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

信学技報
TECHNICAL REPORT OF IEICE.

ソースコードコーパスを利用したシームレスな再利用支援

山本　哲男† 　吉田　則裕†† 　肥後　芳樹†††

† 立命館大学 情報理工学部　〒 525–8577 滋賀県草津市野路東 1–1–1
†† 奈良先端科学技術大学院大学 情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
††† 大阪大学 大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †tetsuo@cs.ritsumei.ac.jp, ††yoshida@is.naist.jp, †††higo@ist.osaka-u.ac.jp

あらまし　効率的にソフトウェアを開発するための手段として再利用が注目されている．しかし，再利用に必要な作業（コピーアンドペーストを行なう際にコピー元のファイルを探して開く，キーワードを用いてソースコードを検索する際にキーワードを考える等）自体にコストがかかってしまう．本稿では，そのような再利用に伴うコストを極力排除した，シームレスな再利用支援手法を提案する．提案手法では，再利用を行なう際にユーザは再利用のトリガを入力するだけで，現在開発しているコンテキストで再利用可能な候補が提示される．また，本稿では，提案手法を Eclipse プラグインとして実装して行なった適用実験についても述べる．

キーワード　ソースコード再利用，統合開発環境

Seamless Reuse Support using Source Code Corpus

Tetsuo YAMAMOTO†, Norihiro YOSHIDA††, and Yoshiki HIGO†††

† College of Information Science and Engeneering, Ritsumeikan University　Nojihigashi 1–1–1, Kusatshu-shi, Shiga, 525–8577 Japan
†† Graduate School of Information Science, Nara Institute of Science and Technology　Takayama 8916–5, Ikoma-shi, Nara 630–0192 Japan
††† Graduate School of Information Science and Technology, Osaka University　Yamadaoka 1–5, Suita-shi, Osaka 565–0871 Japan
E-mail: †tetsuo@cs.ritsumei.ac.jp, ††yoshida@is.naist.jp, †††higo@ist.osaka-u.ac.jp

Abstract　Recently, software reuse is attracting much attention as a key technique for efficient software development. However, reuse itself requires human resources: for example, searching and opening source files including code snippets that users want to reuse, or considering keywords matching code snippets for reuse. The present paper proposes a novel method that hardly requires such reuse cost. In the proposed method, all users have to do for getting reusable code snippets is just inputting a trigger key on their development environments. Also, this paper describes some applications on open source software with a prototype tool working on Eclipse.

Key words　Source code reuse, IDE

1. はじめに

効率的なソフトウェア開発を実現するための方法として再利用が注目されており，これまでにさまざまな再利用支援手法やツールが開発されている．開発者が最も頻繁に（手軽に）行なう再利用は，既存コードのコピーアンドペーストである．コピーアンドペーストを行なうことによって実現したい機能を瞬時に実装することができる．しかし，開発者はコピー元のコードを探すための作業が必要となるなど，効率的に再利用が行なれているとはいえない．

このような問題を解決するために，キーワードによるソースコード片検索システムが開発されている [1]〜[4]．ユーザは実装したい機能を表すキーワードをシステムに入力すると，システムに登録されているソースコードのうち，入力されたキーワードと関連しているソースコードが表示される．このようなシステムを用いることにより，ユーザは再利用元のソースコードを探す手間が無くなる．また，コピーアンドペーストでは再利用元のソースコードは自分自身が過去に書いたソースコードであることが多いが，このようなシステムを用いることによって不特定多数のソースコードを再利用対象とすることができる．

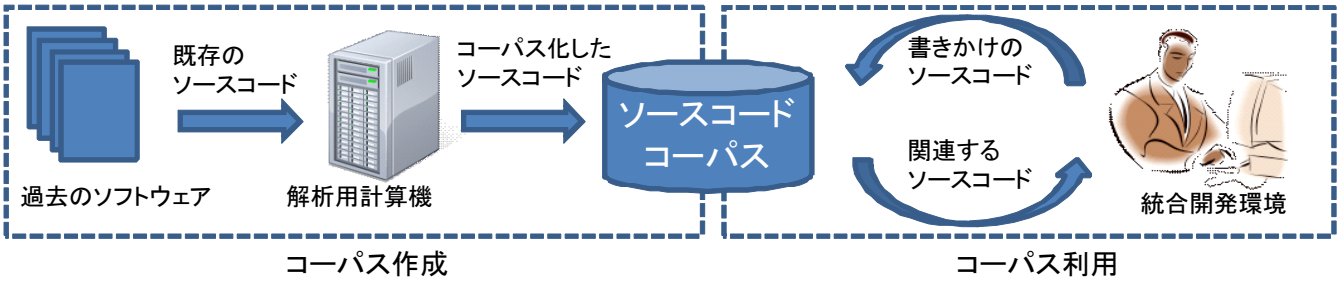

図 1　提案手法概要

また，Ye らは CodeBroker というシステムを開発している [9]．CodeBroker は Emacs エディタのプラグインとして実装されており，利用者が Javadoc コメントを記述すると，それと関連するソースコードを提示する．キーワード検索システムとは違い，ユーザは開発作業を中断すること無く，必要に応じて再利用を行なうことができる．しかし，ソースコード中の単語を基に関連するソースコードを特定するため，コメントの質に検索結果が大きく影響を受けてしまい，再利用可能なソースコードが存在しているのにそれを正しく提示できない場合がある．

さらに，再利用するソースコードの質も重要な課題である．近年，Google サジェストや日本語入力のように，世の中の人が作成・利用した情報を集めて解析し，推薦する仕組みが多く存在する．これらの仕組みでは，人々の集合知を利用することで，ユーザの求めているものを推薦することが可能な点である．この仕組みは，利用者と同じドメインを対象にして解析することで，精度が向上する可能性がある．ソースコードも同様な考えに基づいて，世の中の人に多く利用されているソースコードが再利用するのによいソースコードといえる．ソースコード開発者と同じドメインであるオープンソースの開発者が作成したオープンソースへ適用することで，有能な開発者が作成した「知」を収集でき，利用できる．

本稿では，上述した既存研究の問題点を解決した実用的な再利用支援手法を提案する．具体的には，本研究の特徴は以下のとおりである．

- 再利用元コードを検索するキーとなるのは，現在利用者が書きかけのコードである．ユーザは再利用のために検索を行なうキーワードを考える必要が無い．再利用の際に用いるキーがユーザの書きかけのコードであるため，再利用を行ないたいときは，非常に単純な操作で関連するコードを閲覧できる．このため，キーワード検索システムを利用する場合に比べてコーディングを中断する時間が短くなる．
- 関連するコードの検索は，コードのロジックに基づいて行なわれるため，実行に影響を与えないコメントや変数名などに影響されることなく，再利用を行なうことができる．
- 提案システムを Eclipse プラグインとして実装した．Eclipse は現在広く使われている無償の統合開発環境であり，Eclipse と連携することにより，提案手法を多くの人に利用してもらうことができる．

以降本稿は，2. で提案手法を紹介し，3. では実装した Eclipse プラグインを用いて行なった実験について述べる．4. では，関連研究について触れ，最後に 5. で本稿をまとめる．

2. 提案手法

本節では，ソースコードの再利用を支援するための手法について説明する．全体の概要を図 1 に示すように，本手法は二つの仕組みから構成される．一つは再利用の元となるソースコードを解析し，ソースコードコーパスと呼ぶデータベースに保存する処理（コーパス作成）である．もう一つは，ソースコードコーパスに保存されたソースコードを統合開発環境に提示するための処理（コーパス利用）である．コーパス作成とコーパス利用はそれぞれ別の処理として考えることができ，独立して実行できる．

コーパス作成処理では，過去に作成したソフトウェアから，ソースコードの用例を作成する．本手法の用例はソースコード中のブロック内のソースコードを対象にする．例えば，C 言語の関数，Java 言語でのメソッドなどが該当する．ブロック内の先頭に記述されたソースコード片が出現した後に出現するソースコードとして，どのようなソースコードの出現確率が高いか計算し，用例としてデータベースへ保存しておく．このデータベースを本研究ではソースコードコーパスと呼ぶ（以降，単にコーパスと呼ぶ）．

コーパス利用処理は，実際に開発者が開発する際に利用する部分であり，開発者が統合開発環境からソースコードの再利用を要求するたびに利用される．統合開発環境で書いている途中のソースコードをコーパスに問い合わせ，書きかけのソースコードのあとに続くソースコードとしてどのようなものが過去のソフトウェアの中で確率が高かったかを提示する．

以降，提案手法の詳細とその実装について記述する．

2.1 コーパス作成手法

コーパス作成手法は，ソースコード解析手法と頻度計算手法に分けることができる．ソースコード解析手法ではソースコードを解析し，コーパスへ保存できる形式に変換する方法であり，実際の登録作業は行なわない．頻度計算手法では，ソースコード解析結果を検索しやすいように並び替え，実際にコーパスへ保存するための方法である．

ソースコード解析手法は，各ソースコードを以下の手順で解析する．手法の流れを図 2 に示す．

（1） ソースコードを字句解析および構文解析し，ブロックを抽出する．

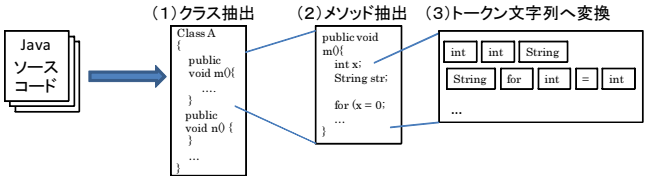

図 2　ソースコード解析手法概要

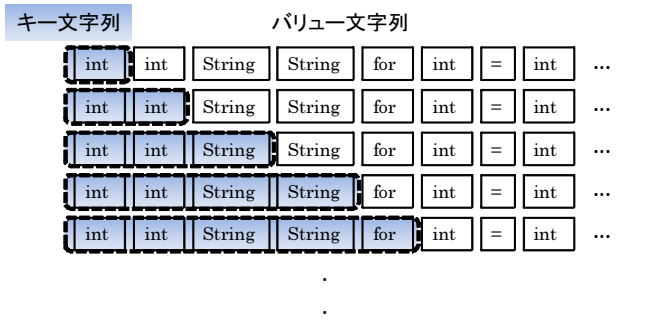

図 3　キー文字列・バリュー文字列分割手法

（2）　抽出した各ブロックに対してトークンの分割処理をする．

トークンの分割処理の手順は以下の通りである．

（1）　ブロック開始から終わりまでに含まれるすべてのトークンに対して，それを表す文字列に変換する．通常は，ソースコード中に現れる文字列をそのまま利用する．ただし，言語に依存する処理として，変換ルールを別途定義して別の文字列へ変換することを可能にする．例えば，変数名を表すトークンをそのまま変数名として用いるのではなく，その変数の型を表す文字列に変換する，などの処理である．この変換後の文字列をトークン文字列とよぶ．

（2）　トークン文字列をソースコードの出現順に並べる．

（3）（2）で並べたトークン文字列の並びを二分割する．ニ分割する箇所はすべてのトークン文字列とトークン文字列の間とする．ただし，ニ分割した文字列は一個以上のトークン文字列を含むものとする．n 個のトークン文字列の並びがあると，n-1 個の分割箇所をもつ．分割した前半のトークン文字列の並びをキー文字列とよび，後半のトークン文字列の並びをバリュー文字列とよぶ．

（4）　キー文字列とバリュー文字列のペアを生成する．n 個のトークン文字列の並びが存在する場合は，n-1 個のペアが生成される．図 3 にその処理の流れを示す．

次に，生成したペアをコーパスに登録する必要がある．同じキー文字列に対して，複数のバリュー文字列が存在する．さらに，キー文字列とバリュー文字列が両方とも同じペアがすでにコーパスに保存されている可能性もある．そのため，キー文字列とバリュー文字列と頻度という三つ組みにしてコーパスへ登録する．頻度には，これまでに解析したソースコードの中で同じキー文字列とバリュー文字列の数を保存する．

そこで，キー文字列とバリュー文字列のペアを登録する際には，以下の頻度計算手法に従い登録する．

（1）　すでにデータベースに，そのペアが存在しているか確認する．

（2）　なければ，頻度を 1 にした三つ組みを生成し，（4）へ

（3）　すでに存在していれば，コーパス中の三つ組み頻度の値を 1 増やす．

（4）　頻度の多い順に取得できるように，同じキー文字列の三つ組みをまとめておき，頻度順にソートして保存しておく．

コーパス検索時に，短時間で検索を終了させるたに，登録時には頻度の値を変更するたびに，並び替えをしておく．

2.2　コーパス利用手法

統合開発環境で書きかけのソースコード片を入力として，そのソースコード片に対応した残りソースコード片を頻度の多い順に出力する方法について説明する．

統合開発環境中で入力されたソースコードを監視し，ブロック中の書きかけのソースコードを常にキー文字列に変換する．変換手法はソースコード解析手法と同様に，字句解析・構文解析・トークン変換ルールを適用することで取得する．

取得したキー文字列をコーパスに問い合わせ，キー文字列を含む三つ組を取得する．複数存在する場合は，頻度順にソートされたものが取得できる．取得した三つ組みからバリュー文字列を取得し，ソースコードに逆変換する．逆変換したソースコードを頻度順に開発者に提示する．

2.3　実　　装

本手法の対象ソースコードを Java 言語としてソースコード解析手法，頻度計算手法，コーパス利用手法を実装した．実装にあたり，以下の言語依存ルールを採用した．

- ソースコード解析手法で抽出するブロックとしてはクラスのメソッドとした．
- トークンを変換するルールを表 1 の通りにした．型名および変数名を表すトークンをクラス名に変換する．また，コメント，およびメソッド呼び出し文で利用する ．はトークン文字列に含めない．さらに，ブロックを表す { と } を省略している箇所においては自動的に挿入する．

実際の Java のソースコードを例としてソースコード解析手法を説明する．図 4(a) の Java ソースコードを取得したととする．このメソッド `printFile` 内のトークンをソースコード解析手法に従って変換したトークン文字列の並びを図 4(b) に示す．トークン文字列は紙面の都合上改行で区切っているが，一続きの並びである．

型名はその型のクラス名に変換する．型名が完全限定名で記述されていてもクラス名のみに変換する．たとえば，

表 1　トークンの変換ルール

トークンの種類	変換処理
型名	クラス名に変換
変数名	クラス名に変換
リテラル	クラス名に変換
コメント	トークンを生成しない
“.”（メソッド呼び出し式で利用）	トークンを生成しない
上記以外のトークン	ソースコードに記述された文字列

```
public class Sample1 {
   private int x;
   private int y;

   /**
    * ファイルの内容を標準出力へ書き出す
    */
   public void printFile(String filename) {
      File file = new File(filename);
      try {
         FileInputStream is = new FileInputStream(file);
         InputStreamReader sr = new InputStreamReader(is);
         BufferedReader br = new BufferedReader(sr);
         String str;
         // ファイルの終わりまで読み取る
         while ((msg = br.readLine()) != null)
            System.out.println(str);
         br.close();
         sr.close();
         is.close();
      } catch(Exception e) {
         e.printStackTrace();
      }
   }
}
```

(a) Java ソースコード

```
File File = new File ( String ) ;
try {
FileInputStream FileInputStream = new
FileInputStream ( File ) ;
InputStreamReader InputStreamReader = new
InputStreamReader ( FileInputStream ) ;
BufferdReader BufferdReader = new
BufferdReader ( InputStreamReader ) ;
String String ;
while ( ( String = BufferdReader
readLine ( ) ) != null ) {
System out println ( String ) ;
}
BufferdReader close ( ) ;
InputStreamReader close ( ) ;
FileInputStream close ( ) ;
} catch ( Exception Exception ) {
Exception printStackTrace ( ) ;
}
```

(b) トークン文字列

図 4　ソースコードからトークン文字列への変換

`java.io.File` とソースコード中に記述されていても `File` と変換する．変数名に関しても同様に，その変数名が宣言された型のクラス名に変換する．それぞれ，完全限定名を特定し変換することも考えられるが，クラス名が特定できれば十分であると判断し，解析コストとのトレードオフを考え，クラス名だけにした．また，`while` 文の後の `System.out.println` の前後に自動的に {} を補う．{} のあり・なしを，ありに統一させることで，キー文字列の一致精度を上げる．

本手法を実装したツールについて説明する．シームレスに利用するために，統合開発環境から利用する際の待ち時間を減らす工夫や使い勝手を高める工夫をしている．

統合開発環境には Eclipse を用いており，ソースコード解析ツールに Masu(注1) を用いている．コーパスには BerkleyDB(注2) を用いている．コーパス登録時にはトークン文字列をそのまま登録するのではなく，MD5 などのハッシュ関数を利用して，文字列をハッシュ値に変換して登録する．Eclipse からの検索時には，書きかけのソースコード片を解析し，トークン文字列に変換後，さらにハッシュ関数によりハッシュ値にした値をキーにしてコーパスを検索している．

コーパス利用手法を Eclipse プラグインとして実装した．開発者が Eclipse のエディタ上でソースコードを記述中に，開発者のトリガーによって書きかけのソースコードをキーとしてコーパスへ候補を問い合わせる．コーパスに保存されているレコードからキーに完全一致したレコードを検索するだけなので，高速に処理が終わる．

検索結果の Eclipse のエディタ上へ表示し，開発者にどの候補がふさわしいか選択してもらい，確定したソースコードをエディタ上へ貼り付ける．

3. 適 用 実 験

提案手法を評価するために実験を行なった．

3.1 データベース構築のパフォーマンス評価

本節では，データベースに構築に要した時間や構築したデータベースのサイズについて調査した．この調査では，規模の異なる三つのソフトウェア（群）に対してデータベースを構築した．各ソフトウェアの規模を表 2 に示す．なお，Apache Project とは，http://www.apache.org/で公開されているソフトウェア群を表す．

表 3 は，各ソフトウェアについて，データベース構築に要し

表 2　対象ソフトウェア

ソフトウェア	ファイル数	行数
Ant 1.8.1	829	212,401
JDK 1.6.0	7,154	2,071,178
Apache Project	51,777	9,669,445

表 3　データベース構築時間とサイズ

ソフトウェア	構築時間	サイズ
Ant 1.8.1	3 分 7 秒	493 MB
JDK 1.6.0	53 分 41 秒	4.13 GB
Apache Project	12 時間 50 分 21 秒	28.2 GB

(注1)：http://sourceforge.net/project/masu/

(注2)：www.oracle.com/technetwork/database/berkleydb/overview/index.html

```
/**
 * find a file in a directory in case unsensitive way
 * @param parentPath        where we are
 * @param soughtPathElement what is being sought
 * @return                  the first file found or null if not found
 */
private String findPathElementCaseUnsensitive(String parentPath,
  String soughtPathElement) {
  // we are already in the right path, so the second parameter
  // is false
  FTPFile[] theFiles = listFiles(parentPath, false);
  if (theFiles == null) {
    return null;
  }
  for (int icounter = 0; icounter < theFiles.length; icounter++){
    if (theFiles[icounter] != null &&
      theFiles[icounter].getName().equalsIgnoreCase(soughtPathElement)){
      return theFiles[icounter].getName();
    }
  }
  return null;
}
```

図 5　FTP を介してファイル検索を行うソースコード (Ant 1.8.1)

た時間と構築したデータベースのサイズを表している．構築時間はソフトウェアの規模に比例して長くなっていることがわかる．特に，Apache Project では，約 13 時間と非常に長い時間を必要とした．しかし，全てのソースコードの登録処理は，利用開始時の一度だけ行なえばよく，利用開始後は更新されたファイルについてのみ，データベースも更新をすればよい．このため，利用に関して特に問題とはならないと著者らは考えている．

また，データベースサイズもソフトウェアのサイズに比例して大きくなっていることがわかる．しかし Apache Project からデータベースを構築した場合でも，たかだか 28GB であり，この程度の大きさであれば，現在の PC であれば問題なく収容することができる．

3.2 適　用　例

提案手法を Ant 1.8.1 に適用した例について説明する．Ant の FTP クラスと FTPTaskMirrorImpl クラスには，FTP を介してファイル検索を行うメソッドが含まれている（図 5）．これらメソッドは，コメントも含め完全に一致していた．

以下の手順で，提案手法の適用を行った．

手順 A　2. で述べたソースコード解析と頻度計算を Ant 1.8.1 のソースコードへ適用

手順 B　Eclipse の Java 開発環境のソースコードエディタにおいて，図 5 で挙げたメソッド本体の冒頭（FTPFile[] theFiles = listFiles(parentPath, false);）を入力

手順 C　上述のソースコードエディタ上からコンテンツアシスト機能を呼び出すことで，補完候補を要求

手順 C までを行うと，提案する Eclipse プラグインは入力したメソッドの冒頭を取得し，それをキー文字列としてデータベースに渡す．次にデータベースは渡されたキー文字列に対応するバリュー文字列を返し，最後に提案する Eclipse プラグインはそのバリュー文字列を補完候補としてユーザに提示する．

上述の手順で，メソッド本体の冒頭（FTPFile[] theFiles = listFiles(parentPath, false);）を入力し補完候補を要求したところ，メソッド本体の全体が補完候補として提示された．

この結果から，図 5 に挙げたメソッドを知らない開発者からみて，提案手法が行う補完候補の推薦は有効であると考えられる．

4. 関 連 研 究

本研究と同様にメソッドの補完を行う手法として，Hill らの手法 [6] が挙げられる．この手法は，メソッドを行数や複雑度等を要素とする特徴ベクトルで表現し，入力されたメソッドの近傍に存在し，かつ入力されたメソッドより行数の大きいメソッドを補完候補として提示する．Hill らの手法は，入力されたメソッドと類似し，かつ少し行数の大きいメソッドを補完候補として提示することが目的である．そのため，提案手法が目的としているメソッドの冒頭を入力としたメソッド全体の補完には適していない．提案手法は，メソッドの冒頭のコード片をキーとし，冒頭以後の部分をコード片を値とするデータベースを構築することによって，メソッドの冒頭を入力としたメソッド全体の補完を実現している．また，提案手法は入力されたコード片のトークン列をクエリ（検索質問）として検索を行うため，入力されたコード片のロジックの補完に適している．Hill らの手法は，入力されたメソッドを特徴ベクトルに変換し，その特徴ベクトルをクエリとして検索を行う．そのため，クエリから構文の順序に関する情報が欠落しており，ロジックの補完には適していない．

本研究と同様に，シームレスなソースコード再利用を支援する手法として，CodeBroker [9] や A-SCORE [10] が挙げられる．これらは，ユーザの入力から自動的に抽出したキーワードをクエリとしてソースコードの検索を行う．本研究では，入力されたコード片の補完することを目的としているため，入力されたコード片をトークン列に変換し，そのトークン列をクエリとして検索を行う．

この他にも，ソースコードを収集し，その中からユーザからの要求に応じたものを提示するシステムの研究は数多く行われている [2] [4] [5] [8]．公開されているものとしては，Google ソースコード検索 [2] や SPARS-J [4]，Koders [3] などのソースコード検索エンジンが挙げられる．これらは，ユーザが考案したキーワードをクエリとして検索を行うシステムであり，本研究のようにシームレスに補完候補を検索するためのシステムとは異なる．また，API の使用方法の理解支援を目的として，API の使用を含むコード片を検索する手法 [7] が提案されている．この手法は，クエリとして与えられた API の使用例を示すコード片を提示することを目的としており，提案手法とはクエリの内容や提示する内容が異なる．

5. おわりに

本稿では，開発者が開発作業を中断すること無く，必要に応じてソースコードの再利用を行なうことができる手法を提案した．再利用元コードを検索するキーとなるのは，現在開発者が書きかけのソースコードである．さらに，提案手法さらに，提案システムを Eclipse プラグインとして実装したツールを用いて実現可能性を考察した．その結果，現在の PC であれば問題なく利用可能であることを示した．

今後は，トークン文字列の変換ルールやキー文字列とバリュー文字列の区切り方に関する実験，キー文字列の完全一致以外の方法について考える．さらに，コーパス利用時に表示するバリュー文字列のランキングは出現回数順だが，そのほかの要素を加えるべきかについて考える予定である．

謝辞　本研究は一部，日本学術振興会科学研究費補助金 研究活動スタート支援 (課題番号:22800040) の助成を得た．

文　献

[1] Codase. `http://www.codase.com/`.
[2] Google Code Search. `http://www.google.com/codesearch`.
[3] Koders. `http://www.koders.com/`.
[4] SPARS-J. `http://demo.spars.info/`.
[5] S. Chatterjee, S. Juvekar, and K. Sen. SNIFF: A search engine for java using free-form queries. In *Proc. of FASE 2009*, pp. 385–400, 2009.
[6] R. Hill and J. Rideout. Automatic method completion. In *Proc. of ASE 2004*, pp. 228–235, 2004.
[7] R. Holmes, R. J. Walker, and G. C. Murphy. Approximate structural context matching: An approach to recommend relevant examples. *IEEE Trans. Softw. Eng.*, 32(12):952–970, 2006.
[8] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking Significance of Software Components Based on Use Relations. *IEEE Trans. Softw. Eng.*, 31(3):213–225, 225.
[9] Y. Ye, G. Fischer, and B. Reeves. Integrating Active Information Delivery and Reuse Repository Systems. In *Proc. of SIGSOFT 2000/FSE 8*, pp. 60–68, 2000.
[10] 島田, 市井, 早瀬, 松下, 井上. 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. 情報処理学会論文誌, 50(12):3095–3107, 2009.