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.

再利用実績に基づいたコード片検索手法の提案

石原　知也† 　堀田　圭佑† 　肥後　芳樹† 　楠本　真二†

† 大阪大学大学院情報科学研究科

あらまし　ソースコードの再利用を支援する手法の 1 つとしてコード片検索手法が広く知られている．コード片検索手法とは，ユーザが求める機能を表すクエリを入力として与えると，クエリに関連する機能を有するコード片を提示する手法である．しかし，既存のコード片検索手法は，プログラミング言語の構造を提示するコード片の単位としている．そのため，手法が提示するコード片の規模や抽象度がユーザの求めるものとは異なる場合がある．特に，ユーザの不要な部分を含むコード片を提示することが多い．本研究では，再利用実績を考慮したコード片検索手法を提案する．過去に再利用されているコード片のみを提示することで，ユーザの必要なコード片のみが提示される．本研究では，6 名の被験者の協力のもと，既存手法との比較を行った．実験の結果，提案手法を使うことで既存手法に比べて効率的にソフトウェアの開発を行えることが確認された．

キーワード　コードクローン，コード片検索，ソースコード再利用

Searching code fragments based on past reuse

Tomoya ISHIHARA†, Keisuke HOTTA†, Yoshiki HIGO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University

Abstract　Code fragment search techniques are well-known as the one of the techniques helping code reuse. If users input queries that represent functionalities that they want, the techniques suggest code fragments that have the functionalities. However, conventional techniques suggest code based on structural unit of programming languages. Hence, it is possible that sizes and abstraction levels of code fragments suggested by the techniques are different from those of users' requirements. In particular, they often suggest code fragments including extra functions that users do not need. In this research, we propose a new code fragment search technique based on past reuse. This technique suggests only code fragments including functionalities that users actually want because it suggests only code fragments that have been reused. In this research, we conducted an experiment with 6 participants in order to compare the proposed technique with conventional techniques. As a result, we confirmed that users could develop software efficiently by using the proposed technique.

Key words　Code Search, Code Clone, Source Code Reuse

1. はじめに

ソフトウェア開発において，ソースコードの再利用を行うことは有益である．既存のソースコードを適切に再利用することで，開発者が新しい機能を実装する必要がなくなり，開発効率が向上する．また，十分にテストされているソースコードを再利用すれば，信頼性の高い機能を簡単に利用することができる．このように，ソースコードの再利用には様々なメリットがあり，現在までにソースコードの再利用を支援する研究が多く行われている．

ソースコードの再利用を支援するシステムの 1 つにコード片検索システムがある [1–3]．コード片検索システムは，ユーザが求める機能を表現したクエリを入力すると，クエリに関連する機能が実装されているソースコードを出力する．また，コード片検索システムは独自の指標をもっており，その指標に基づいて出力するソースコードの順番を決定する．コード片検索システムの利点の 1 つはユーザに対して複雑な操作を要求しない点である．ユーザが行う操作はシステムに対してクエリを入力するだけである．

しかし，既存のコード片検索システムには問題点も存在する．既存システムはユーザが必要としていない機能を含むソースコードを提示することがある．そのため，ユーザは提示されたソースコードから自分が必要な機能を探すコストが生じる．この問題の原因は，既存のコード片検索システムがプログラミング言語の構造を基にソースコードを提示することにある．多くの既存のコード片検索システムはファイルやクラス単位でソー

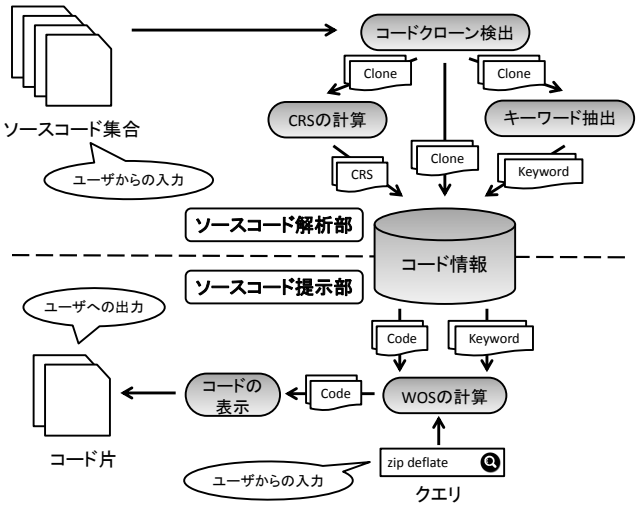

図 1　提案手法の概要

スコードを提示するため，特にユーザが単一の機能や規模の小さい機能の再利用を行う際にこの問題が発生しやすい．ユーザは常に同じ規模や抽象度の機能を求めるわけではない．時には数行で成り立つような規模の小さいコード片を再利用することもあり，また時にはクラス全体を再利用することもある．そのため，コード片検索システムはユーザの要求に合った規模や抽象度のソースコードを提示する必要がある．

本研究では，過去に再利用されたコード片を検出し，その中からユーザの入力するクエリと関連する機能を持つコード片を提示する手法を提案する．既存研究はメソッドやクラス等のプログラミング言語単位でコード片を提示するのに対し，提案手法は過去に行われた再利用を単位としてコード片を提示する．また，既存手法である API 検索 [2] とは異なり，コード片内に API が存在しなくても過去に再利用が行われていればそのコード片を提示対象にすることができる．再利用されたコードが存在するということは，過去に誰かがそのコード片を必要としたことを示す．過去に再利用されたコード片は今後再び再利用される可能性が高いと著者らが考えており，このようなコード片を提示することで既存手法と比較してユーザの要求をより満たすことができると考えられる．提案手法では過去の再利用を検出するためにコードクローン検出を利用している [4]．コードクローンの発生原因の 1 つにコードの再利用があるため，コードクローンを検出することで過去の再利用を特定している．

また，6 名の被験者の協力のもと，小規模な実験を行った．被験者は提案手法を実装したツールを含む 3 つのツールを用いて与えられたタスクを完成させ，その完成までの時間を比較することで有効性を評価した．実験の結果，提案手法は既存手法と比べ効率的な再利用支援を行えることが確認された．

本論文の貢献は以下のとおりである．

- 過去に再利用されたコード片を検出し，そのコード片をユーザに提示する手法を提案した．
- 小規模な実験を行い，提案手法の有効性を確認した．

2. コード片検索手法の構成

2.1 概　　要

本研究では，ユーザの要求にあった規模や抽象度のコード片を提示するために，過去の再利用に基づいたコード片検索手法を提案する．図 1 は提案手法の概要を示す．提案手法はあらかじめ対象となるソースコード集合を解析し，解析した情報をデータベースに登録する．提案手法に対しユーザがクエリを入力すると，提案手法はデータベースの情報を基にクエリに関連するコード片をユーザに提示する．

提案手法は**ソースコード解析部**と**ソースコード提示部**の 2 つから構成される．ソースコード解析部では，提案手法は提示対象となるコード片集合を構築するために対象となるソースコード集合からコードクローンを検出する．ソースコード提示部では，提案手法はソースコード解析部で構築したコード片集合の情報に基づいてユーザが入力したクエリに関連するコード片を提示する．ソースコード提示部でコード片を提示するためには，それ以前にソースコード解析部が実行されデータベースが構築されていなければならない．ソースコード解析部はデータベースを構築するために 1 度だけ実行されればよいのに対し，ソースコード提示部はユーザがクエリを入力するたびに実行される．

2.2 ソースコード解析部

ソースコード解析部は以下の 3 つのステップから構成される．

STEP1: 提示対象となるコード片集合を構築するためにコードクローンを検出する．

STEP2: コードクローンとして検出された各コード片に対して，メソッドの呼び出し関係を基に重要度を計算する．

STEP3: 各コード片からキーワードとなる単語を抽出する．

提案手法は，過去に再利用されたコード片のみをユーザに提示する．過去の再利用の特定はコードクローン検出を用いることで実現できる．コードクローンの発生原因の 1 つにコードの再利用がある．そのため，コードクローンとして検出されたコード片は過去に再利用されたコード片であるとみなすことができる．

加えて，ユーザの要求にあったコード片を提示するために，提案手法は各コード片の重要度を計算する．本研究では，各コード片の重要度の計算にコンポーネントランク法 [1] を用いる．コンポーネントランク法は関数間の呼び出し関係を基に関数の重要度を計算する．コンポーネントランク法はソースコードの静的な情報のみを用いるため，ユーザが入力するクエリとは独立に重要度を計算できる．提案手法では，コード片の重要度をそのコード片を含む関数の重要度と等しい値をとると定義する．このように定義することで，コード片の重要度をコンポーネントランク法を用いて計算することができるようになる．また，提案手法は過去の再利用回数も重要度の 1 つとして考慮する．つまり，過去に何度も再利用が行われているコード片は重要度が大きくなる．

2.3 ソースコード提示部

ソースコード提示部は以下の 3 つのステップから構成される．

STEP1: データベースの情報を用いて，ユーザが入力した

クエリに関連するコード片を見つける．

STEP2: コード片の重要度やクエリとコード片の関連性の強さを基にコード片に順位をつける．

STEP3: STEP2 でつけた順番でコード片を提示する．

本研究では，*Component Rank Score(CRS)* と *Word Occurence Score(WOS)* [3] という 2 つの指標を基にコード片を順位づけする．*CRS* はコンポーネントランク法によって計算された重要度のことを示す．*WOS* はユーザの入力したクエリとコード片の関連度の強さを表す．コード片の機能をよく示すキーワードとクエリがマッチするほど，*WOS* の値は大きくなる．提案手法はあらかじめソースコード解析部で *CRS* を計算しておくのに対し，ユーザがクエリを入力するたびにソースコード提示部で動的に *WOS* を計算する．最終的に，提案手法はこの 2 つの指標を基に決定した順番でコード片を提示する．

3. コード片検索手法の詳細

3.1 コードクローン検出

本研究では，様々な規模や抽象度のコード片をユーザに提示するため，規模の小さいコードクローンも見つけなければならない．細粒度なコードクローン検出手法を使用すれば，数行のコードクローンからファイル全体に渡るコードクローンまで検出できる．一方で，ファイルクローン検出法などの粗粒度なコードクローン検出手法を使用した場合は，規模の大きいコードクローンしか検出できない．また，本研究では大量のソースコードからコードクローンの検出を行うため，スケーラブルなコードクローン検出が求められる．以上の点から，本研究ではインデックスベースでありかつ文単位のコードクローン検出手法 [4] を利用する．この手法は以下のステップを通じてコードクローンを検出する．

STEP1: すべてのメソッドから文の並びを抽出する．

STEP2: p だけ連続した文に対してハッシュ値を計算する．p の値はあらかじめユーザが決定する．

STEP3: ハッシュ値が一致する文の並びをコードクローンとして検出する．

提案手法は STEP3 におけるハッシュ値の比較によってスケーラブルなコードクローン検出を実現している．

3.2 コンポーネントランク法

提案手法はコード片の重要度の計算にコンポーネントランク法 [1] を使用する．コンポーネントランク法は呼び出し関係を基に各関数の重要度を計算する手法で，ウェブページの相対的な重要度を計算する *PageRank* 手法を応用している．コンポーネントランク法は以下の 2 つのコンセプトを基に重要度を計算する．

- 多くの関数に呼ばれている関数は重要である．
- 重要な関数から呼ばれている関数は重要である．

また，コンポーネントランク法は類似関数をグループ化している．コンポーネントランク法では，メトリクスを用いた類似関数の特定を行っている [5]．各関数に対してその関数を構成するトークンの種類別出現頻度を類似度メトリクスと定義し，このメトリクス値やサイクロマチック数等の複雑度メトリクス値の類似性により関数の類似性を判定している．グループ内の要素の重要度の合計をグループの類似度と再定義することで，類似関数を統合した重要度を計算することができる．

本研究では，各コード片の重要度を計算するために，コンポーネントランク法のアルゴリズムを使用して各メソッドの重要度を計算する．コード片の重要度はそのコード片を含むメソッドの重要度と等しい値として定義する．

3.3 キーワード抽出

提案手法ではコードクローンとして検出されたコード片からキーワードを抽出する．抽出されたキーワードのみをクエリとのマッチングに使用するため，コードクローンとして検出されないコード片は提示対象にはならない．キーワードとして抽出されるのは，変数名，メソッド名，クラス名やメソッドやクラスに付随する Javadoc コメントに存在する単語である．また，提案手法は抽出したキーワードに対してステミングやストップワードの除去などの自然言語処理を施す．このようにして抽出されたキーワードは最終的にはデータベースに保存される．

3.4 クエリとキーワード間における関連性の強さの計算

本研究では，*WOS* 値を計算する手法として *TF-IDF* 法 [6] を採用する．*TF-IDF* 法は，*TF* と *IDF* という 2 つの指標を基にすべての文書内に存在する各単語の重要度を計算する．*TF* はある文書内に存在するある単語の出現回数を示す．*IDF* はある単語を含む文書の逆頻度を示す．*TF-IDF* 法は文書検索システムで広く用いられている手法である．

提案手法では，キーワードを *TF-IDF* 法における単語，コード片を *TF-IDF* 法における文書として扱う．この時，提案手法は以下の式を用いて *WOS* 値を計算する．

$$W_{wos}(i,j) = \frac{t_{i,j}}{|T_i|} \times \log \frac{|D|}{d_i} \tag{1}$$

- $t_{i,j}$ はコード片 j に含まれるキーワード i の出現回数を示す．
- T_i はすべてのコード片におけるキーワード i の集合を示す．
- D はすべてのコード片の集合を示す．
- d_i はキーワード i を含むコード片の数を示す．
- $|S|$ は集合 S の要素数を示す．

入力されたクエリが少数のコード片からしか抽出されないキーワードとマッチする場合，そのコード片の *WOS* 値は大きくなる．

3.5 指標の統合

提案手法は *CRS* と *WOS* の 2 つから各コード片の相対的な重要度を決定する．本研究では，*CRS* と *WOS* それぞれでコード片の順位を決定し，2 つの順位を足しあわせてその値が小さいコード片ほど重要度を高く設定している．最終的に，統合された重要度が大きいコード片からユーザに提示される．

3.6 実　　装

本研究では，提案手法を実装したプロトタイプを試作した．ただし，提案手法は高速にコードクローン検出を行い，また高速に重要度の計算を行う必要がある．そのため，この実装にはいくつかの工夫が施されている．例えば，提案手法はコンポー

ネントランク法の計算において近似計算を行っている．既存手法はクラスに対してコンポーネントランク法の計算を行うのに対し，提案手法はメソッドに対して計算を行う．通常クラスの数に対してメソッドの数は極めて多いため，提案手法における計算量が爆発的に増加する．そのため，提案手法はプロジェクト内のメソッド呼び出しとプロジェクト外へのメソッド呼び出しを別々に計算した後に統合するという近似計算を行っている．この近似計算は広く用いられている [7]．

4. 実　　験

4.1 実験の準備

提案手法の有効性を評価するために，3.6 で実装したプロトタイプを用いて比較実験を行った．この章では，以後このプロトタイプを P と呼ぶ．本研究では，実験対象のソースコード集合として既存研究である *SPARS* で使用されているソースコード集合を使用した [1]．このソースコード集合は約 400 のプロジェクトで構成されており，約 19 万の Java ファイルが存在する．このソースコード集合に対して P を適用し，データベースを作成した．

加えて，我々は比較のために以下の 2 つのツールを実装した．

- 提案手法から過去の再利用回数の考慮を除いたもの．例えば，過去に 10 回再利用されたコード片と 1 回しか再利用されなかったコード片が等しく扱われる．以後このツールを V_1 と呼ぶ．
- コードクローン検出を行わず，プログラミング言語の構造のみを使ってコード片を提示するもの．このツールはメソッド単位でコード片を提示する．以後このツールを V_2 と呼ぶ．

P と V_1 を比較することで，過去の再利用を考慮することが順位付けに有効であるかを評価できる．また，P と V_2 を比較することで，再利用単位でのコード片の提示が有効であるかを評価できる．

上記の 2 つのツールも P と同様に実験対象のソースコード集合に適用して，データベースを作成した．

4.2 実験方法

本研究では，複数の被験者に 3 つのツールのいずれかを使用してもらい，与えられたタスクを完成させてもらう．初めに，被験者に対して著者がツールの使い方と各タスクの内容を説明する．説明の後，被験者はそれぞれ独立にタスクを開始する．各タスクにおいて，被験者は自分で入力するクエリを決める．入力したクエリに合うコード片が存在しない場合は，被験者は入力するクエリを自由に変更できる．与えられた終了条件をみたすことを被験者が確認した段階でタスクが終了する．その際，被験者はタスクの開始から終了までの時間を計測しておく．本実験では，実際の再利用の状況を考慮して，クエリの入力回数を制限しなかった．また，各タスクについて 20 分の制限時間を設けた．

表 1　各グループがタスクを完了させるために使用したツール

	タスク 1,2,3	タスク 4,5,6	タスク 7,8,9
グループ 1	P	V_1	V_2
グループ 2	V_2	P	V_1
グループ 3	V_1	V_2	P

```
157  protected String streamAsset(HttpServerRequeset request,
                        HttpServerResponse response)
158  {
                        :
                        :
260     ServletOutputStream out = response.getOutPutStream();
261     ZipOutputStream zip = new ZipOutputStream(out);     //Servlet out
262     zip.setMethod(ZipOutputStream.DEFLATED);
263     zip.setLevel(Deflater.BEST_COMPRESSION);
264     zip.setComment(readme);
265
266     //  Readme File
267     ZipEntry entry = new ZipEntry("readme.txt");
268     entry.setExtra(assetInfo);
269     zip.putNextEntry(entry);
270     zip.write(readme.getBytes(), 0, readme.getLength());
271     zip.closeEntry();
                        :
                        :
328  }
```

(a) ツールの出力例.P はハイライトされた部分のみを出力する.V_2 はメソッド全体を出力する．

```
                        :
                        :
     FileInputStream in = new FileInputStream(input);
     ZipOutputStream zip = new ZipOutputStream(output);   //Servlet
                                                          //out
     zip.setMethod(ZipOutputStream.DEFLATED);
     zip.setLevel(Deflater.BEST_COMPRESSION);

     final byte[] buf = new byte[1024];
     zip.putNextEntry(new ZipEntry(input.getName()));
     int len;
                        :
                        :
```

(b) 被験者が実装したコード

図 2　ツールの出力例と被験者が実装したコードの例

4.3 タ　ス　ク

本実験では，既存研究 *SPARS* で使用されたタスクを使用した [1]．ただし，既存研究にある 10 のタスクのうち簡単に実行でき結果の確認が行える 9 のタスクを採用した．また，被験者に各タスクについて 3 段階で難易度を決めてもらった．その際に，各難易度に該当するタスクの数を 3 にするという条件を設けた．表 2 は本実験で使用したタスクの内容を示している．

4.4 実験結果

表 3 は実験の結果を示している．2，3，4 行目は被験者がタスクを完成させるまでに要した時間の合計を表している．被験者がタスクを終了した後に，すべてのタスクについて著者が正しくタスクを実行できているかを確認している．また，被験者が制限時間内にタスクを終了できなかった場合は，タスクの終了時間を 20 分としている．5 行目はタスクの難易度を示している．6 行目は各タスクにおける最も短い時間でタスクを終了させたツールを表している．タスク 3 に関しては，被験者全員が制限時間内にタスクを終わらせることができなかったため，

表 3　被験者が各タスクを完了させるのに要した時間の合計

	タスク 1	タスク 2	タスク 3	タスク 4	タスク 5	タスク 6	タスク 7	タスク 8	タスク 9
P	1,091	1,393	2,400	2,400	1,525	848	1,360	1,529	853
V_1	1,309	1,109	2,400	1,994	367	1,286	2,400	1,890	2,011
V_2	365	925	2,400	1,508	689	954	1,930	2,400	240
難易度	易	易	難	難	易	中	中	難	中
最良ツール	V_2	V_2	-	V_2	V_1	P	P	P	V_2

どのツールも該当しないことを表している．

Tukey の HSD 検定を用いて $\alpha = 0.05$ で各ツールの平均終了時間を検定したところ，各ツール間で統計的に差はないことがわかった．そこで，各ツールにおいて最も終了時間を短くさせるツールについて調査した．表 3 を見ると，V_2 が 4 つのタスクにおいて最も効率的に再利用を支援していることがわかる．しかし，P と比べると V_2 は比較的難易度の低いタスクにおいて終了時間が短い．一方で，P は比較的難易度の高いタスクにおいて終了時間が短くなっている．

このような差が生じる原因として，各タスクで実装する機能の規模の違うことが考えられる．多くの難しいタスクは複数のメソッドを組み合わせて実装しなければならないが，簡単なタスクは 1 つのメソッドで実現可能である．P は各メソッドの必要な部分だけを提示するため，必要な部分だけを抽出して簡単に機能の組み合わせが実現できる．一方で，V_2 はメソッド内でどの部分がタスクを実現するために必要なのかを調べるコストが必要になる．ただし，タスク 4 に関しては 1 つのメソッドで実現できるにも関わらず，被験者の Java Applet の知識の少なさから難易度が高くなっている．図 2 は P と V_2 が提示したコードの一例である．P は図 2(a) のハイライトされている部分のみを提示したのに対し，V_2 は約 200 行もあるメソッドの全体を提示した．ソースコードの再利用の効率性を高めるには，実装が難しいとユーザが思っているタスクに対しての支援が必要不可欠である．言い換えれば，ユーザが簡単に実装できるタスクに対する支援はさほど必要ではない．そのため，実験の結果から P は V_2 に比べて有効的な再利用支援を実現できているといえる．

また，P と V_1 を比較すると，半数以上のタスクで P の終了時間のほうが短いことがわかる．例えば，P は “zip deflate” とクエリを入れると図 2(a) のコード片を最初に提示する．一方で，V_1 は同様のクエリを入力すると 43 番目に図 2(a) のコードを提示する．このことから，過去の再利用回数を考慮することでユーザの要求にあったコード片を早く提示することができるといえる．

表 2　実験で使用したタスク

タスク	内容
タスク 1	クイックソートアルゴリズムを実装する
タスク 2	2 分探索のアルゴリズムを実装する
タスク 3	アナログ時計を表示するアプレットを作成する
タスク 4	テキストエリアを表示するアプレットを作成する
タスク 5	乱数生成を行う機能を実装する
タスク 6	プッシュやポップ等のスタックの機能を実装する
タスク 7	ファイルやディレクトリを ZIP 形式で圧縮する機能を実装する
タスク 8	クラスファイルをダンプする機能を実装する
タスク 9	ストリームを介してファイルを読み込み，そのコピーを書き出す機能を実装する

5. 妥当性への脅威

被験者 本実験では，6 人の被験者に 3 つのツールを使用してタスクを完成させてもらった．被験者は全員 Java の経験が 1 年以上あるが，被験者の間で Java プログラミングの能力に大きな隔たりがあった場合，実験結果に大きな影響を与えることになる．ただし，本実験では各グループに修士課程の学生と博士課程の学生が 1 人ずついるため，グループ間で能力の差がそれほど大きくないと考えられる．

時間制限 被験者は 20 分という制限時間内で各タスクを終了させなければならない．そのため，被験者が焦燥して再利用すべきコードを見失い，結果として必要以上に時間が掛かっている可能性がある．

実験対象 提案手法は提示するコード片の順番を決める指標の 1 つとしてコードクローンの数を考慮している．しかし，どのコード片がコードクローンとして検出されるかは検出の対象となるソースコード集合に依存する．そのため，実験対象のソースコード集合を変更することで結果が変わる可能性がある．

実装 本研究では，データベースの作成を高速化するために，いくつかの近似計算を実装している．例えば，各コード片の重要度を計算する際に，プロジェクト内のメソッド呼び出しとプロジェクト外へのメソッド呼び出しを別々計算し，後に統合するという近似計算が提案手法に実装されている．そのため，高性能マシンを使用する等で厳密な計算を行った場合，本研究の結果と異なる結果が得られる可能性がある．

6. 関 連 研 究

Inoue らは，関数の呼び出し関係からそれぞれの関数の重要度を計算するコンポーネントランク法とソースコード上の位置によってキーワードの重要度を変えるキーワードランク法を提案している [1]．また，それらを実装したソースコード検索システムである *SPARS* を開発した [1]．コンポーネントランク法では，多くの関数から呼び出されている関数の重要度は高くなる．また，重要な関数から呼び出されている関数は重要度が高くなる．加えて，コンポーネントランク法では類似関数をグループ化して，要素の重要度の合計をグループの重要度として再定義している．キーワードランク法では，ソースコードから

抽出されたキーワードの重要度をそのトークンの種類から決定している．トークンの種類が重要であるほどキーワードの重要度が高くなる．例えば，メソッド名やクラス名から抽出されたキーワードは重要度が高くなる．

McMillan らは *Navigation Model* と *Association Model* という 2 つのモデルから関数の重要度を決定する手法を提案し，これらを実装した *Portfolio* を開発した [3]．*Navigation Model* は開発者がどのようにして関数をたどるかを表したモデルである．このモデルでは，関数の呼び出し関係を基にそれぞれの関数の重要度を計算しており，*PageRank* 手法を応用したものとなっている．*Association Model* はキーワード間の関連性を表したモデルである．このモデルでは，*Spreading activation* 法を用いてキーワード間の関連性を計算している．また，*Portfolio* はソースコードを提示する際に関数のコールグラフも提示する．ユーザはグラフをたどる事によってその関数の使い方を知ることができる．

上記の 2 つの既存手法は関数の呼び出し関係を基に重要度を計算するという観点で提案手法と類似している．特に，*SPARS* は類似関数のグループ化という点も同じである．しかし，これらの既存手法はプログラミング言語の構造を基にソースコードを提示している．一方で，提案手法は過去の再利用単位でコード片を提示する．そのため，特にユーザが小さい機能の再利用を行う際に，ユーザの必要な部分だけを提示するという点で既存手法に比べて提案手法は大きな効果を発揮できる．

Holmes らは実用的な再利用を支援する手法を提案している [8]．この手法は，開発者が再利用したいと考えている機能を見つけ出し，保存されている実用的な再利用情報を基に半自動的にその機能を補完する．Holmes らの手法はユーザが再利用したいと考えているソースコードをみつけて入力として与える必要があるのに対し，提案手法はあらかじめ多くのソースコードからコード片の情報を取り出してデータベースを構築するため，ユーザが知らないシステムからの再利用が可能である．

7. おわりに

本論文では，ユーザの要求にあったコード片の提示を行うために，過去の再利用を基にコード片を提示する手法を提案した．提案手法は各コード片の過去の再利用回数を記録し，コード片を提示する際の順番を決める指標の 1 つとして利用する．また，6 人の被験者に与えられたタスクを完成させてもらい，タスクが終了するまでの時間を比較するという実験を行った．被験者は提案手法を実装したツールを含んだ 3 つのツールを用いて与えられたタスクを完成させた．

実験の結果，提案手法は既存手法と比べ効率的な再利用支援を行うことができたことが確認された．また，過去の再利用回数を考慮することでよりよい順位付けが行われることも確認した．

今後の課題として，提案手法をウェブシステムとして実装して多くの人に使用してもらえる環境を構築することを考えている．これによって，多くの被験者を募って大規模な実験を行うことができ，提案手法の有効性を確認することができる．実装の際には，クエリとキーワードの関連性の強さだけでなく再利用のしやすさという観点での順位付けを取り入れることも考えている．また，提案手法をコード補完手法に適用することも考えている．ユーザが途中まで書いたコードに対してコード補完を実行することで，過去に再利用されたコード片が自動的に補完される．

謝辞　本研究は，日本学術振興会科学研究費補助金基盤研究 (S)(課題番号：25220003)，挑戦的萌芽研究 (課題番号：24650011)，および文部科学省科学研究費補助金若手研究 (A)(課題番号：24680002)，独立行政法人情報処理推進機構　技術本部　ソフトウェア高信頼化センター（SEC:Software Reliability Enhancement Center）が実施した「2012 年度ソフトウェア工学分野の先導的研究支援事業」の支援を受けた．

文　献

[1] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto, “Ranking significance of software components based on use relations,” IEEE Transactions on Software Engineering, vol.31, no.3, pp.213–225, 2005.

[2] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, “A search engine for finding highly relevant applications,” Proc. of the 32nd International Conference on Software Engineering, pp.475–484, 2010.

[3] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, “Portfolio: Finding relevant functions and their usages,” Proc. of the 33rd International Conference on Software Engineering, pp.111–120, 2011.

[4] B. Hummel, E. Jürgens, L. Heinemann, and M. Conradt, “Index-based code clone detection: incremental, distributed, scalable,” Proc. of the 26th International Conference on Software Maintenance, pp.1–9, 2010.

[5] K. Kobori, T. Yamamoto, M. Matsushita, and K. Inoue, “Classification of java programs in spars-j,” Proc. of the International Workshop on Community-Driven Evolution of Knowledge Artifacts, 2003.

[6] K. Jones, “A statistical interpretation of term specificity and its application in retrieval,” Journal of Documentation, vol.28, no.1, pp.11–21, 1972.

[7] A.Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen, “Efficient pagerank approximation via graph aggregation,” Information Retrieval, vol.9, no.2, pp.123–138, 2006.

[8] R. Holmes and R.J. Walker, “Systematizing pragmatic software reuse,” ACM Transactions on Software Engineering and Methodology, vol.21, no.4, pp.1–44, 2012.