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.

類似するコーディングパターンの利用状況調査ツールの提案

小笠原康貴† 　神田　哲也† 　井上　克郎†

† 大阪大学大学院情報科学研究科　〒560-0871 大阪府吹田市山田丘 1 − 5
E-mail: †{k-ogaswr,t-kanda,inoue}@ist.osaka-u.ac.jp

あらまし　コーディングパターンとは，ソースコード中に頻出する定型的なコード片のことである．コーディングパターンを調査し再利用することは，開発作業を円滑に進めるための重要な作業である．しかしながら，コーディングパターンは多様化する傾向にあり，再利用者にとって適切なパターンを選択することが困難になっている．そのため，ソフトウェア中に類似するコーディングパターンがどの程度存在するのか，どの程度の割合で利用されているかを把握できることが好ましい．本研究では，検索クエリに複数の検索ワードを記述し，各検索ワードに一致するか一致しないかによりコードブロックを分別する検索を提案する．Java で書かれたオープンソースソフトウェアに対して提案手法を適用した結果，類似するコーディングパターンを含むコードブロックを分別して提示した上で，それぞれの利用割合を確認することができた．

キーワード　コーディングパターン, コード検索

Proposal of a usage investigation tool of similar coding pattern

Koki OGASAWARA†, Tetsuya KANDA†, and Katsuro INOUE†

†
E-mail: †{k-ogaswr,t-kanda,inoue}@ist.osaka-u.ac.jp

Abstract　Coding pattern is a code snippet that appears repeatedly in source code. Reusing coding pattern helps software developers. Therefore, it is imprtant for develpers to understand how similar coding patterns are used in software. We propose a search method that classifies code blocks according to the matching with multiple search words. In the case study, we confirmed that our approach can classify similar coding patterns and show the frequency of use for each pattern.

Key words　coding pattern,code search

1. ま え が き

コーディングパターンとは，ソースコード中に頻出する定型的なコード片のことである．コーディングパターンは特定の機能の典型的な実装方法を表していたり，特定のソフトウェアにおける実装上のルールを表すことが多い．そのため開発者が既存ソフトウェア中のコーディングパターンを把握することにはいくつかの利点が存在する．まず，開発者が特定の機能を実装する際に，同一の処理を行うコーディングパターンを再利用することで効率的に開発を行うことができる [1]．また開発者はコーディングパターンを確認することで、そのソフトウェアにおける実装上のルールに則った記述を行うことが容易になる．しかし，ソフトウェアが大規模複雑化することで，同一の処理に対して複数の類似したコーディングパターンが形成される場合がある．これにより，実装上のルールが不明瞭となり，ソフトウェアの保守に悪影響を与える [2]．さらに，類似する複数のコーディングパターンがあることで再利用者が適切なコーディングパターンを選択することが困難になる．

ソフトウェア中のコーディングパターンを検出するための研究としては，石尾らはシーケンシャルパターンマイニング [3] を用いたアプローチでソフトウェア中のコーディングパターンを検出した [4]．また，竹之内らはプログラミング言語の構文情報を考慮した検索クエリを記述可能なソースコード検索により，一種のコーディングパターンである API の利用法を表すコード片を検出した [5]．これらのアプローチでは，ソフトウェア中のコーディングパターンを効率的に検出することが可能であるが，類似しているが一部の記述が異なるコーディングパターンが区別されずに検出される可能性が存在する．そのため開発者が検出結果を再利用するためには，検出結果から開発者が求めるコーディングパターンを選択する必要がある．しかし，そのためには開発者がコーディングパターンを一つ一つ確認する必要があり，従来のコーディングパターン検出ではこのような確

認作業を行うことは困難であった．

本研究では，類似するコーディングパターンの利用状況を提示するためのソースコード検索を提案する．提案手法では、検索クエリとして複数の検索ワードを入力とし，コードブロックを各検索ワードを含むものと含まないものに分けて結果を提示する。これにより、従来は把握が困難であった類似するコーディングパターンを分離して提示することができる。また、検索結果の割合を見ることで、どのコーディングパターンが頻繁に利用されているかが容易に把握できる。

提案手法によるコードブロックのグループ化が，類似するコーディングパターンの利用状況の理解に有効であることを示すため，ソースコードの静的解析による実装を行った．ケーススタディでは，オープンソースソフトウェアに対し提案手法を適用することで，ソフトウェア中の類似するコーディングパターンを形成するコードブロックを分別してグループ化することが出来ることを確認した．その結果を考察することで従来では得ることが難しかったコーディングパターンの利用状況を得ることができたとともに，提案手法の問題点を確認した．

以降，2 章では本研究の背景について説明し，3 章では本研究の提案手法を説明する．4 章ではケーススタディについて説明し，最後に，5 章で本研究のまとめと今後の課題を述べる．

2. 背　　景

本章では，コーディングパターンとその関連研究について述べる．

2.1 コーディングパターン

ソフトウェア開発において，頻繁に使用される機能はモジュール化してまとめることが推奨される [6]．しかし，頻出する機能がモジュール化されずにソースコード中に分散して出現する場合がある [7]．そのように分散して出現する定型的なコード片をコーディングパターンと呼ぶ．コーディングパターンを形成する例として，API の利用が挙げられる．API には特定の処理を実装するためのイディオム的なメソッド呼び出し系列が存在することが多く，そのようなメソッド呼び出し系列はソースコード中に分散しコーディングパターンを形成する．例えば，Java において java.util.File クラスのインスタンスは，open メソッドと close メソッドを同時に呼び出して使用することが知られている．またコーディングパターンは，しばしば制御構造を伴って記述される．例えば，Iterator クラスのインスタンスは繰り返し処理とともに操作される場合が多い．図 1 に Iterator クラスのインスタンス操作が形成するコーディングパターンを記載する．

コーディングパターンを形成するコード片は，互いに同一の機能を実装するコード片であるため，類似していることがほとんどであり，コードクローン [8] を形成している場合が多い．したがって，ソフトウェア保守の観点から，コーディングパターンに属するコード片の 1 つを変更する場合は，同一のパターンに属する他のコード片に対する一貫した変更が必要か検討しなければならない [9]．またコーディングパターンを再利用する場合，類似したそれぞれのコード片から，再利用するべきコーディングパターンを状況に応じて判断する必要がある．しかし，このようなコーディングパターンの分析は，多くのコード片を確認しなければならず，必要としているコーディングパターンを発見することが困難となっている．

2.2 シーケンシャルパターンマイニングによるコーディングパターン検出

シーケンシャルパターンマイニングとは，与えられた配列データベースから頻出する部分列をパターンとして抽出する手法である [3]．頻出する部分列の個々の出現は，順序が同一でなければならないが，不連続なものでよい．石尾らはシーケンシャルパターンマイニングのアルゴリズムの 1 つである PrefixSpan を用いて，Java プログラムからコーディングパターンを抽出した [4]. この手法は，ソースコードの静的解析を行い，データベースを構築するコストがかかるが，検出条件を満たすコーディングパターンを一括して抽出することが可能であった．しかし，この手法により検出されるパターン数は膨大になり，ソースコードを確認して意味のあるコーディングパターンを抽出する作業が必要である．また石尾らは，コーディングパターンの意味を理解する際に，コーディングパターンの要素の一覧や，コーディングパターンのインスタンスを持つメソッドの一覧などの情報が有用であったと述べている．

2.3 パターンマッチを用いたソースコード検索

コーディングパターンの一種である API の典型的な利用法を調査するために，竹之内らはパターンマッチを活用した検索ツールを作成した [5]．竹之内らは特殊なトークンを含むクエリを用いた検索を行うことで，API の利用法を表すコード片の検出を可能とした．ソースコード検索を用いたアプローチは，パターンマイニングなどの手法と比べて，検索を行う前にデータベース構築などの処理を行う必要が無く，特定の API の利用法を知りたいといった要望に即座に対応できるという利点がある．しかし，検出結果に API の利用法を十分に表さないコード片が多く含まれる場合や，類似しているが用途の異なるコーディングパターンが混在する場合があり，利用者が検出結果を確認して，必要とするコード片を選択しなければならない問題が残っている．

2.4 コードクローンの検出

2.1 節で述べたように，コーディングパターンを形成するコード片は互いにコードクローンを形成している場合が多い．コードクローンの検出手法は多数提案されており，例えば，字句解析による検出を行う CCFinder [10]，抽象構文木の比較による検出を行う Jiang らの手法 [11]，プログラム依存グラフを比較

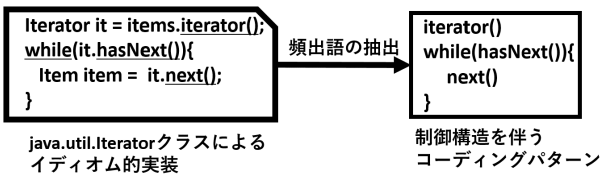

図 1　制御構造を伴うコーディングパターン

する Krinke の手法 [12] 等が挙げられる．これらの手法によりコードクローンを検出することができるが，検出の条件として，一致するコード片のサイズが一定以上の必要がある．コーディングパターンは短いコード片で形成される場合もあるため，一般的にコードクローンの検出法のみでコーディングパターンの検出を行うことは困難であるとされる [13]．

3. 提 案 手 法

本研究では，Java で記述されたソースコードを対象に，類似するコーディングパターンの利用状況を提示するためのソースコード検索を提案する．あるコーディングパターンに対して、それに類似するコーディングパターンは共通する部分と共通しない部分が存在する．類似するコーディングパターンを分別して提示するためには、まずソースコード全体から共通する部分を含むコードブロックを抽出したのち、共通しない部分の有無により分別する必要がある．この処理を実現するために，次の手法を提案する．

提案手法の入力はソースコードと検索クエリである．検索クエリに複数のワードを記述し，ソースコードから抽出したコードブロックに対し各ワードで検索を行う．そして，各検索ワードに一致するか一致しないかによってコードブロックを分別する．提案手法の出力として得られるのは，同一のワード群を持つコードブロックのグループである．

また，利用者の関心の高い検索ワードに注目してコードブロックを分別するために，提案手法の検索クエリでは特殊なオプション入力を行うことができる．このオプション入力により，利用者は任意の検索ワードを必須条件に指定することができ，検索結果から必須条件を満たさないものを省くことができる．

3.1 提案手法の利用方法

提案手法は，あるコーディングパターン，またはそれに類似するコーディングパターンの対象リポジトリ内での利用状況を知るために用いることができる．利用者は，知りたいコーディングパターンに含まれる，あるいは含まれそうなワードを複数選んで検索クエリに記述する．このクエリを用いてリポジトリ内のソースコードを検索することで，出力としてコードブロックのグループを得ることができる．同一のグループに含まれるコードブロックは，検索クエリに用いたワードの内，同じワードのみを含むため，それぞれのグループは互いに一部が異なるコーディングパターンの集合である．利用者は各グループ内のコードブロックを参照することができ，各グループにどの程度のコードブロックが含まれているかを確認することができる．概要を図 2 に示す．

次に，提案手法の特殊な入力について説明する．検索クエリを記述する際に，利用者は一部のワードを必須条件として設定することができる．検索を行うときに，コードブロック内に必須条件のワードが存在しなければ，そのコードブロックは検索結果に含まない．これにより，調査したいコーディングパターンの共通部分を明示するとともに，検索結果から利用者の興味のない検索結果を省くことができる．

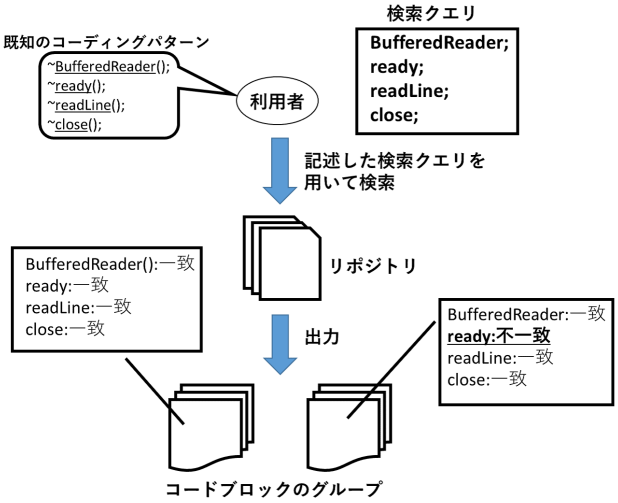

図 2　提案手法の利用方法

3.2 用語の説明

3.2.1 抽象構文木

提案手法では，Java の抽象構文木を用いる．抽象構文木とは，言語の構文構造を木構造で表したグラフのことを意味する．図 3 に抽象構文木の例を示す．

3.2.2 コードブロック

一般的にコードブロックとは複数の命令文を一括りにまとめたものである．本研究では，ソースコードの構文構造に則り，中括弧で閉じられた部分をコードブロックとして扱う．

3.3 提案手法の手順

提案手法は以下の手順で構成される．

（1） コードブロックの抽出
（2） コードブロックの検索
（3） コードブロックの分別

手順 1 :コードブロックの抽出

ソースコードからコードブロックの抽出を行う．まず，ソースコードを構文解析し，抽象構文木を生成する．このとき，抽象構文木の特定の部分木はコードブロックに相当する．例えば，メソッドの宣言は抽象構文木の部分木を構成するので，メソッドの宣言に該当するソースコードを，一つのコードブロックとして抽出できる．コードブロックの抽出例を図 4 に示す．この図では，メソッド宣言のソースコードがコードブロックとして

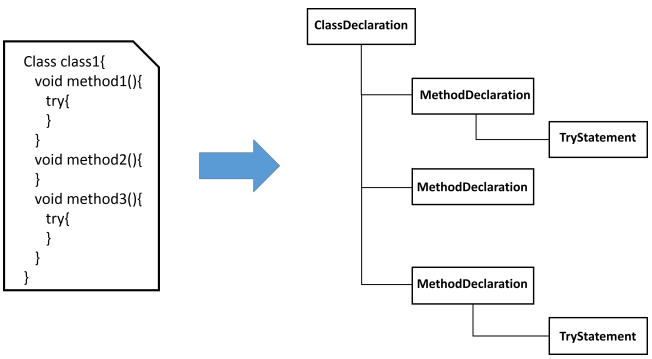

図 3　抽象構文木の生成例

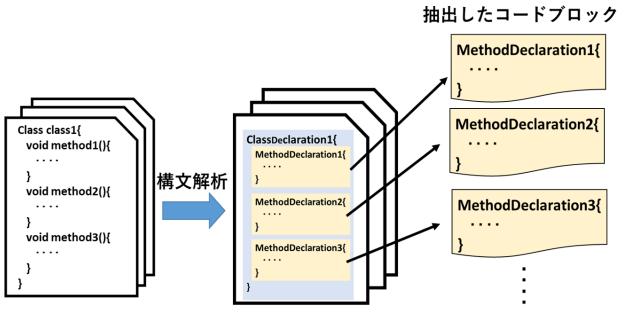

図 4　コードブロックの抽出

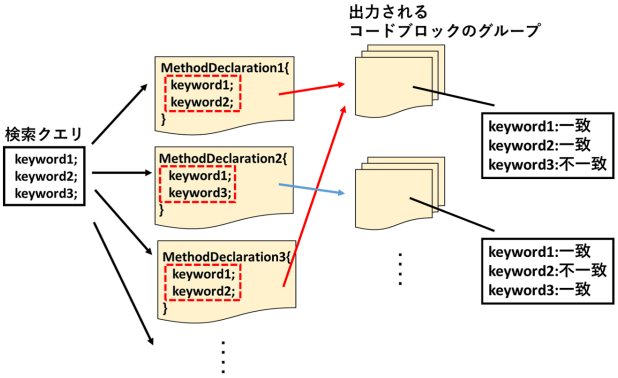

図 5　検索によるコードブロックの分別

抽出されている．

手順 2 :コードブロックの検索

抽出された各コードブロックに対して検索を行う．検索では，クエリに記述された一つ以上のワードを用いる．対象のコードブロック内に検索ワードが検出されるか否かを判定する．この作業を，検索クエリに記述された各検索ワードを用いて行う．

手順 3 :コードブロックの分別

検索により，それぞれのコードブロックは各ワードが検出されるか否かという結果を持つことになる．同一の検索結果を持つコードブロックをそれぞれグループにしたものが提案手法の出力である．手順 2，手順 3 の検索によるコードブロックの分別を図 5 に示す．

3.4　検索クエリの入力方法

提案手法で使用する検索クエリの入力方法について説明する．検索クエリにはコードブロック中に含まれているか確認したいワードを入力する．また，提案手法では一つの検索クエリに複数の検索ワードを記述することができる．そのため，各検索ワードの後ろにセミコロンを続けて入力することで検索ワードの区切りとすることができる．

次に，提案手法で使用する検索クエリでは，特殊なオプション入力を付けることができる．オプションを付けるためには，検索ワードとセミコロンを入力した後に“- e”と入力する．このオプションを付けることで，その検索ワードを必須条件として指定することができる．この時，コードブロック中にその検索ワードが含まれていなければ，そのコードブロックを出力から除外する．そのため，検索結果から状況に応じて不要な結果を除外することができる．

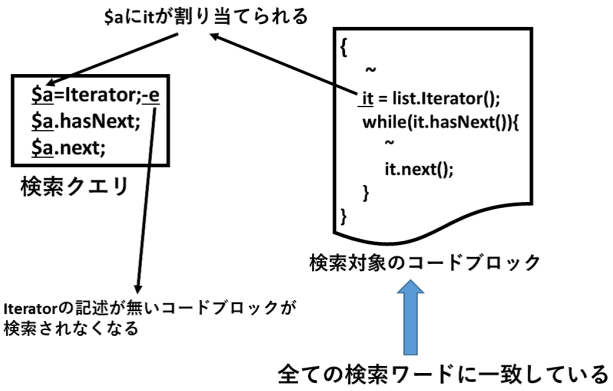

図 6　検索クエリの入力例

また，この検索クエリではトークンを用いて代入文とメソッド呼び出しに対応した検索を行うことができる．検索クエリ中に“$～=‥‥”と入力する．“～”の部分には任意の文字列を入力し，“‥‥”の部分には検索ワードを入力する．このような入力をした場合，コードブロック中の代入文だけを対象に検索を行う．その際に，検索に一致した代入文の左辺に記述されているインスタンス名が記録され，同一の検索クエリ中で使用されている“$～”にはそのインスタンス名が割り当てられる．同様に，検索クエリ中に“$～．‥‥”と入力した場合，コードブロック中のメソッド呼び出しだけを対象に検索を行い，検索が一致した場合はそのメソッド呼び出しをしているインスタンス名が割り当てられる．このトークンを利用することで，メソッドを呼び出しているインスタンスの違いを考慮して検索を行うことができる．図 6 に検索クエリの入力例を示す．

3.5　実　　装

本研究では提案手法を EclipsePlugin として実装した．

3.5.1　EclipseJDT による構文解析

本研究では Java ソースコードの構文解析を行うために，Eclipse Java Development Tools (EclipseJDT) を用いた．EclipseJDT とは，Eclipse Foundation が提供する Java 開発のためのソフトウェア群である．EclipseJDT の機能により，Java ソースコードから抽象構文木を生成した．抽象構文木の各部分木は，Java プログラムを構成する構文構造に分類される．本研究では全ての構文構造を扱うのではなく，一部の構文構造のみを扱う．表 1 に今回扱った構文構造を記載する．本実装では，これらの内から 1 つの構文構造を指定して，その構文構造に該当する部分木をコードブロックとして抽出する．図 7 にコードブロックの抽出例を示す．

3.5.2　検索クエリの入力画面

図 8 に本手法で用いる検索ウィンドウを示す．ウィンドウ上部のテキストエリアに検索クエリを入力する．ウィンドウ中の検索グループと記載されたエリアで，検索したいコードブロックの種類を選択することができる．検索したい種類のラジオボタンを選択することで，検索対象の構文構造を設定できる．なお，最下部のドロップダウンリストで，事前に行われた検索に

表 1　本研究で扱う構文構造

構文要素	説明
CompilationUnit	Java ファイル全体
TypeDeclaration	クラス宣言
MethodDeclaration	メソッド宣言
IfStatement	if 文 (else 文を含む)
TryStatement	try{}
CatchClause	catch{}
ForStatement	for 文
WhileStatement	while 文

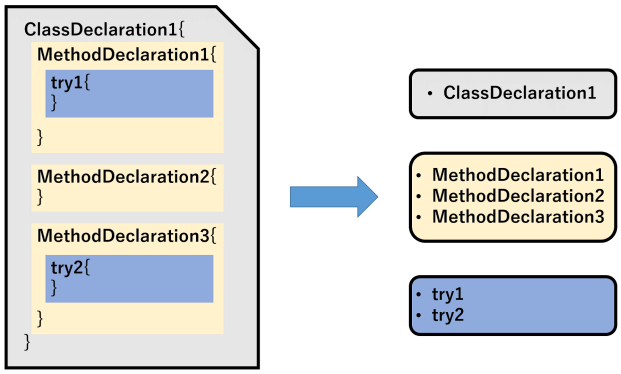

図 7　構文構造ごとにおけるコードブロックの抽出

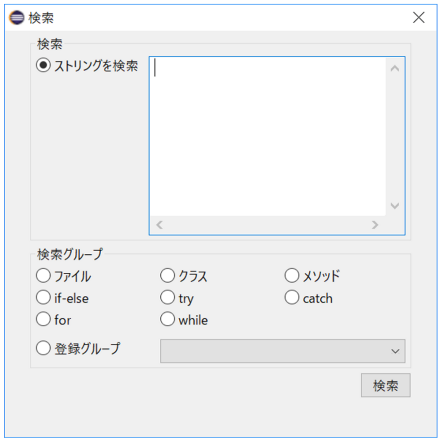

図 8　実装した検索ウィンドウ

よって出力されたグループを検索対象に選択することができる．

3.5.3　検索結果の出力

提案手法の出力は，検索クエリにより分別された，コードブロックのグループである．出力は以下の要素から構成される．

- 各グループの名前
- 分別前のコードブロック数に対する，各グループ内のコードブロック数の割合
- 各グループ内のコードブロックのリスト

各グループの名前は，グループに含まれるコードブロックが検索クエリ中のどの検索ワードを検出したかを示す．リスト中のコードブロックの表記をクリックすることでソースコード中の該当箇所を参照することができる．出力例を図 9 に示す．

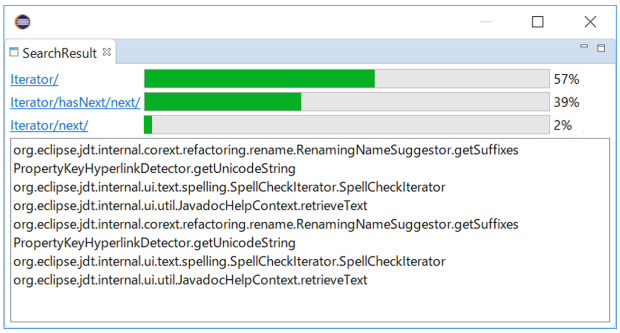

図 9　検索結果の出力例

4.　ケーススタディ

本手法を用いて，ソースコードから類似するコーディングパターンを形成するコードブロックを，分別して提示することが可能であるかを確認するため，ケーススタディを行う．ケーススタディのシナリオを次のように定める．ある API の利用例を表すコーディングパターンが 1 つ判明している状態で．そのコーディングパターンと，それに類似するコーディングパターンの利用状況を理解するために，本手法をいくつかのオープンソースソフトウェアに対し適用する．

4.1　ケース 1 :ResultSet クラス

```
rs = stmt.executeQuery();
if(rs.next()){
   ~=rs.getString(1);
}
rs.close();
```

図 10　ResultSet クラスに関するコーディングパターン

竹之内らは既存研究において，java.sql.ResultSet クラスの利用法の調査を行い，java ソースコードから ResultSet クラスを利用しているコード片を抽出した [5]．対象は velocity-1.6.4, ant-1.8.4, tomcat-7.0.2 , webmail-0.7.10, struts-2.2.1, roller-4.0.1 の 6 つのオープンソースソフトウェアである．竹之内らは抽出したコード片を手作業で確認することで，ResultSet クラスの典型的な利用例を表すコーディングパターンを提示した．その例を図 10 に示す．この結果を利用してケーススタディを行う．図 10 に示すコーディングパターンが既知であるとし，上記のオープンソースソフトウェアを対象に本手法を適用する．

検索クエリに使用する検索ワードを設定する．与えられたコーディングパターンから，ResultSet クラスのインスタンスを生成するメソッドが executeQuery であることが判明している．コーディングパターンを形成するコードブロックはインスタンスを生成する文が含まれていると仮定し，検索ワードの一つとして executeQuery を用いる．この際に-e オプションを記述することで，この検索ワードが含まれていないコードブロックが検索の対象外となり，関係の無いコードブロックを除外することができる．また，トークンを用いた代入文検索の形で記

```
$a=executeQuery;-e
$a.next;
$a.getString;
$a.close;
```

図 11　記述した検索クエリ

述することで，そのコードブロックで使用されているインスタンス名をトークンに割り当てることができる．残りの検索ワードは，ResultSet クラスのメソッド呼び出しをトークンを用いて記述したものとする．記述した検索クエリを図 11 に示す．

この検索クエリによる検索結果を図 12 に示す．各グループ内のコードブロックを確認することで，類似するコーディングパターンを形成するコードブロックが正しく分別されているか調査した．図 12 中の 4 つのグループを確認すると，getString の記述がないコードブロックが少数派であることがわかる．該当するコードブロックのソースコードを確認したところ，図 10 における getString の記述を getInt，getBoolean に置き換えた利用例を確認することができた．一方で，getString の記述がある多数派のコードブロックは，close メソッドの有無により大きく二分されていることがわかる．このことから，図 10 におけるコーディングパターンだけでなく，close メソッドを用いない運用も同等に使用されていると判断することができる．ただし，これらのパターンが同一ソフトウェア中に混在することはなく，close メソッドを含むパターンのほとんどが tomcat，含まないパターンのほとんどが roller から検出されたものであることもわかった．このことから，ResultSet クラスはそれぞれのソフトウェア内においては一貫した形で利用されていると判断することができる．以上の結果から，類似するコーディングパターンを含むコードブロックを分別して提示し，それぞれがどの程度の頻度で使用されているか，どのソフトウェアで使用されているかという情報は，既存手法の結果からは確認が難しかったが，本手法を用いることで確認することが可能であった．

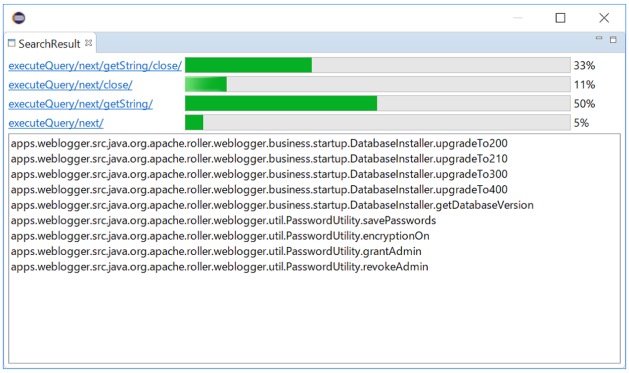

図 12　記述したクエリによる検索結果

5. まとめと今後の課題

本研究では，類似するコーディングパターンの利用状況を提示するための検索手法を提案し，実装を行った．複数の検索ワードを記述する検索クエリを用いて各コードブロックに対し検索を行い，各検索ワードの検出結果によりコードブロックを分別して提示した．

ケーススタディでは，提案手法を用いて ResultSet クラスに関するコーディングパターンの利用状況を調査し，類似するコーディングパターンの分別が行えるかを確認した．その結果，従来では調査することが難しかった類似するコーディングパターンを分別できたとともに，それらのコーディングパターンがどの程度の割合で使用されているかなどの情報を提示することができた．

今後の課題として次の点が挙げられる．まず，検索ワードの出現順序や制御構造などの構文的要素を考慮できないため，検索クエリを拡張して対応する必要がある．また，コーディングパターンの利用状況として，利用頻度だけでなく別の有用な情報量が存在しないか検討する必要がある．

文　献

[1] 中村勇太，崔　恩瀞，吉田則祐，春名修介，井上克郎，“パターンマイニング技術を用いた c 言語プログラムからのコーディングパターン抽出，” 電子情報通信学会技術研究報告，vol.115，no.20，pp.41–46，2015.

[2] B.S. Baker, “A program for identifying duplicated code,” Computing Science and Statistics, vol.6, pp.49–57, 1992.

[3] R. Agrawal and R. Srikant, “Mining sequential patterns,” Proc. 11th International Conference on Data Engineering, pp.3–14, 1995.

[4] 石尾　隆，伊達浩典，三宅達也，井上克郎，“シーケンシャルパターンマイニングを用いた コーディングパターン抽出，” 情報処理学会論文誌，vol.50，no.2，pp.860–871，2009.

[5] 竹之内啓太，石尾　隆，井上克郎，“プログラミング言語の構造を考慮した api 利用例検索ツール，” ソフトウェア工学の基礎 23，pp.23–32，2016.

[6] 後藤　祥，吉田則裕，井岡正和，井上克郎，“差分を含む類似メソッドの集約支援ツール，” 情報処理学会論文誌，vol.54，no.54，pp.922–932，2013.

[7] M. Marin, “Reasoning about assessing and improving the seed quality of a generative aspect mining technique,” Proc. 2nd International Linking Aspect Technology and Evolution Workshop, pp.23–27, 2006.

[8] 肥後芳樹，楠本真二，井上克郎，“コードクローン検出とその関連技術，” 電子情報通信学会 論文誌，vol.J91-D，no.6，pp.1465–1481，2008.

[9] 肥後芳樹，神谷年洋，楠本真二，井上克郎，“コードクローンを対象としたリファクタリング支援環境，” 電子情報通信学会論文誌，vol.J88-D-I，no.2，pp.186–195，2005.

[10] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A multi-linguistic token-based code clone detection system for large scale source code,” IEEE Transactions on Software Engineering, vol.28, no.7, pp.654–670, 2004.

[11] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, “Deckard: Scalable and accurate tree-based detection of code clones,” Proc. 29th Intenational Conference on Software Engineering, pp.96–105, 2007.

[12] J. Krinke, “Identifying similar code with program dependence graphs,” Proc. 8th Working Conference on Reverse Engineering, pp.301–309, 2001.

[13] M. Bruntink, A. vanDeursen, R. vanEngelen, and T. Tourw’e, “On the use of clone detection for identifying crosscutting concern code,” IEEE Trans. Softw. Eng, vol.31, no.10, pp.804–818, 2005.