guide, Technical Report CS-93-194, University of Tennessee, Knoxville, TN (1993).
12) : JLAPACK, http://www.netlib.org/java/f2j/.
13) : Apache Axis2, http://ws.apache.org/axis2/.
14) Holmes, R.: Approximate Structural Context Matching: An Approach to Recommend Relevant Examples, *IEEE Transactions on Software Engineering*, Vol. 32, No. 12, pp. 952–970 (2006).
15) 川口真司, 松下誠, 井上克郎：潜在的意味解析法 LSA を利用したソフトウェア分類システムの試作, 情報処理学会研究報告, pp.55–62 (2003).
16) 横森励士, 梅森文彰, 西秀雄, 山本哲男, 松下誠, 楠本真二, 井上克郎：Java ソフトウェア部品検索システム SPARS-J, 電子情報通信学会論文誌 DI, Vol.87, pp.1060–1068 (2004).