情報処理学会論文誌 Vol.53 No.2 644–652 (Feb. 2012)

ソースコードコーパスを利用したシームレスなソースコード再利用手法

山本 哲男[1,a)] 吉田 則裕[2,b)] 肥後 芳樹[3,c)]

受付日 2011年6月14日, 採録日 2011年11月7日

概要：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など）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ともな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ソースコードの候補が提示される．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，コーパス

Seamless Souce Code Reuse Using Source Code Corpus

Tetsuo Yamamoto[1,a)] Norihiro Yoshida[2,b)] Yoshiki Higo[3,c)]

Received: June 14, 2011, Accepted: November 7, 2011

Abstract: 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.

Keywords: source code reuse, IDE, corpus

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わ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], [3], [4], [10]．実装したい機能を表すキーワードを開発者が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，開発者は再利用元のソースコードを探す手間がなくなる．また，コピーアンドペーストにおいて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

1 日本大学
Nihon University, Koriyama, Fukushima 963–8642, Japan
2 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
3 大阪大学
Osaka University, Suita, Osaka 565–0871, Japan
a) tetsuo@cs.ce.nihon-u.ac.jp
b) yoshida@is.naist.jp
c) higo@ist.osaka-u.ac.jp

数のソースコードを再利用対象とすることができる．

また，Ye らは CodeBroker というシステムを開発している [11]．CodeBroker は Emacs エディタのプラグインとして実装されており，開発者が 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，開発者は開発作業を中断することなく，必要に応じて再利用を行うことができる．しかし，ソースコード中の単語をもと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．これらは，人々の集合知を利用することで，開発者の求めているものを推薦する仕組みである．この仕組みは，開発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推薦することで，推薦精度が上がるのではないかと考える．つまり，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適したソースコードではないかと考える．ある開発者と同じドメインのソフトウェアを開発しているオープンソースの開発者が作成したソースコードへ集合知の考えを適用することで，有能な開発者が作成した「知」を収集・利用でき，開発の生産性向上につながる．

本稿では，上述した既存研究の問題点を解決し，大規模なソースコードの中から適切なコード片を推薦する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，現在開発者が書きかけのコードである．開発者は再利用のために検索を行うキーワードを考える必要がない．そのため，非常に単純な操作をするだけで，書きかけのソースコードの後に続くものとして最適なソースコードを取得できる．特別な入力を必要としないため，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．
- ソースコードの検索は，コードの構文に基づいて行われるため，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 章で提案手法を紹介し，3 章では実装した Eclipse プラグインを用いて行った実験について述べる．4 章では，関連研究について触れ，最後に 5 章で本稿をまとめる．

2. 提案手法

本章では，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．図に示すように，本手法は 2 つの仕組みから構成される．1 つは再利用のもととなるソースコードを解析し，ソースコードコーパスとよ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 1 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名前の付けられた一塊のソースコード片を対象にする．C 言語の関数，Java 言語でのメソッドなどが対象となり，本研究では，それらを再利用コード片とよぶ．ソースコードのどの構文を再利用コード片にするかはソースコードに記述されたプログラミング言語の種類によって変えられるものとする．再利用コード片内のソースコードを任意の場所で 2 分割し，前方に記述されたソースコード片と後方に記述されたソースコード片のペアを作成し，データベースへ保存しておく．多くのソフトウェアに対してペアを作成することで，ペアの出現頻度を計算し，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よぶ（以降，単にコーパスとよぶ）．

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後に続くソースコードとしてどの

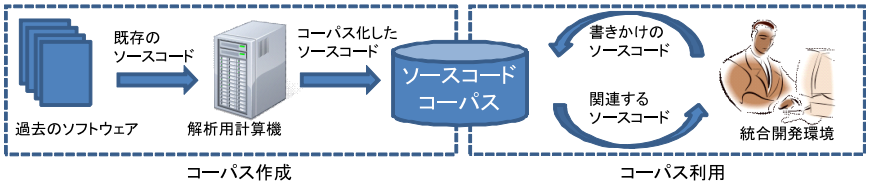

図 1 提案手法概要
Fig. 1 The outline of proposed method.

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べ替え，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，再利用コード片を抽出する．

(2) 抽出した各再利用コード片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とおりである．

(1) 再利用コード片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．たとえば，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，よく用いる定型的な構文を消去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 2 分割する．2 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，2 分割した文字列は 1 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，$n-1$ 個の分割箇所を持つ．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しうる．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にしてコーパスへ登録する．頻度として，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ペアが出現した数を保存する．

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．

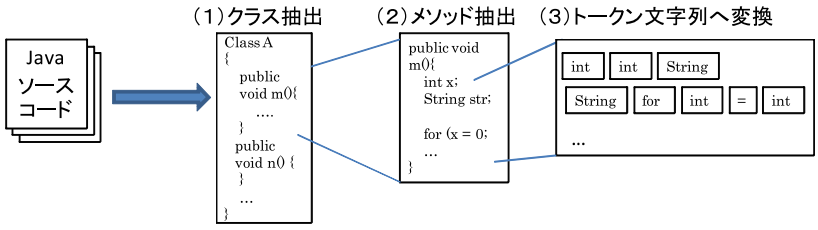

図 2 ソースコード解析手法概要

Fig. 2 The outline of source code analysis.

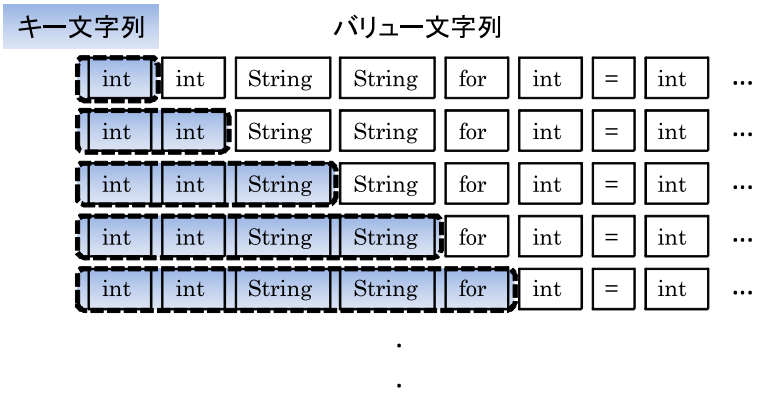

図 3 キー文字列・バリュー文字列分割手法

Fig. 3 Key strings and value strings separation method.

(2) なければ，頻度を 1 にした三つ組を生成し，(4) へ．
(3) すでに存在していれば，コーパス中の三つ組頻度の値を 1 つ増やす．
(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をまとめておき，頻度順に並べ替えをして保存しておく．

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，並べ替えをしておく．

また，キー文字列とバリュー文字列をソースコードに復元するために，ソースコード解析手法の (1) が終了した直後のトークン列をソースコードのトークン列として保存しておく．

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，書きかけのソースコードをつね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・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，頻度順に並べ替えられたものが取得できる．取得した三つ組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．逆変換するためにコーパス作成時に保存したソースコードのトークン列を利用する．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，以下の変換ルールを採用した．オープンソースのソースコードを調査した結果，メソッド冒頭にエラー処理や例外処理の記述があるソースコードが多く存在することが分かった．そのため，検索精度を向上させるために，例外処理やエラー処理を事前に削除することとした．メソッド開始直後に引数の値をチェックするコードが存在すると，キー文字列にも同様のコードを含める必要が生じるためである．さらに，エラー処理のあるソースコードとないソースコードを同様のソースコードと見なすことで，それらのソースコードから作成された三つ組の頻度が増すため，検索結果が向上することが期待できる．

- ソースコード解析手法で抽出する再利用コード片を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** のとおりにした．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．コメントは無視し，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．
- `if (expression == null) {statements}` 文を削除する．ただし，expression は任意の式とし，statements は 1 文以上の任意の文とする．
- `if (expression1) {return expression2;}` 文を削除する．ただし，expression1 と expression2 は任意の式とする．
- `if (expression1) {throw expression2;}` 文を削除する．ただし，expression1 と expression2 は任意の式とする．
- メソッドの先頭から始まる連続する変数宣言文をまとめる．トークン変換ルール適用後，型名と変数名はクラス名に置き換わる．変数初期化部分が存在すると右辺に式が並ぶが，この右辺をすべて削除する．さらに，それらの宣言文の並びをトークンの出現順ではなく，アルファベット順に並べ替える．

表 1 トークンの変換ルール
Table 1 Transformation rule.

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	そのまま

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，`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

```
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) {
        if (filename == null) { return; }
        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
try {
FileInputStream FileInputStream = new
FileInputStream ( File ) ;
InputStreamReader InputStreamReader = new
InputStreamReader ( FileInputStream ) ;
BufferedReader BufferedReader = new
BufferedReader ( 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

Fig. 4 Conversion source code into token strings.

ド解析ツールに MASU [9] *1 を用いている．コーパスには BerkleyDB *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．データベースにキーバリューストアを採用することで，キー文字列によるバリュー文字列の取得時間を短くするする工夫をしている．

*1 http://sourceforge.net/project/masu/

*2 http://www.oracle.com/technetwork/database/berkleydb/overview/index.html

表 2 対象ソフトウェア

Table 2 Target software.

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

コーパス利用時にキー文字列やバリュー文字列からソースコードを復元する必要がある．そのため，コーパス登録時にソースコードのトークン列もデータベースに保存しておく．変換ルールを適用する前のトークン列であるため，Java のメソッド内のすべてのトークン列となる．さらに，バリュー文字列のハッシュ値から実際のソースコードへ復元できるように，トークン列を保存しているデータベースへの参照もあわせて記録しておく．

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のみであるので，高速に処理が終わる．

検索結果を 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．貼り付け処理時にソースコードを復元するために，ハッシュ値に対応づけているトークン列をデータベースから取得し，整形ツールを利用して貼り付ける．頻度が複数の場合，どのソースコードのトークン列を復元するかを，あらかじめ決めておく．本稿の実装では，変換ルールにおいて削除された文が最も多いソースコードを復元することとした．これは，エラー処理や例外処理などが付与されたソースコードを復元することを意図している．

3. 評価

提案手法を評価するために実験を行った．本章では，実用的な速度で実現可能かを検証するためのパフォーマンス評価について記述し，その後，本手法の有効性の評価について述べる．

3.1 データベースのパフォーマンス評価

本節では，データベースに構築や利用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 3 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 データベース構築時間とサイズ
Table 3 Time of building database and database size.

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

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分かる．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，すべてのソースコードの登録処理は，利用開始時の 1 度だけ行えばよく，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分かる．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 28 GB 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．

JDK 1.6.0 のデータベースを用いて，データベース利用時に要した時間を測定した．ランダムに選んだキー文字列を用いて検索結果の上位 10 件を取得するのに必要な時間は，すべて 500 ミリ秒以下となった．注意すべきことは，キー文字列はハッシュ値に変換されるため，キー文字列のトークン数は取得時間に影響しないことである．測定時間には Eclipse 上に候補の表示をする時間は含まれておらず，キー文字列からバリュー文字列を取得するまでの時間である．

本稿での実装ではデータベースのキー文字列をあらかじめ並べ替えして保存しているため，利用時の時間に関しては，データベースのサイズに比例して時間がかかる処理ではない．データベースを利用する際，単純な問合せのみですませることでキーバリューストア型データベースの利点をいかすことができ，データのサイズが大きくなったとしても，現在の PC であれば問題なく利用できると考える．

3.2 適用例

提案手法を Ant 1.8.1 [*3] 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 章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．

[*3] http://ant.apache.org/

```
/**
 * 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）
Fig. 5 Source code for file retrieval via FTP (Ant 1.8.1).

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 であ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開発者に提示する．

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 にあ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．

3.3 有効性の評価

提案手法の有効性を評価する実験を行った．トークン数が少ない場合は大量の候補が表示され，開発者が想定しているソースコードと無関係なものまで表示される可能性がある．そのため，開発者がどの程度のトークンを入力した段階で適切なソースコードが表示されるか評価する．

利用場面は，開発者もしくは同一組織内で過去に記述されたソースコードを対象として，再利用する場合を想定

表 4 再利用事例数
Table 4 The number of reuse instances.

クローン率	再利用事例数
90%以上	32
80%以上 90%未満	90
70%以上 80%未満	211

している．そこで，ソースコードが公開されているソフトウェアの中から異なるバージョンで似ているメソッド（再利用の事例とよぶ）を探しだし，バージョンの新しいメソッドを本手法を実装した Eclipse 上に入力していき候補が表示されるか実験した．具体的には，オープンソースソフトウェア Ant からメソッドのソースコードが再利用された事例を収集した．Ant 1.1 から 1.8.2 までの 23 リリースバージョンの中で，直前のリリースバージョンに含まれるメソッドとクローン率が閾値以上のメソッドを抽出し，再利用の事例とした．

3.3.1 再利用事例の抽出方法

リリースバージョン V_n に含まれるメソッド m_a と直前のリリースバージョン V_{n-1} に含まれるメソッド m_b のクローン率が閾値以上であった場合，V_{n-1} に含まれるメソッド m_b を再利用して，V_n に含まれるメソッド m_a が作成されたとする．また，このとき m_a と m_b は再利用関係を持つとする．この m_a と m_b のペアを再利用事例とよぶ．

再利用事例クローン率は，(メソッド m_a とメソッド m_b 間でクローンになっているトークン数の和)/(メソッド m_a とメソッド m_b のトークン数の和) で計算される．CCFinder [8] を最小一致トークン数 30 に設定し検出した．ただし，m_a もしくは m_b が 100 トークン未満の再利用事例は取り除いた．

表 4 にクローン率と再利用事例の検出数を示す．クローン率によって 3 種類の分類に分け，事例数を調査した．その結果，クローン率が 90%以上一致する再利用事例は 32 個存在した．以下，80%以上 90%未満の場合は 90 存在し，70%以上 80%未満は 211 と最も多くの再利用事例が存在した．各分類のそれぞれを事例集とよぶ．

3.3.2 検出結果

検出した再利用事例の m_a の冒頭を検索クエリとして Eclipse の Java エディタ上に入力した．そして，直前のリリースバージョン V_{n-1} に含まれるメソッド m_b を効率的に提示できるかを適合率および再現率で評価した．検索クエリは，先頭から 30，40，50，60，70，80，90，100 トークンの 8 通りを作成し入力した．適合率および再現率の定義は以下のとおりである．

適合率 1 つの事例集に含まれる各再利用事例から検索クエリを作成・入力し，提示メソッド（提示された補完候補）の中の正答メソッド（検索クエリを含むメソッドと再利用関係を持つメソッド）の割合を求め，平均した値．

再現率 1 つの事例集に含まれる各再利用事例から検索クエリを作成・入力し，正答メソッド（検索クエリを含むメソッドと再利用関係を持つメソッド）の中の，提示メソッド（提示された補完候補）の割合を求め，平均した値．

表 5 適合率と再現率
Table 5 Precision and recall.

適合率

トークン数＼閾値	90%以上	80%–90%	70%–80%
30	0.888	0.274	0.203
40	0.842	0.128	0.038
50	0.856	0.085	0.026
60	0.8	0.092	0.025
70	0.867	0.133	0.02
80	1	0.267	0.08
90	1	0	0.182
100	1	0	0

再現率

トークン数＼閾値	90%以上	80%–90%	70%–80%
30	0.781	0.589	0.464
40	0.781	0.211	0.052
50	0.781	0.111	0.024
60	0.5	0.033	0.005
70	0.313	0.022	0.005
80	0.094	0.022	0.005
90	0.063	0	0.005
100	0.063	0	0

結果を**表 5** に示す．トークン数とはエディタ上に入力した先頭のトークンの数を表し，閾値は再利用事例のどの種類かを表している．

3.3.3 考察

適合率，再現率ともにクローン率が 90%以上の事例集を入力とした場合が最も高く，70%以上 80%未満のクローン率の場合が最も低かったため，メソッド間クローン率が高い事例に対して本手法は有効である．メソッド間クローン率が低い事例に対する適合率・再現率を向上させるためにトークン列ではなく構文木やプログラム依存グラフなど抽象度の高いプログラム表現に基づいて補完候補を検出する方法が考えられる．しかし，これらの手法を用いるとソースコードを解析する際の空間コストや解析結果を利用する際の時間コストが本手法に比べ大きくなる．本研究では，低コストで，かつ開発者に特別な入力を必要としない手法で有効な再利用を目指している．

検索クエリのトークン数を増加させると適合率が上昇し，再現率および F 値が低下したため，適合率と再現率のバランスを重視する場合は，先頭から 30 トークンを入力

後，補完候補を求めるべきと考える．今回の実験は高いクローン率に基づく実験のため，同一人物や同一組織が過去に作成したソースコードをもとに再利用する場面を想定している．同じようなロジックのソースコードを開発者が記述する際に，先頭から 30 トークン入力すれば残りのソースコードが補完できることを示している．

適合率を向上させるためには，検索クエリ中と補完候補間で呼び出しているメソッドが一致しているかどうかなど，意味解析の結果の類似性を補完候補の検出に用いる方法があげられる．また，複雑度などのメトリクス値の類似性を用いる方法も併用することで向上すると考える．

本実験では，メソッド単位の補完を行う実験のみを行ったため，より小さいブロック単位の補完を行った場合は，性能が異なる可能性がある．しかし，メソッド単位でなければ抽出できない情報は用いていないため，同規模のブロックや再利用事例が対象である場合，大きな性能差はないと考える．

本実験では，コードクローン検出ツール CCFinder を利用して計測したメソッド間クローン率に基づいて，7 割以上のトークンを再利用した事例を収集した．そのため，他のコードクローン検出ツールを用いる，もしくはコピーアンドペーストの履歴を用いて再利用事例を収集した場合，異なる実験結果になる可能性がある．

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 [5] があげられる．この手法は，メソッドを行数や複雑度など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 [11] や A-SCORE [12] があげられる．これらは，開発者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．

このほかにも，ソースコードを収集し，その中から開発者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], [7], [10]．公開されているものとしては，Google ソースコード検索 [4] や SPARS-J [10], Koders [1] などのソースコード検索エンジンがあげられる．これらは，開発者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 [6] 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なく，必要に応じてソースコードの再利用を行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．

参考文献

[1] Black Duck Software, Inc.: Koders, available from ⟨http://www.koders.com/⟩.
[2] Chatterjee, S., Juvekar, S. and Sen, K.: SNIFF: A Search Engine for Java Using Free-Form Queries, *Proc. FASE 2009*, pp.385–400 (2009).
[3] Codase Inc.: Codase, available from ⟨http://www.codase.com/⟩.
[4] Google: Google Code Search, available from ⟨http://www.google.com/codesearch⟩.
[5] Hill, R. and Rideout, J.: Automatic Method Completion, *Proc. ASE 2004*, pp.228–235 (2004).
[6] Holmes, R., Walker, R.J. and Murphy, G.C.: Approximate Structural Context Matching: An Approach to Recommend Relevant Examples, *IEEE Trans. Softw. Eng.*, Vol.32, No.12, pp.952–970 (2006).

[7] 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).
[8] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol.28, No.7, pp.654–670 (2002).
[9] 三宅達也，肥後芳樹，楠本真二，井上克郎：多言語対応メトリックス計測プラグイン開発基盤 MASU の開発，電子情報通信学会論文誌 D，Vol.J92-D, No.9, pp.1518–1531 (2009).
[10] The SPARS Project: SPARS-J, available from ⟨http://demo.spars.info/⟩.
[11] Ye, Y., Fischer, G. and Reeves, B.: Integrating Active Information Delivery and Reuse Repository Systems, *Proc. SIGSOFT 2000/FSE-8*, pp.60–68 (2000).
[12] 島田隆次，市井 誠，早瀬康裕，松下 誠，井上克郎：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，Vol.50, No.12, pp.3095–3107 (2009).

山本 哲男 （正会員）

平成 9 年大阪大学基礎工学部情報工学科卒業．平成 14 年同大学大学院博士後期課程修了．同年科学技術振興事業団計算科学技術研究員．平成 16 年立命館大学情報理工学部情報システム学科講師．平成 20 年同学科准教授．平成 23 年日本大学工学部情報工学科准教授．博士（工学）．ソフトウェア開発支援環境の研究に従事．電子情報通信学会，IEEE-CS 各会員．

吉田 則裕 （正会員）

平成 16 年九州工業大学情報工学部知能情報工学科卒業．平成 21 年大阪大学大学院情報科学研究科博士後期課程修了．平成 22 年奈良先端科学技術大学院大学情報科学研究科助教．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．ソフトウェア科学会，電子情報通信学会，人工知能学会，IEEE，ACM 各会員．

肥後 芳樹 （正会員）

平成 14 年大阪大学基礎工学部情報科学科中退．平成 18 年同大学大学院博士後期課程修了．平成 19 年同大学院情報科学研究科コンピュータサイエンス専攻助教．博士（情報科学）．ソースコード分析，特にコードクローン分析やリファクタリング支援に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．