特別研究報告

題目

コーディングパターンのあいまい検索の提案と実装

指導教員
井上 克郎 教授

報告者
小笠原 康貴

平成 30 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 29 年度 特別研究報告

コーディングパターンのあいまい検索の提案と実装

小笠原 康貴

内容梗概

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．そのためソフトウェア開発において，ソフトウェア中の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，ソフトウェア資源の大規模複雑化に伴いコーディングパターンは多様化される傾向にあり，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．また同一ソフトウェア中に類似したコーディングパターンが把握されずに混在する状況は，実装上のルールを不明瞭にし，ソフトウェアの保守に悪影響を与える．そのため開発者にとって，ソフトウェア中の類似するコーディングパターンの運用状況を理解できることが好ましい．

コーディングパターンに関する既存研究は多数存在するが，その多くは定められた条件に一致するコーディングパターンを検出することに特化しており，その結果にコーディングパターンの運用を判断するために必要な情報が十分でない可能性があった．

そこで本研究では，検索クエリに一致するコードブロックだけでなく，検索クエリに一致しないコードブロックをグループ化して出力する検索を提案する．以降この検索をあいまい検索と呼称する．あいまい検索によるコードブロックのグループ化により，コーディングパターンの運用状況を提示するため，Java ソースコードの静的解析によるあいまい検索を実装した．ケーススタディでは幾つかのオープンソースソフトウェアに対しあいまい検索を実行し，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，従来は確認することが難しかった類似するコーディングパターンの運用状況を確認することができた．

主な用語

コーディングパターン
ソフトウェア開発

検索
静的解析

目 次

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処理のことである．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．1 つは，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 [14]．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のっと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，実装上のルールを不明瞭となり，ソフトウェアの保守に悪影響を与える [3]．また，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っている [14]．そのため開発者はソフトウェア中の類似するコーディングパターンの運用状況を理解できることが好ましい．

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 [2] 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 [15]．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 [10]．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を区別せずに検出す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判断する必要がある．しかし，そのためには開発者がソースコードを一つ一つ確認するしかなく，従来のソースコード検索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，開発者に類似するコーディングパターンの運用状況を提示するためのソースコード検索を提案する．検索クエリに一致するソースコードを提示する従来の検索に対して，提案手法では検索クエリに一致するコードブロックと検索クエリに一致しなかったコードブロックを提示する．これにより，従来では把握することが困難であった類似するコーディングパターンを分別して提示することができ，類似するコーディングパターンの運用状況の理解が容易となる．以降本研究では，この検索手法を従来のソースコード検索と区別してあいまい検索と呼称する．

あいまい検索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運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あいまい検索を実装した．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あいまい検索を実行す

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できた．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運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 章では評価の結果を説明し，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 [5][11]．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 [9]．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 の利用法が挙げられる．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，一種のコードクローンを形成している場合が多い [12]．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 [13]．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する作業を行わなければならず，必要としているパターンを発見することが困難となっている．

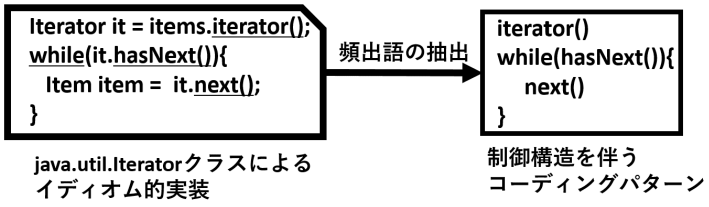

図 1: 制御構造を伴うコーディングパターン

2.2 関連研究

2.2.1 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 [2]．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．シーケンシャルパターンマイニングを用いてコーディングパターンを検出する研究は既に行われている．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 [15]. 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った．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．

2.2.2 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 [10]．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．竹之内らは特殊なトークンを含む検索クエリを用いた検索を行うことで，検索クエリにマッチするソースコード中のコード片を抽出し，API の利用法を表すコード片の検出を可能とした．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判別しなければならない問題が残っている．

2.2.3 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[7]，抽象構文木の比較による検出を行う Jiang らの手法 [6]，プログラム依存グラフを比較する Krinke の手法 [8] 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，検出の条件に，一致するコード片のサイ

ズが一定以上であることが含まれる．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 [4]．

3　提案手法

本研究では，類似するコーディングパターンの運用状況を提示するためのソースコード検索を提案する．提案手法の入力はソースコードと検索クエリである．検索クエリとして任意の文字列を与え，ソースコードから抽出したコードブロックに対し検索を行う．そして，検索クエリに一致するコードブロックと，検索クエリに一致しない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，検索クエリに一致したコードブロックのグループと，検索クエリに一致しなかったコードブロックのグループである．検索クエリに一致するソースコードを提示する従来のキーワード検索に対し，検索クエリに一致するコードブロックと一致しなかったコードブロックを分別し提示する提案手法を，本研究ではあいまい検索と呼称する．

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。類似したコーディングパターンを分別して検出するためには、まずソースコード全体から共有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。あいまい検索を用いることで，この処理を実現することができる．

以降 3.1 節で用語の説明を行い，3.2 節であいまい検索の具体的な解説を行う．その後 3.3 節にて，あいまい検索をどのように用いてコーディングパターンの運用状況の提示を実現するかを説明する．

3.1　用語の説明

3.1.1　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，ソースコード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 2 に抽象構文木の例を示す.

3.1.2　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のっとり，中括弧で閉じられた部分をコードブロックとして扱う.

3.2　あいまい検索の手順

提案手法は以下の手順で構成される．以降の節で各手順について説明を行う.

1. ソースコードの構文解析を行い，コードブロックを抽出

2. 抽出したコードブロックに対して検索を実行

3. 検索に一致するグループと，検索に一致しないグループにコードブロックを分別

3.2.1 手順１:コードブロックの抽出

手順１ではソースコードからコードブロックの抽出を行う．まず，ソースコードに対して構文解析を行い，抽象構文木を生成する．このとき，抽象構文木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３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抽出されている．

3.2.2 手順２:コードブロックの検索

手順２では検索によるコードブロックの分別を行う．検索の対象はソースコードから抽出されたコードブロックである．検索は各コードブロックに対して行う．つまり，複数のコードブロックが抽出されている場合，検索はそれぞれのコードブロックに対して一回ずつ実行される．

3.2.3 手順３:コードブロックの分別

各コードブロックの検索では，コードブロック内に検索クエリが存在すれば検索の一致，存在しなければ検索の不一致として結果を判定する．これにより，各コードブロックを検索に一致するグループ，検索に一致しないグループに分別する．これら二種類のグループ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４に示す．

3.3 提案手法の適用方法

この節では提案手法の適用方法について述べる．提案手法は，目的とする類似するコーディングパターンの分別を実現するために，利用者による複数回の検索の実行を想定したものである．手順２の入力として、手順３で出力されたコードブロックの士グループのうち１つを与えることで、提案手法の出力結果をさらに絞り込んで分別することが可能である。図５に提案手法の適用例を記載する．この図に示すように，検索クエリによる分別を繰り返すことで，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を差異の種類ごとに分別してグループ化する．

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
Assignment	代入文
MethodInvocation	メソッド呼び出し
SimpleName	変数名[1]

3.4 実装

本研究では提案手法を EclipsePlugin として実装した．

3.4.1 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 [1]．EclipseJDT の提供機能により，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ている．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った．表 1 に今回扱った構文構造を記載する．

3.4.2 コードブロックの抽出

本研究では，生成した抽象構文木からコードブロックを抽出する必要がある．今回は，特定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実装を行った．表 1 で示す構文構造を抽出することができる．図 6 にコードブロックの抽出例を示した．

[1]本来の SimpleName の定義にはクラス名とメソッド名が含まれるが，本研究では除外している

3.4.3 検索クエリの入力

検索クエリを入力するために，図 7 に示す検索ウィンドウを実装した．画面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グループを選択することができる．検索したいグループのラジオボタンを選択することで，検索の対象を設定できる．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．

3.4.4 検索結果の出力

提案手法の出力は，検索クエリとの一致，不一致で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．

- 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

各コードブロックの表記名は完全限定名とする．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実際の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 8 に示す．

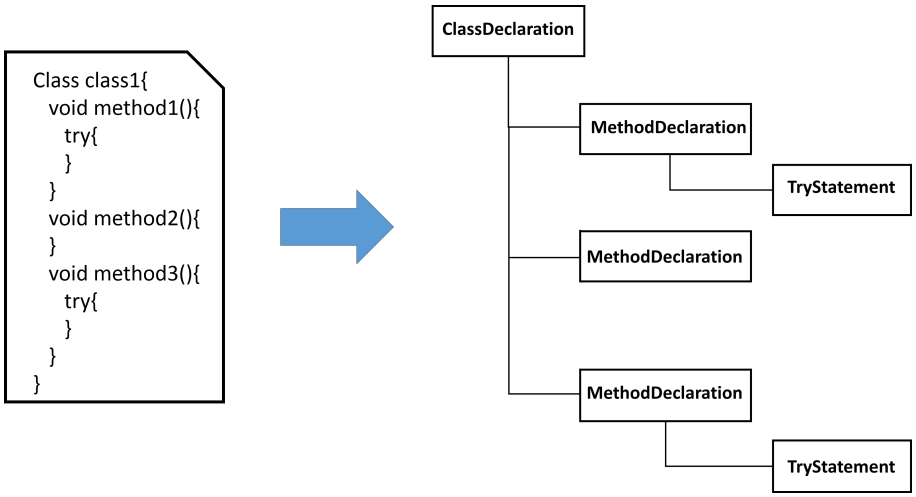

図 2: 抽象構文木の生成例

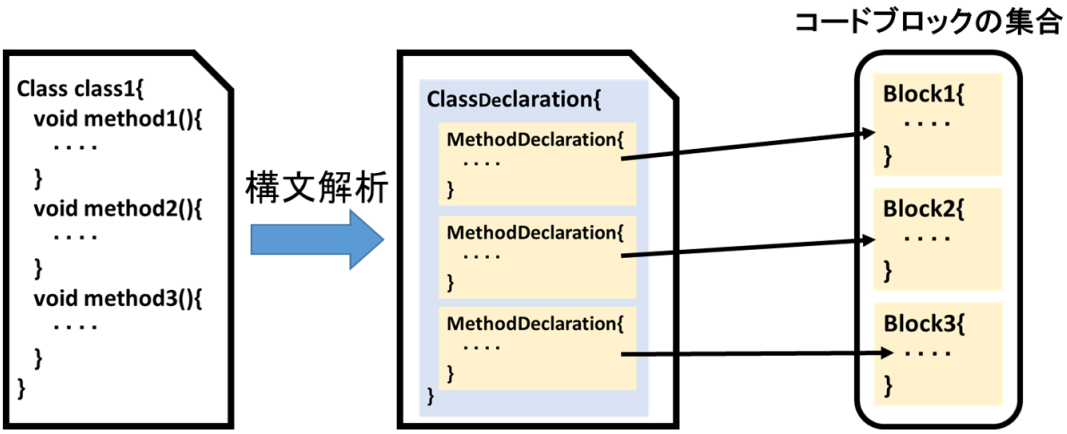

図 3: コードブロックの抽出

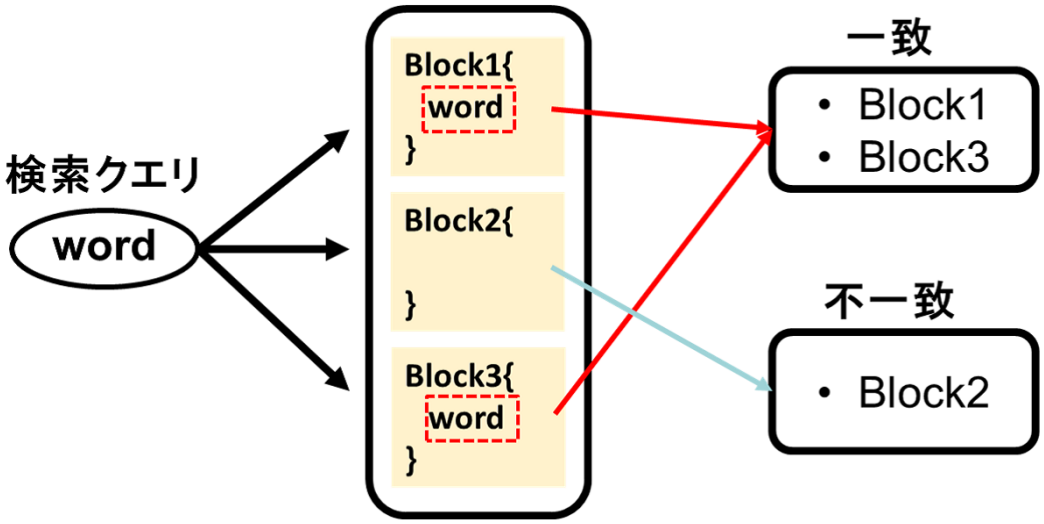

図 4: 検索によるコードブロックの分別

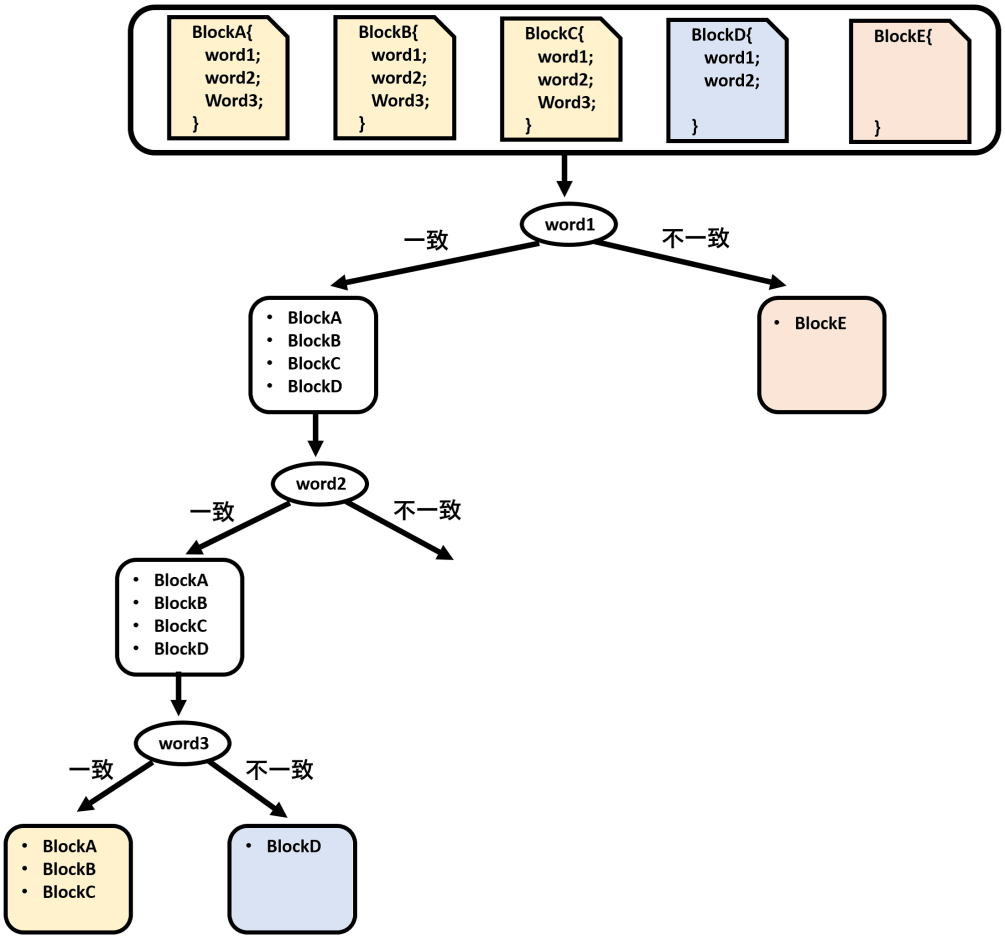

図 5: 提案手法によるコードブロックの分別

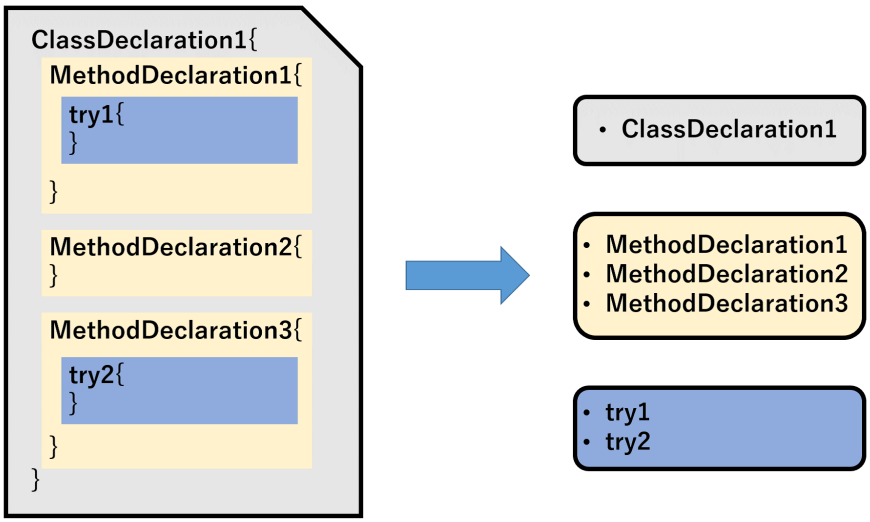

図 6: 構文構造ごとにおけるコードブロックの抽出

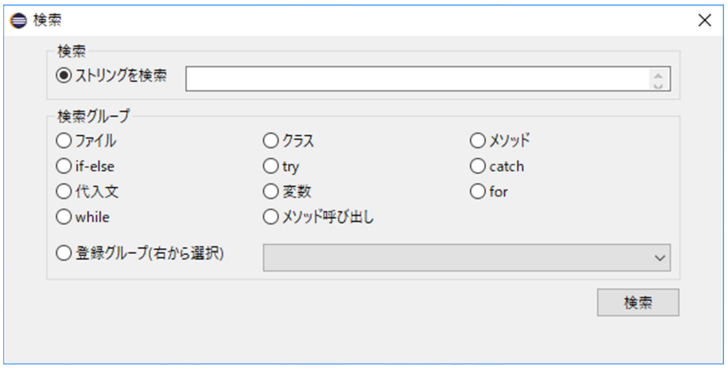

図 7: 実装した検索ウィンドウ

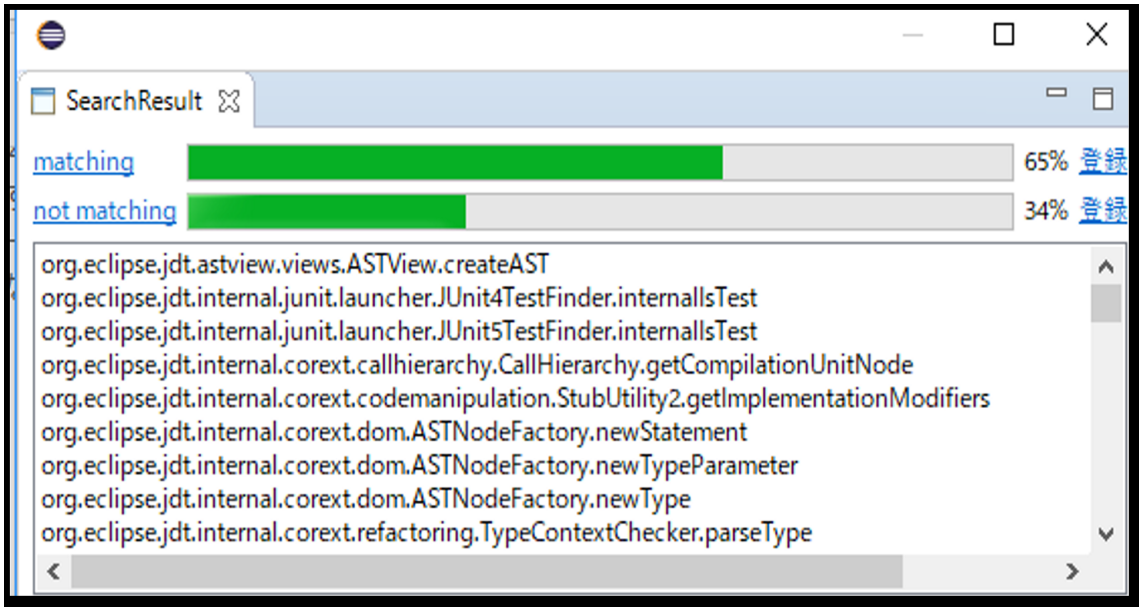
SearchResult
matching
65%
登録
not matching
34%
登録
org.eclipse.jdt.astview.views.ASTView.createAST
org.eclipse.jdt.internal.junit.launcher.JUnit4TestFinder.internalIsTest
org.eclipse.jdt.internal.junit.launcher.JUnit5TestFinder.internalIsTest
org.eclipse.jdt.internal.corext.callhierarchy.CallHierarchy.getCompilationUnitNode
org.eclipse.jdt.internal.corext.codemanipulation.StubUtility2.getImplementationModifiers
org.eclipse.jdt.internal.corext.dom.ASTNodeFactory.newStatement
org.eclipse.jdt.internal.corext.dom.ASTNodeFactory.newTypeParameter
org.eclipse.jdt.internal.corext.dom.ASTNodeFactory.newType
org.eclipse.jdt.internal.corext.refactoring.TypeContextChecker.parseType

図 8: 検索結果の出力例

4　ケーススタディ

あいまい検索を用いて，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った．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 つわかっている状態で．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運用状況を理解するために，あいまい検索を幾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

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 [10]．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のソースコードである．竹之内らは抽出したコード片を目視で確認することで，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 9 に示す．この結果を利用してケーススタディを行う．図 9 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あいまい検索を適用する．

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 であることがわっかている．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，executeQuery を含むメソッド宣言文を最初に抽出する．その後，ResultSet クラスのインスタンスに対するメソッド呼び出しをキーワードにして，順次あいまい検索を行った．図 10 に検索によるコードブロックの分別過程を記載する．図中の角丸はコードブロックのグループを意味しており，その左に記載される数字はグループ内に含まれるコードブロックの数を意味している．

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

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

図 9: ResultSet クラスに関するコーディングパターン

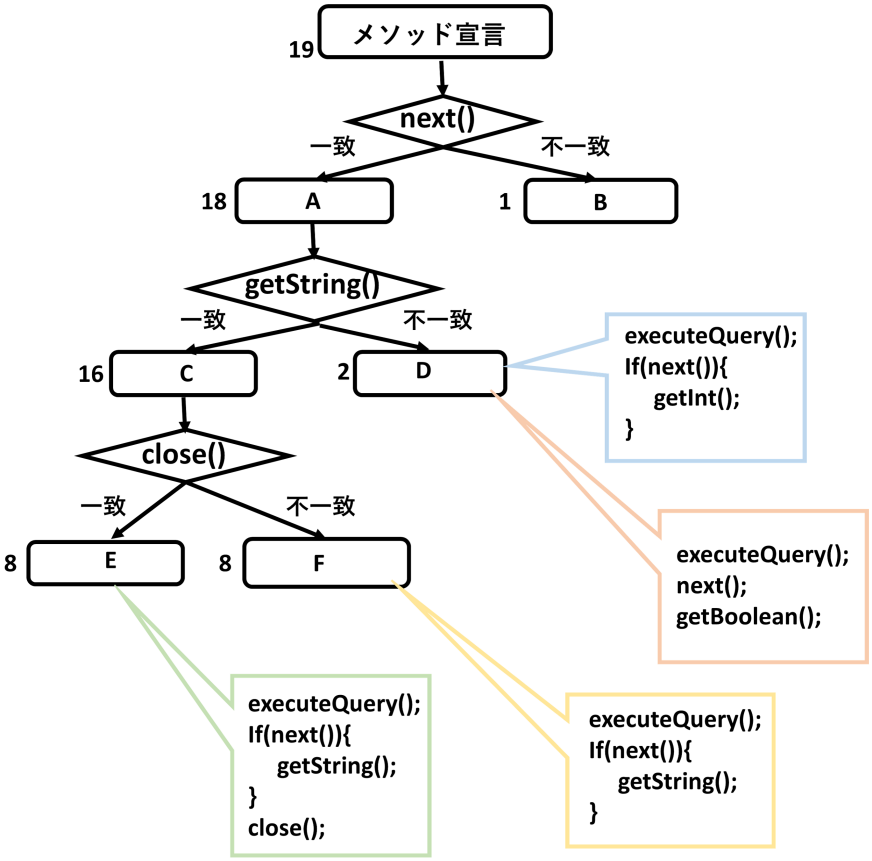

図 10: ケース 1 におけるコードブロックの分別過程

するコードブロックが正しく分別されているか確認した．図 10 中のグループ D において，図 9 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．またグループ E，グループ F の結果より，close メソッドの呼び出しを行わないコーディングパターンが多数存在することがわかった．close メソッドに代替する記述が見当たらないことから，図 9 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はほとんどが tomcat から検出されたものであることもわかった．このように，類似した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．

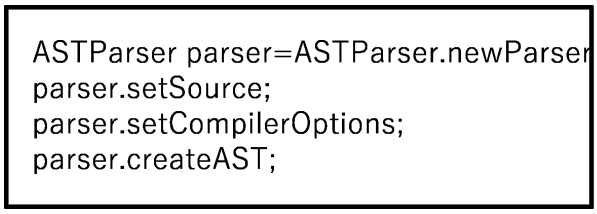

図 11: 公式ドキュメントにおける ASTParser クラスの使用例

4.2 ケース２:ASTParser クラス

EclipseJDT の API に，ASTParser クラスが存在する．ASTParser クラスのインスタンスの使用法として，公式ドキュメントに図 11 のような例が記載されている．しかしながら実際に使用される場合において，このインスタンスに対する操作は多数存在し，その使用法を把握することは困難である．そこで，図 11 に示す利用例を既知のコーディングパターンとし，どのような類似するコーディングパターンがどの程度の頻度で出現するのかの調査を行った．対象のソフトウェアは，Eclipse の UI に関する機能を提供する，eclipse.jdt.ui-master を用いた．

検索の方針を説明する．調査の対象としたコードブロックは，ケース 1 と同様にインスタンス生成文に着目し，ASTParser クラスのインスタンスを生成する newParser メソッドの呼び出しが存在するメソッド宣言とした．検索に用いるキーワードとして，図 11 で示した ASTParser クラスの使用例におけるメソッドを最初に検索に用いた．その後は出力されるグループ内のコードブロックを確認し，既出でない ASTParser クラスのメソッドが存在すれば，そのメソッドをキーワードとして検索を行った．検索を終了するタイミングは，出力されるグループ内のコードブロックに，既出でない ASTParser クラスのメソッドが存在しなくなった時点，またはグループ内のコードブロック数が，目視によるコードブロックの確認が容易な数以下になった時点である．今回は検索終了の条件を 10 個と設定した．検索終了後，グループ内のコードブロックを確認してコーディングパターンを判別し，その結果，表 2 に示すように，コーディングパターンを分別して検出することができた．図 12 に検索によるコードブロックの分別の過程を記載する．これらの分別結果を確認すると，グループ I，M，Q 等に属するコードブロックの数が他と比較して大きくなっていることがわかり，加えて他の多くのグループが，これらのグループが表すコーディングパターンに特定のメソッド呼び出しを追加したものであり，そのグループ内のコードブロック数が少数であることがわかった．これらのことから，グループ I，M，Q に示されるコーディングパターンを ASTParser クラスの典型的なメソッド呼び出しとして推測することができた．さらに，これらのコーディングパターンに対して使用数は少ないものの，複数回の使用が行われているグループ B，C，R のようなコーディングパターンを分別して提示することができた．

4.3 提案手法の問題点

ケーススタディにおいて提案手法の問題点も確認できた．ケース 1 におけるコーディングパターンは制御構造を伴うコーディングパターンであったが，現状の検索クエリの場合，メソッド呼び出しの系列が同様であっても，使用されている制御構造が違う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することができないことがわかった．次に，ケース 2 のようにコーディングパターンを形成しうるメソッド呼び出しの候補が多い場合，コードブロックの分別に必要な検索回数が非常に多くなり煩雑であった．さらに，検索を終了するかの判定を行うために，場合によっては多数のコードブロックを確認する必要があった．また，メソッド呼び出しは一致するがその順番が異なるコードブロックは同一のグループに分別される．必ずしもメソッド呼び出しの順番が同一である必要はないが，コーディングパターンによってメソッド呼び出しの順番が不可逆であることを考えると，キーワードの順番を考慮できる検索クエリが必要である．

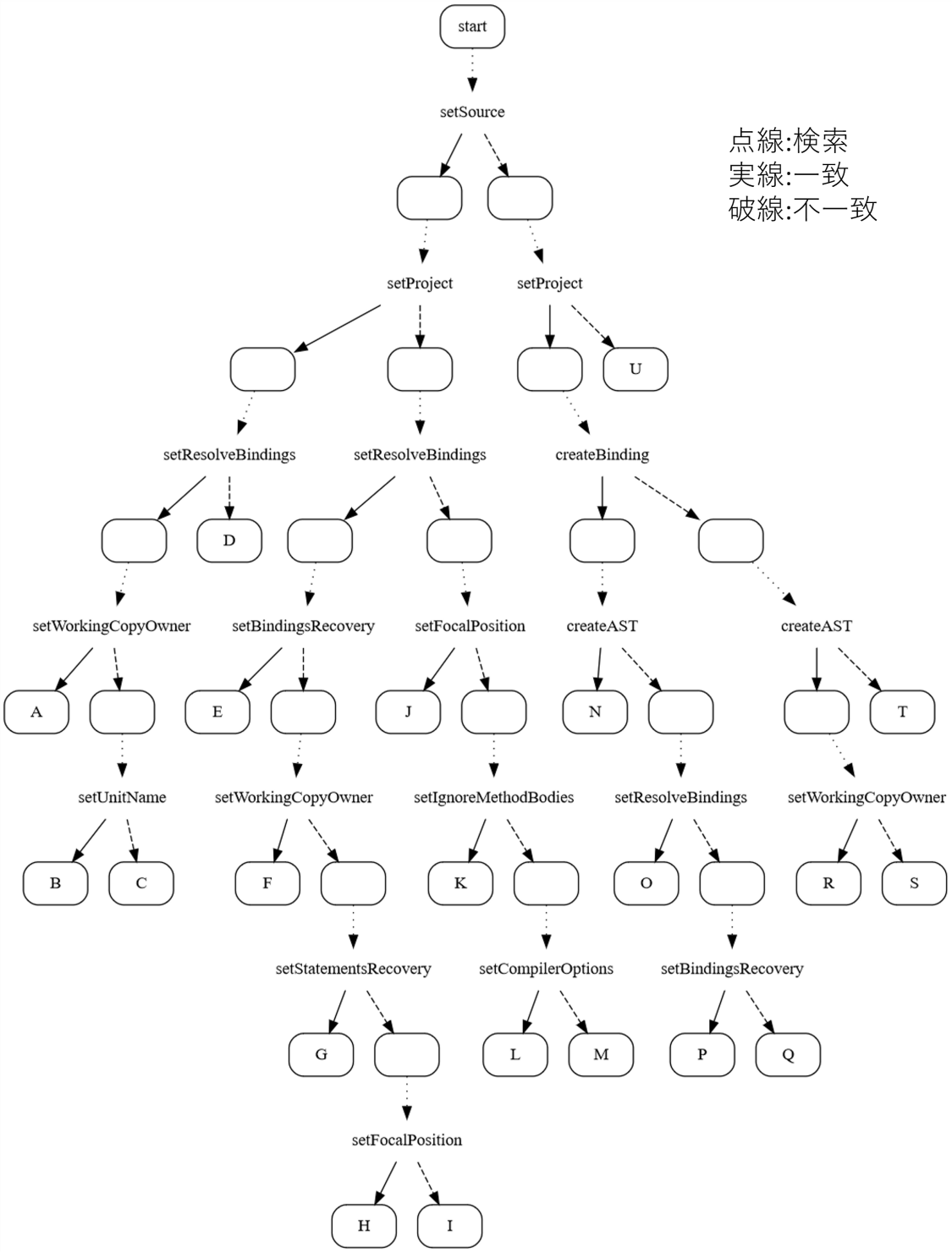

図 12: ケース 2 におけるコードブロックの分別過程

表 2: ケース 2 におけるグループ化したコーディングパターン

パターン	該当グループ	出現回数
setSource/setResolveBindings/setProject/setUnitName/createAST	B	4
setProject/setResolveBindings/setSource/createAST	C	4
setSource/setProject/createAST	D	3
setSource/setResolveBindings/setFocalPosition/createAST	H	2
setResolveBinding/setSource/createAST	I	17
setSource/setFocalPosition/createAST	J	4
setSource/setCompilerOptions/createAST	L	2
setSource/createAST	M	9
setResolveBindings/setProject/createBindings	O	2
setProject/setBindingsRecovery/TRY/createBindings	P	7
setProject/createBindings	Q	7
setWorkingCopyOwner/setResolveBindings/setProject /setCompilerOptions/createASTs	R	4
setResolveBindings/setProject/createASTs	S	2
インスタンス生成のみ	U	4

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運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．提案手法では，ソースコードの構文解析を行うことでコードブロックを抽出する．各コードブロックに対し検索を行い，検索に一致するコードブロックと，検索に一致しないコードブロックの双方を結果として分別して提示する．また，この検索手法を反復して実行することで，コーディングパターンを形成する単語の違いをもとに類似するコーディングパターンを分別し，それぞれをグループ化して提示する利用ができることを示した．

ケーススタディでは，２種類のコーディングパターンに対して提案手法を実行し，提案手法により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，従来では確認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頻度で出現し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，コーディングパターンを形成しうるキーワードの数が多い場合に，検索による分別が煩雑になるという問題点の解決が挙げられる．方針として，複数のキーワードを検索クエリに記述し，今まで自身で行っていた検索の反復行程を一度の検索で実行できるように変更することが考えられる．また，検索クエリの対応能力に関して幾つか問題があることがわかったが，竹之内らの既存研究に示されているようなパターンマッチを取り入れることで改善を行える．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授には，研究活動においてたくさんの貴重な御意見及び御指導を賜りました．井上 教授の御監督のおかげで，研究活動に適切に取り組むことができました．井上 教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には，研究活動においてたくさんの貴重な御意見及び御指導を賜りました．松下 准教授の適切な御指摘のおかげで，自身の研究の問題点を把握し，改善に繋げることができました．松下 准教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科 春名 修介 特任教授には，研究活動において適切な御指摘を賜りました．いただいた御指摘のおかげで，自身の研究発表を改善することができました．春名 特任教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科 神田 哲也 特任助教には，研究の方針から本論文の執筆に至るまで，貴重な直接の御指導及び御助言を賜りました．神田 特任助教の親身な担当指導のおかげで本論文を完成させることができました，心より深く感謝いたします．

奈良先端科学技術大学院大学情報科学研究科 石尾 隆 准教授には，研究に関する貴重な御意見及びご助言を賜りました．いただいた御助言により，研究に関する見識を深めることができました．石尾 准教授に心より深く感謝いたします．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上研究室の先輩方におきましては，たくさんの研究に関する御助言をいただきました．特に，大阪大学大学院情報科学研究科コンピュータサイエンス専攻 嶋利 一真 氏には，本論文の添削を行っていただくなど，様々な場面でご協力していただきました．先輩方の御助言のおかげで，研究活動を滞りなく続けることができました，心より深く感謝いたします．

最後に，自身の研究活動を支えてくださった，大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に，心より深く感謝いたします．

参考文献

[1] Eclipse java development tools (jdt). *https://www.eclipse.org/jdt/*.

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

[3] B.S. Baker. A program for identifying duplicated code. *Computing Science and Statistics*, Vol. 6, pp. 49–57, 1992.

[4] M . Bruntink, A. van Deursen, R . van Engelen, 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.

[5] M . Fowler. improving the design of existing code. *Addison Wesley*, 1999.

[6] L . Jiang, G. Misherghi, Z Su, S. Glondu. Deckard: Scalable and accurate tree-based detec- tion of code clones. *Proc. 29th Intenational Confer- ence on Software Engineering*, pp. 96–105, 2007.

[7] 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.

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

[9] 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*, 2006.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.