Vol.2017-SE-197 No.3
2017/11/9

開発作業のモニタリングによるコードクローン集約支援環境の構築

沼田 聖也[1,a)]　吉田 則裕[2]　崔 恩瀞[3]　井上 克郎[1]

概要：コードクローンとは，ソースコード中に存在する同一，あるいは類似したコード片を意味する．また，ソースコード中に存在するコードクローンの集合を単一の関数にまとめることをコードクローンの集約という．本研究は，Eclipse 上で行われた開発作業を分析することで，コードクローンの集約を支援する環境を構築する．具体的には，開発作業からメソッド抽出の集約パターンを見つけ，その抽出するコード片のコードクローンを開発者に提示し，同時に集約を検討することを促す．本環境により，保守性の高いソースコード開発が期待できる．

キーワード：コードクローン，コードクローンの集約，メソッド抽出，モニタリング

Development of a Code Clone Refactoring Environment by Monitoring Development Tasks

Seiya Numata[1,a)]　Norihiro Yoshida[2]　Eunjong Choi[3]　Katsuro Inoue[1]

1. まえがき

ソフトウェア開発にかかるコストを増大させている要因の 1 つとして，ソースコード中のコードクローンが挙げられる．コードクローンとは，ソースコード中に存在する同一，あるいは類似した部分を持つコード片を意味し，主に既存のコード片のコピーアンドペーストによって生成される [1][2][6]．例えば，あるコード片に欠陥が見つかった場合，そのコード片のコードクローンにも同様の欠陥が含まれる可能性が高い [14]．そこで，開発者はコードクローンに欠陥が見つかった場合，同一クローンセット（互いにコードクローンとなっているコード片の集合）内に含まれる全てのコードクローン対して同様の修正を検討する必要があるが，すべてのコードクローンを開発者が手作業で見つけて管理するのは困難である．そのため，開発者はコードクローン管理作業のためにコードクローン検出ツールを利用する．現在までに，ソースコードからコードクローンを自動検出する手法は数多く提案されている [8]．

コードクローンに対する開発コストを削減する方法の 1 つとして，コードクローンのリファクタリングが挙げられる [7][9]．コードクローンに対するリファクタリングに，コードクローンの集約がある．コードクローンの集約とは，コードクローンの集合を単一の関数にまとめることである．コードクローンの集約を適切に行うことで，コードクローンの修正にかかるコストを削減することができる．

これまでに，ソースコード中からリファクタリング対象のコードクローンを自動抽出することで，コードクローンを対象としたリファクタリング支援を行う手法が提案されている [3][18][17]．リファクタリング支援手法には，優れたリファクタリング候補を提示できるだけでなく，開発者の作業内容に応じて適切な時期に支援を行うことが求められる．その理由は，開発者の作業内容に関連したリファクタリングを推薦すると，作業内容に関する記憶を参照しながら効率的にリファクタリングできるからである．また，テストが完了し，別作業を開始した段階でリファクタリング

[1] 大阪大学大学
[2] 名古屋大学
[3] 奈良先端科学技術大学院大学
[a)] s-numata@ist.osaka-u.ac.jp

推薦しても，変更作業に関する記憶が曖昧になっており，かつ変更内容の妥当性を確認するためのテストをやり直す必要があるため，効率的なリファクタリングにかかるコストが大きくなってしまうという問題点も挙げられる [15][16]．しかし，我々の知る限り，今まで提案された手法では，開発者の作業内容を考慮せずに，コードクローンのリファクタリング支援を行っている．

この問題を解決するためには，開発者の作業内容に応じて適切な時期にコードクローンのリファクタリングを支援することが必要である．本研究では，メソッド抽出リファクタリングに着目し，開発者の作業内容に応じて，適切な時期にコードクローンのリファクタリングを支援する環境を構築する．具体的には，統合開発環境 Eclipse 上での開発者のソースコードの編集作業をモニタリングする．そして，モニタリング中に開発者がメソッド抽出リファクタリング [7] を行ったことを検知すると，そのメソッド抽出リファクタリングを行ったコード片のコードクローンを開発者に提示し，同時に集約を検討することを促す．これにより，開発者はその場でコードクローンの存在を知り，提示されたコードクローンに対して集約作業を検討することができる．本論文では，構築する支援環境の説明を行い，構築環境の利用例を用いながらその利用シナリオを説明する．

以降，2 章では，本研究に関わるコードクローンおよびリファクタリングの関連研究について説明する．3 章では，メソッド抽出の集約パターンを検出し，そのコードクローンを開発者に提示する支援環境を提案する．4 章では，本環境の使い方を利用例を用いて説明する．5 章では，まとめと今後の課題について説明する．

2. 関連技術

2.1 コードクローン検出ツール CCFinderX

現在まで，関数単位や行単位等，様々な粒度に基づいてコードクローンを検出するツールが開発されてきた [8]．CCFinder はその中でも，字句単位のコードクローンを検出するツールであり，CCFinderX はその CCFinder を拡張したものである [10]．字句単位の検出では，プログラムの字句解析を行った後，その字句を要素とした系列に対してコードクローンを検出する．字句解析を行うことにより，空白やコメントを無視することができる．また，識別子や定数等の特定の種類の字句を特殊な 1 つの字句に固定することで，変数名や関数名の変更されたコード片もコードクローンとして検出することができる．CCFinderX はこの方法により，タイプ 1(空白の有無，レイアウト，コメントの有無等の違いを除き完全に一致する) コードクローンと，タイプ 2(タイプ 1 の違いに加えて，変数名等のユーザー定義名，関数の型等が異なる) コードクローンを検出可能である．CCFinderX はコードクローンを高速で検出することができ，多くの企業や研究で使用されている．

```
void printOwing(double amount){
    printBanner();

    //明細の表示
    System.out.println("name:" + _name);
    System.out.println("amount:" + amount);
}
```

```
void printOwing(double amount){
    printBanner();
    printDetails(amount);
}

void printDetails(double amount){
    System.out.println("name:" + _name);
    System.out.println("amount:" + amount);
}
```

図 1　メソッド抽出リファクタリングの例 [5]

2.2 コードクローン変更管理システム Clone Notifier

Clone Notifier は，版管理システムの 2 つのバージョン間のコードクローンの変更情報を得ることができるツールである [15][16]．コードクローンの変更情報とは，前のバージョンではコードクローンでなかったものが，新しいバージョンではコードクローンになったり，逆に，前のバージョンではコードクローンであったものが，新しいバージョンではコードクローンでは無くなったといった，2 つのバージョン間のコードクローンがどう変更されたかに関する情報である．このコードクローンの変更情報を得ることにより、開発者はコードクローンに対して同時修正を検討することができ，ソフトウェア開発の保守性を高めることができる．しかし，Clone Notifier は版管理システムを介して，コードクローンの変更情報を分析する．そのため，作業の後戻りが必要になる可能性が高くなる．

2.3 メソッド抽出リファクタリング

リファクタリングには様々な種類がある [5]．本研究ではその内，メソッド抽出リファクタリングに着目する．メソッド抽出リファクタリングとは，ひとまとめにできるコード片を新たなメソッドとして定義し，抽出されたコード片を抽出先のメソッドへの呼び出し文に置き換えるリファクタリングである．本論文では，メソッド抽出リファクタリングのことを単にメソッド抽出と呼ぶこととする．メソッド抽出の主な目的は，長すぎるメソッドの分割や，複数の機能が実装されたメソッドの分割である．メソッドを適切なサイズに分割することで可読性を向上させることができ，さらに，機能追加やバグ修正が容易になるという利点が挙げられる．メソッド抽出は，開発者が頻繁に行うリファクタリングの 1 つであることが分かっている [11]．メソッド抽出の例を図 1 に示す．図 1 では，printOwing メソッドの明細表示部分のコード片を，新たに printDetails というメソッドとして定義し，メソッド呼び出ししている．

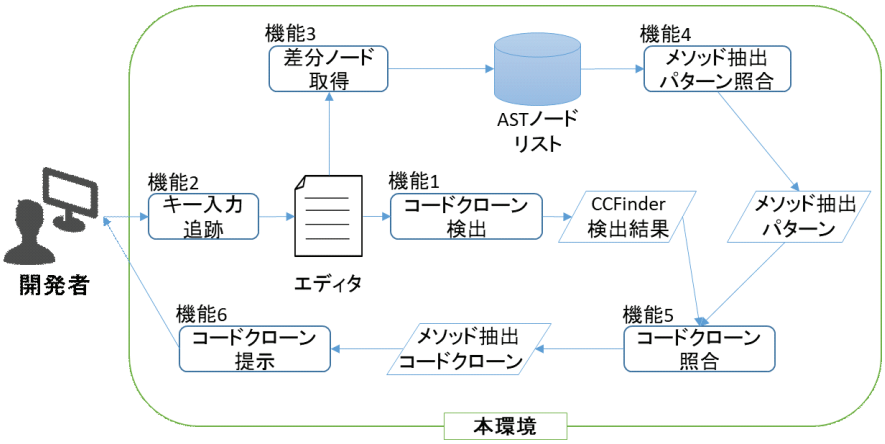

図 2　提案する環境の概要

2.4　リファクタリング支援ツール WitchDoctor

WitchDoctor とは，手動で行われているリファクタリングを進行中に検出し，自動でリファクタリングを完遂するツールである [4]．WitchDoctor はすべてのキー入力を追い，キー入力が起こるたびにプログラム行差分を取得する．そして，差分片と AST(抽象構文木) を対応付けし，差分の AST ノードを取得する．差分の AST ノードを蓄積していき，開発者の進行中の操作が特定のリファクタリング操作パターンと一致しているかどうかを検知する．最終的に，リファクタリング操作を検知すると，進行中の操作を元に戻し，自動でリファクタリングを適用する．本研究では，メソッド抽出の集約パターンを検出する際に，WitchDoctor の手法を参考にした．

3.　提案する環境

開発作業のモニタリングによりメソッド抽出の集約パターンを検知し，その抽出したコード片のコードクローンを開発者に提示する支援環境について説明する．本環境の概要図を図 2 に示す．図 2 に示す通り，本環境は機能 1〜6 を有している．それぞれの機能の詳細は，3.1 節から 3.6 節で述べる．

3.1　コードクローン検出

メソッド抽出を行ったコード片のコードクローンを検出するために CCFinderX を利用する．CCFinderX は，多くの研究や企業で使用実績があり，高速にコードクローンを高い精度で検出するため，本研究でも利用することにした．メソッド抽出を行ったコード片のコードクローンを検出するためには，メソッド抽出が行われる前のソースコードに対してコードクローンを検出する必要がある．そこで，本環境では，作業内容のモニタリングを開始する際に CCFinderX によるコードクローン検出を行い，その後は，開発者の好きなタイミングでコードクローン検出を行えるようにしている．今後は，このコードクローン検出に対しても，適切なタイミングを考え，自動化できるようにしようと考えている．

3.2　キー入力追跡

開発作業のモニタリングを行うために，本環境では，開発者のキー入力の追跡を行う．メソッド抽出を行う際に必要な情報は，1 行以上の行差分情報である．そこで，キー入力を追跡し，1 行以上の変更を検知できるようにした．行差分を取る際には，Myers の差分検出アルゴリズムを適用した [12]．

3.3　差分ノード取得

ソースコードに対して 1 行以上の変更があった場合，その差分を AST のノードとマッピングし，取得する．AST の構築には Eclipse の ASTParser[13] を使用する．差分は組 (delta_type, ast_node) の集合から構成される．delta_type が取る値は Insert あるいは Delete のいずれかである．Insert は 1 行以上の行が挿入したことを表し，Delete は 1 行以上の行が削除されたことを表す．ASTParser を利用して作った AST は，ASTVisitor クラスを実装することによってたどることができ，これにより，AST からノードを取得することができる．そこで，diff アルゴリズムによって取得した行差分と一致する AST のノードを AST から取得する．ast_node はその取得した AST のノードの情報から成り，ノードの名前，ノードの種類，ノードの位置，ノードの中身等の情報が含まれている．ここでノードの種類とは，MethodDeclaration(メソッド宣言) や MethodInvocation(メソッド呼び出し), VariableDeclarationStatement(変数宣言文) 等の情報のことである．後のメソッド抽出の集約パターンの照合の節で説明するが，メソッド抽出の集約パターンを検出するためには，(Insert, MethodDeclaration) と (Insert，MethodInvocation)，(Delete，code_block) の 3 つの情報が必要であるため，本研究では，この 3 種類の AST ノードのみを取得して，AST ノードリストとして保

存するようにする．

3.4 メソッド抽出の集約パターン照合

差分ノードを取得すると，蓄積した差分ノードがメソッド抽出の集約パターンと一致するかを照合する．本環境では，以下を満たす差分ノードが検出された時，メソッド抽出が行われたと判断する．

$$(Delete, code\_block)$$
$$\wedge(Insert, MethodDeclaration)$$
$$\wedge(Insert, MethodInvocation)$$

$where$

$$position(code\_block) = position(MethodInvocation)$$
$$\wedge\ name(MethodDeclaration)$$
$$= name(MethodInvocation)$$

あるコード片が削除され，新しいメソッドが宣言され，そしてコード片が削除された位置に新しいメソッドの呼び出し文が追加された際にメソッド抽出が行われたと判断する．上式を見てみると，1 行目がコード片が削除されたことを表し，2 行目が新しいメソッド宣言が行われたことを表し，3 行目が新しくメソッド呼び出し文が挿入されたことを表す．しかし，それだけではメソッド抽出が行われたと判断することはできないため，さらに条件を設け，4 行目は削除されたコード片の位置とメソッド呼び出し文が挿入された位置が同じであるか判定を行い，5 行目は新しく宣言されたメソッド名と，新しく挿入されたメソッド呼び出し文のメソッド名が同じであるか判定を行う．以上，5 つの条件が満たされた時，本研究ではメソッド抽出の集約パターンとして検出される．

3.5 コードクローン照合

メソッド抽出の集約パターンが検出されると，次に，メソッド抽出で抽出されたコード片のコードクローンを検出する必要がある．そこで，3.1 節で検出したコードクローンからメソッド抽出されたコード片のコードクローンに一致するものを照合して探す．CCFinderX のコードクローンの出力ファイルは，クローンセット ID，コードクローンが属するファイル，コードクローンの位置情報から成る．CCFinderX の出力ファイルの情報に基づいて，まずは，メソッド抽出により抽出されたコード片の位置情報と，コードクローンの位置情報が一致するコードクローンを探す．そして，それを満たすコードクローンが属するクローンセットのコードクローンをすべて検出する．

3.6 コードクローン提示

3.5 節で検出したコードクローン一覧を開発者に対して提示する．開発者は Eclipse の View を用いて，メソッド抽出が行われた，コード片のコードクローン一覧を見ること

図 3 コードクローンの例

図 4 メソッド抽出

ができる．さらに，開発者がコードクローン一覧からコードクローンを選択すると，そのコードクローンが含まれるファイルを開き，そのコードクローン部分をハイライトし，視覚的にもコードクローンを把握しやすくなっている．

4. 利用シナリオ

本章では図 3 にあらわすコードクローンの例を用いて，実際に本研究で構築した環境の利用シナリオを説明する．図 3 のソースコードは，ユーザー ID を設定する際，すでに登録されているユーザー ID ではなく，4 文字以上 15 文字以下という制限があり，さらにパスワードを設定する際，アルファベットと数字を両方必ず含み，4 文字以上 15 文字以下であるという制限があるシステムを想定している．そ

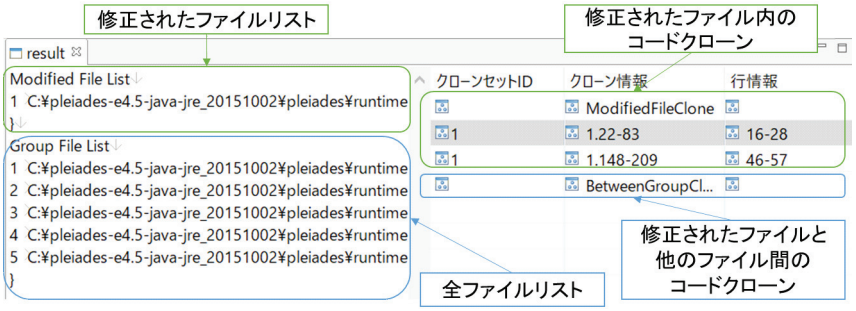

図 5　同時集約候補のコードクローン

れぞれの認証は，14 行目～32 行目の checkUserID メソッドと，35 行目～53 行目の checkPass メソッドによって行われている．これら 2 つのメソッドは，それぞれ重複ユーザー ID の確認とフォーマットの確認は別処理であるが，18 行目～27 行目と 42 行目～52 行目の長さに関する制限を確認する処理はコードクローンとなっている．

開発者は図 3 のソースコードに対して，編集作業を行う．ここからの開発者の編集作業を本環境がモニタリングする．モニタリングの際，本環境は，キー入力を追い，ソースコードから行差分を取得して，メソッド抽出の集約パターンに関係する AST ノードを探す．そして，開発者が図 3 のソースコードに対して図 4 のように，42 行目～52 行目の長さに関する制限を確認する処理部分を checkLength メソッドとして抽出する．開発者はその際，(1) 図 3 の 42 行目～52 行目のコード片の削除，(2) 新しい checkLength メソッドの宣言，(3)42 行目に新しい checkLength メソッドの呼び出し文の挿入，という 3 つの操作を順不同で行っているはずである．本環境は，その操作列を追い，各操作を (delta_type, ast_node) の組として蓄積し，3 つの操作列が検知された際に，メソッド抽出の集約パターンとして検出する．そして，本環境はメソッド抽出が行われたことを検出すると，“メソッド抽出を検出しました．”というメッセージダイアログを表示し，開発者に通知する．

次に，本環境は，メソッド抽出で抽出されたコード片のコードクローンを検出し，図 5 のように Eclipse の View 機能を用いて，同時集約候補のコードクローンを開発者に対して提示する．同時集約候補のコードクローンを提示する View では，左半分で，プロジェクトに含まれるファイルリストを表示し，右半分で実際に検出されたコードクローンを表示する．ファイルリストは以下の 2 種類に分類される．

Modified File List

修正が加えられたファイルが含まれる．

Group File List

プロジェクト内のすべてのファイルが含まれる．

本環境では 2 つの理由からファイルリストを 2 種類に分けた．まず 1 つ目の理由としては，CCFinderX の検出速度に起因する．CCFInderX は，入力としてプロジェクトを与え，そのプロジェクト全体のソースコード内のコードクローンを検出するよりも，ファイルリストを作り，そのファイルリスト内，ファイルリスト間でコードクローンを検出する方が，より高速にコードクローンを検出することができる．そのため，ファイルリストを 2 種類作ることにより，同時集約候補のコードクローンをより高速に検出することができる．また，2 つ目の理由としては，同時に集約されるべきコードクローンに対して優先順位を与えるためである．本環境では，ファイル内，ファイル間のコードクローンを検出することができる．しかし，コードクローンの集約において，ファイル間をまたぐコードクローンよりも，ファイル内に存在するコードクローンの方が優先順位が高いと考えられる．よって，以上の 2 つの理由から 2 種類のファイルリストを用意した．

図 6　コードクローン部分のハイライト表示

右半分の実際に検出したコードクローン情報の見方を説明する．まず，クローンセット ID は同じクローンセットに含まれるコードクローンに一意に与えられるものである．クローン情報はピリオドの前の数字が左半分のファイルリストに対応する数字で，ファイル名を示し，ピリオドの後の数字は，それぞれコードクローンの開始オフセットと，終了オフセットを示している．そして，行情報はコードクローン部分の行の開始位置と終了位置を示している．

先にも述べたように，コードクローンの検出結果は 2 種類に分けて表示するようにしており，上の ModifiedFileClone が修正されたファイル内のコードクローンを表示し，下の BetweenGroupClone は修正されたファイルとそれ以外のファイルをまたぐコードクローンを表示する．

最後に，図 5 の View から開発者が 1 つのコードクローンを選択すると，そのファイルを開き図 6 のように，そのコードクローン部分を緑色にハイライトして表示する．これにより，開発者は，コードクローン部分をより分かりやすく確認することができ，同時に集約するか検討することができると考えられる．

以上が，本環境の利用シナリオである．本環境によって，開発者はメソッド抽出を意識せず行っても，メソッド抽出を行ったことをその場で認識することができ，そのメソッド抽出を行ったコード片のコードクローンもその場で知ることができるため，作業の後戻り等無く，適切な時期に集約の支援を行うことができると考えらえる．

5. まとめと今後の課題

本研究では，開発者の作業内容をモニタリングし，コードクローンの集約を支援する環境を構築した．具体的には，開発者のソースコードの編集作業から，メソッド抽出の集約パターンを検知して，その抽出したコード片のコードクローンを開発者に提示するという支援内容である．利用シナリオに示す通り，本環境は，メソッド抽出の集約パターンを検知して，その抽出したコード片のコードクローンを提示することができた．

今後の課題としては，まず 1 つ目に，適応できるリファクタリングのパターンを増やすことが挙げられる．現在，検出するリファクタリングのパターンはメソッド抽出の集約パターンだけであるが，コードクローンの削減に適用可能なリファクタリングパターンは他にもたくさん考えられる．例えば，メソッドの移動，メソッドの引き上げ，メソッドのパラメータ化，テンプレートメソッドの形成，親クラスの抽出等が挙げられる．今後は，これらのリファクタリングパターンにも適用できることを目指す．今後の課題として，2 つ目に本環境の評価実験が挙げられる．評価実験としては 2 種類考えており，1 つ目は実際に本環境を使用してもらい，フィードバックを受ける実験を今後行っていこうと考えている．2 つ目は，実際に過去に行われたリファクタリング事例を取得して，それらのリファクタリング事例に対して，本環境がリファクタリング操作を検知し，そのコードクローンを検出することができるか評価していこうと考えている．

謝辞 本研究は JSPS 科研費 25220003，16K16034，15H06344 の助成を受けたものです．

参考文献

[1] Baker, B. S.: Finding clones with dup: Analysis of an experiment, *IEEE Trans. Softw. Eng.*, Vol. 33, No. 9, pp. 608–621 (2007).
[2] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S. and Bier, L.: Clone detection using abstract syntax trees, *Proc. of ICSM*, pp. 368–377 (1998).
[3] Choi, E., Yoshida, N., Ishio, T., Inoue, K. and Sano, T.: Extracting Code Clones for Refactoring Using Combinations of Clone Metrics, *Proc. of IWSC*, pp. 7–13 (2011).
[4] Foster, S. R., Griswold, W. G. and Leaner, S.: WitchDoctor: IDE support for real-time auto-completion of refactorings, *Proc. of ICSE*, pp. 222–232 (2012).
[5] Fowler, M.: リファクタリング-プログラミングの体質改善テクニック，ピアソン・エデュケーション (2009).
[6] 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).
[7] 肥後芳樹，吉田則裕：コードクローンを対象としたリファクタリング，コンピュータソフトウェア，Vol. 28, No. 4, pp. 43–56 (2011).
[8] 井上克郎，神谷年洋，楠本真二：コードクローン検出法，コンピュータソフトウェア，Vol. 18, No. 5, pp. 47–54 (2001).
[9] 神谷年洋，肥後芳樹，吉田則裕：コードクローン検出技術の展開，コンピュータソフトウェア，Vol. 28, No. 3, pp. 29–42 (2011).
[10] 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).
[11] Murphy-Hill, E., Parnin, C. and Andrew, P. B.: How we refactor, and how we know it, *Proc. of ICPC*, pp. 199–206 (2013).
[12] Myers, E. W.: An O (ND) difference algorithm and its variations., *Algorithmica*, pp. 251–266 (1986).
[13] Program Creek: Use JDT ASTParser to parse Single Java files., http://www.programcreek.com/2011/11/use-jdt-astparser-to-parse-java-file/.
[14] Roy, C. K., Zibran, M. F. and Koschke, R.: The vision of software clone management: Past, present, and future (keynote paper), *Proc. of CSMR-WCRE*, pp. 18–33 (2014).
[15] 山中裕樹，崔　恩瀞，吉田則裕，井上克郎，佐野建樹：コードクローン変更管理システムの開発と実プロジェクトへの適用，情報処理学会論文誌，Vol. 54, No. 2, pp. 883–893 (2013).
[16] Yamanaka, Y., Choi, E., Yoshida, N., Inoue, K. and Sano, T.: Applying Clone Change Notification System into an Industrial Development Process, *Proc. of ICPC*, pp. 199–206 (2013).
[17] 吉田則裕，肥後芳樹，神谷年洋，楠本真二，井上克郎：コードクローン間の依存関係に基づくリファクタリング支援，情報処理学会論文誌，Vol. 48, No. 3, pp. 1241–1442 (2007).
[18] Yoshida, N., Higo, Y., Kamiya, T., Kusumoto, S. and Inoue, K.: On Refactoring Support Based on Code Clone Dependency Relation, *Proc. of METRICS*, pp. 16:1–16:10 (2005).