Vol.2017-SE-197 No.4
2017/11/9

凝集度を用いたメソッドのインライン化の支援手法

山田 悠貴[1,a)] 崔 恩瀞[1] 吉田 則裕[2] 飯田 元[1]

概要：メソッドのインライン化は, 呼びさされている側のメソッドを呼び出している側に展開するリファクタリング検出パターンの 1 つであり, 実施回数が多いことが知られている. 開発者がメソッドのインライン化を実施する際, 多くのメソッドの中からインライン化対象のメソッドを見つけることは困難である. そこで, 本研究では, メソッドのインライン化に対して, 開発者がメソッドのインライン化対象のメソッドを特定するための支援手法を提案する. 具体的には, プログラムスライスに基づく凝集度メトリクスを用いて, インライン化実施後のメソッドの凝集度が高く保たれる呼び出し関係のある 2 つのメソッドをインライン化すべき関数対として推薦する. また, 提案手法を実装したツールの利用シナリオについて述べる.

キーワード：リファクタリング，メソッドのインライン化，凝集度

Support Inline Method Refactoring Using Slice-based Metrics

Yuki Yamada[1,a)] Eunjong Choi[1] Norihiro Yoshida[2] Hajimu Iida[1]

1. まえがき

ソフトウェアの開発において, ソフトウェアの保守や運用に多くの時間とコストが費やされており, 特にソフトウェアの保守にかかるコストは全体の 70%とされている [2]. このソフトウェア保守作業において, 開発者はソースコードを読み, 理解する作業に多くの時間を費している. リファクタリングとは, ソフトウェアの外部的振る舞いを保ちながら内部構造を変更する技術のことである. [5] リファクタリングを実施することによってソフトウェアの保守性の向上が期待できるため, 開発者はリファクタリング対象を見つけてリファクタリング作業をする必要がある. しかし, 大規模のソフトウェアにおいて, 開発者がリファクタリングの対象をみつけることは困難であるため, 開発者のリファクタリング作業を支援するために様々なツールが提案されている [4].

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[2] 名古屋大学
Nagoya University
[a)] yamada.yuki.yv2@is.naist.jp

メソッドのインライン化とは, 呼びさされている側のメソッドを呼び出している側に展開するリファクタリングパターンの 1 つである [5]. メソッドのインライン化は, 数多くあるリファクタリングパターンの中で, 実施回数が多いことが報告されている [7]. そのために，開発者のメソッドのインライン化作業を支援するツールが必要であるが，我々の知る限り, メソッドのインライン化の支援を行うツールは提案されていない.

凝集度は, モジュールの構成要素が機能的にまとまっているかを測る指標である. 凝集度の高いメソッドは同一の機能を実現していると考えられるため, メソッドの凝集度を高めることで, ソフトウェアの保守性の向上が期待できる. そこで, 本研究は, プログラムスライスに基づく凝集度メトリクス [12] を用いて, メソッドのインライン化を実施した後のメソッドの凝集度が高く保たれるものをインライン化すべき関数対として開発者へ推薦する. 本論文では, 提案手法のメソッドのインライン化の支援手法の説明を行い, その提案手法に基づいて開発するツールの利用シナリオを紹介する.

以降, 2 章では, 研究背景の説明を行い, 3 章では, 提案手

© 2017 Information Processing Society of Japan

法の説明を行う．4 章では, 提案手法の利用シナリオについて説明する．5 章では, 関連研究を紹介し, 6 章で, まとめと今後の課題について述べる．

2. 背景

本章では，本研究で用いる技術とその用語の定義を行う．2.1 節では, 凝集度と, 凝集度を測るためのメトリクスについて述べる．2.2 節では, リファクタリングについて述べ, 2.3 節ではリファクタリングパターンであるメソッドのインライン化について述べ, メソッド抽出とメソッドのインライン化の両者の関係について述べる．

2.1 凝集度

凝集度は, モジュールの構成要素が機能的にまとまっているかを測る指標であり, 一般にメソッドの品質評価に用いられる．凝集度の高いメソッドは, メソッド内のステートメント同士が密接な依存関係を持っており, 同一の機能を実現していると考えられるため, メソッドの凝集度を高めることでソフトウェアの再利用性や保守性の向上が期待できる．凝集度を測定するメトリクスは, CC (Cycromatic Complexity) や LOCM (Lack of Cohesion in Methods) [3] などが挙げられる．Weiser は, メソッドの凝集度を図るために, プログラムスライスを用いて 5 つのメトリクスを提案している [12]．プログラムスライスとは, 互いの演算に影響を与えるステートメントの集合のことである．ステートメントは 1 つの処理のことであり, 通常 ";" で区切られている．プログラムスライスを求めるには, スライス基点と呼ばれるステートメント (行番号) とそのステートメントに対応する変数を定める必要がある．**図 1** はプログラムスライスの例である．この図の例は, スライス基点と, そのスライス基点に対して依存関係のあるステートメントの集合を表している．ステートメント番号 6 (図 1 の赤い四角), 変数 j をスライス基点と定めた場合, 6 行目の演算に対しては, 5 行目の演算が影響を及ぼす．同様に, 5 行目の演算に対しては 3 行目と 4 行目の演算, 4 行目の演算に対しては 2 行目の演算が影響を及ぼす．このように, スライス基点に対して, ステートメントの集合であるプログラムスライスを求めることができる．図 1 の例では, スライス基点を 6 行目, 変数 j としたときに求まるプログラムスライスは, 2, 3, 4, 5, 6 行目のステートメント (図 1 の赤い点線四角) となる．ステートメント間の依存関係には, データ依存と制御依存の 2 つがある．データ依存は, 2 行目と 4 行目のような変数を介した依存関係であり, 2 行目において変数に値が代入されたことにより, 4 行目の演算結果に影響を与える．制御依存は, 5 行目と 6 行目のような制御文を介した依存関係である．6 行目の演算が実行されるかどうかは, 5 行目の制御文の判定に依存する．また, プログラムスライスにはバックワードスライスとフォワードスライスがある．バックワードスライスは, スライス基点に対して影響を与えるステートメントの集合であり, 図 1 の例はバックワードスライスである．フォワードスライスは, スライス基点が影響を与えるステートメントの集合である．

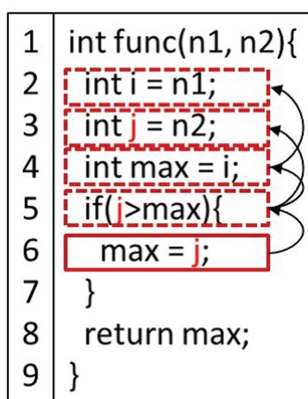

図 1 プログラムスライスの例
Fig. 1 An Example of Program Slicing.

2.2 リファクタリング

リファクタリングとは, ソフトウェアの外部的振る舞いを保ちながら内部構造を変更する技術のことである [5]．リファクタリングを実施することにより, 既存のコードの保守性や可読性を向上させることができる．Fowler は, 自身の本 [5] で, リファクタリングが実施されないプログラムの設計は徐々に劣化していき, 逆に, リファクタリングが定期的に実施されるプログラムは状態がしっかり保たれる, と述べている．また, この本では, 72 個のリファクタリングパターンについて, そのパターンの説明, いつすべきなのか, 手順などが記述されている．Fowler は, いつリファクタリングすべきかという問いに対して, コードの不吉な臭い (Bad Smell) を定義し, これに当てはまる状況のときはリファクタリングを実施すべきだと述べている．ただしこれは, 明確な実施のタイミングの基準ではなく, あくまで必要性を示す不吉な兆候なのであり, 開発者の経験や感覚に任せるべきであると述べている．開発者は, ソフトウェアの保守性や可読性を向上させるために, リファクタリングを実施する必要がある．しかし, 大規模なソースコードからリファクタリング対象を見つけることは困難である．そのため, 開発者のリファクタリング作業を支援するための多くの研究が行われている．

代表的なリファクタリングパターンの一つとしてメソッド抽出がある．メソッド抽出は, 既存のコードの一部を新たなメソッドの一部として抽出するリファクタリングである．メソッド抽出は, 長すぎるメソッドやメソッドに記述されている処理が不明瞭である, 処理が複数実装されている箇所への実施が推薦されている．メソッド抽出によって処理が機能ごとに分割されたメソッドは, 可読性や再利用

© 2017 Information Processing Society of Japan

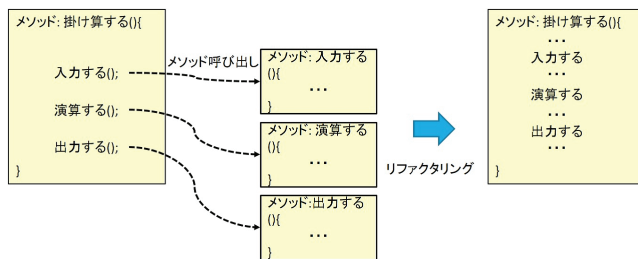

図 **2**　メソッドのインライン化の例

Fig. 2　An Example of Inline Method.

性の向上が期待できる．

2.3　メソッドのインライン化

メソッドのインライン化とは，呼び出している側のメソッドに呼び出されている側のメソッドの展開を行うリファクタリングである [5]．メソッドのインライン化は，呼び出されいる側のメソッドがメソッドとして名前をつけて呼び出すまでもなく処理が明らかな場合などに実施が推薦されている．メソッドのインライン化を実施することによって，不要なメソッドの呼び出しを除去することができ，メソッドの呼び出し構造の理解性を向上することができる．また，機能ごとに分割されていないメソッドに対して一度インライン化を実施し，その後，メソッドを再抽出するときにもメソッドのインライン化の実施が推薦されている．

数多くあるリファクタリングパターンの中で，メソッド抽出やメソッドのインライン化は実施回数が多いことが知られている [8]．メソッド抽出とメソッドのインライン化は互いに逆の操作となっており，メソッド抽出は，メソッドを機能ごとに分割しており，メソッド名とその処理が見て明らかな場合は個々のメソッドの理解性は向上するが，メソッドを細かく分割すると，メソッドの呼び出し関係が増え，全体の構造の理解性は低くなる．その反面，メソッドのインライン化は，適切な場面での実施がされればモジュールの構造を簡単にすることができるが，多くの呼び出し先のメソッドを呼び出し元のメソッドに展開すると，1 つのメソッドに複数の機能を含んでしまう危険性がある．このように，メソッド抽出とメソッドのインライン化は互いにトレードオフの関係があると考えることができ，互いに適切な頻度での双方向の実施，その支援を行う必要があると言える．しかし，メソッド抽出に対する支援の研究は多くされているが，メソッドのインライン化に関する研究は我々の知る限りではされていない．

3.　提案手法

本章では，提案手法の概要の説明と，提案手法の手順について述べる．

3.1　概要

本研究は，メソッドのインライン化を実施すべき箇所を特定するにあたって，以下の条件を満たす呼び出し関係のある 2 つのメソッド (以降，関数対と呼ぶ) がメソッドインライン化の対象に適切だと考えた．

- メソッドのインライン化を実施した後に，構成要素が機能的にまとまる関数対
- 呼び出し元の関数の構造が複雑および冗長にならない関数対

上記の条件を満たすために，本研究では凝集度という概念に着目して，メソッドのインライン化を実施した後のメソッドの凝集度が高く保たれており，かつ，そのサイズが大きくならない関数対をメソッドのインライン化の候補として開発者に提示する．

本手法は，Java ソースファイルを入力とし，メソッドのインライン化の候補の関数対を開発者に推薦する．まず，入力として与えられたソースコードから関数対を全て抽出する．抽出した関数対に対して，呼びさされている側のメソッドを呼び出している側に展開 (以降，インライン化と呼ぶ) した後のメソッドサイズ，共通する呼び出し先メソッドを持つかなどの条件に基づいて，絞込みを行う．次に，絞り込んだ関数対に対してインライン化を実施し，インライン化実施後のメソッドに対してプログラムスライスに基づく凝集度メトリクスを用いて凝集度を測り，凝集度が高いものから順に開発者に提示を行う．

3.2　手法

提案手法の概要を図 **3** に示す．本手法は Java ソースファイルを入力とし，メソッドのインライン化の候補の関数対を開発者に提示する．以降，提案手法の手順を説明する．

手順 1: 呼び出し関係のある関数対を抽出，絞り込み

この手順では，まず，入力として与えられた Java ソースファイルに対して，呼び出し関係のある関数対を抽出する．次に，抽出された関数対から以下の条件に該当するものを除外する．このとき，除外する条件は以下の通りである．

- 条件 1: メソッドのインライン化を実施した後の関数対のサイズ (空白・コメントなどを含む) が 146 行以上である [10].
- 条件 2: 呼び出し先メソッドが共通している．

条件 1 は，三宅らの研究結果に基づいて設定した [10]．三宅らは，COB (Cohesion Of Blocks) , LOC (Lines Of Code), CC を用いてそれぞれのメトリクスにおいてメソッド抽出すべきメソッドの特徴を調査し，メソッドのサイズが 146 行以上の場合，メソッド抽出リファクタリングの実施候補であると提案した．本研究では彼らの提案に基づいて，条件 1 を設定した．また，呼び出し先メソッドが共通している関数対をインライ

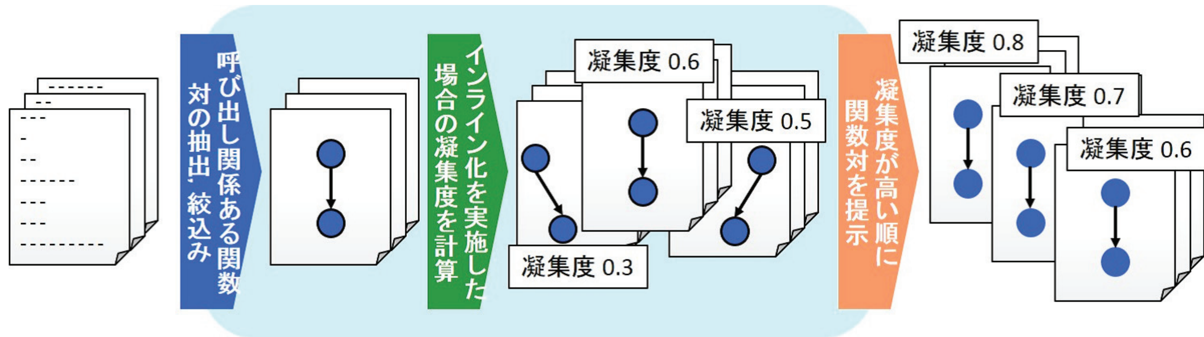

図 3　提案手法の概要
Fig. 3　An Overview of Our Approach.

ン化してしまうことで, コードクローン (ソースコード中に存在する同一または類似した部分を持つコード片) が生成される可能性がある [6]. コードクローンは, 一般にソフトウェアの保守性を低下させる主な原因として言われている. そのため, 本研究では, コードクローンの生成を防ぐために条件 2 を設定した.

手順 2: インライン化を実施した場合の凝集度を計算

この手順では, 手順 1 において絞り込んだ関数対に対して, メソッドのインライン化を実施した場合の凝集度を測る. 本研究では, 凝集度メトリクスは, Weiser が提案した凝集度メトリクス [12] を用いる. 詳細には, メソッド単位の凝集度を計測するため, プログラムスライスに基づく凝集度である $Tightness, Coverage, Overlap$ を用いる. 以下にそれぞれの凝集度メトリクスを計算する. 式中において, M は対象のメソッド, $length(M)$ はメソッド M のステップ数, V_0 は M におけるスライスの総数, SL_x は M における変数 x に対するスライス, SL_{int} は M における全変数に対するスライスを表す.

$$Tightness(M) = \frac{|SL_{int}|}{length(M)} \quad (1)$$

$$Coverage(M) = \frac{1}{|V_0|} \sum_{x \in V_0} \frac{|SL_x|}{length(M)} \quad (2)$$

$$Overlap(M) = \frac{1}{|V_0|} \sum_{x \in V_0} \frac{|SL_{int}|}{|SL_x|} \quad (3)$$

式 (1) に示す $Tightness$ は, メソッドのサイズに対する全てに共通するスライスを表している. $Tightness$ が高い場合, 1 つの処理を実行するために, それぞれのスライスが全て必要とされており, その全てのスライスを必要としている処理がどれくらいあるのかを表している. 図 3 の例では, $Tightness$ は 0.333 となっている. 式 (2) に示す $Coverage$ は, メソッドサイズに対して各スライスがどれほど含まれているかの平均である. メソッドに対して意味のあるスライスの割合を表している. $Coverage$ が高い場合, それぞれのスライスがメソッドの大部分で使用されていおり, 1 つの機能を実現するためにそれぞれのスライスが依存し合っていると考えられる. 図 3 の例では, $Coverage$ は $Coverage$ が 0.722 となっている. 式 (3) に示す $Overlap$ は, 各スライスに対する, 全てに共通するスライスの割合の平均である. $Overlap$ が高い場合, 互いのスライスが協調されて使用されているので, 1 つの機能を実現するために実装されていることが考えられる. 図 3 の例では, $Overlap$ は $Overlap$ が 0.577 となっている.

このように, 凝集度メトリクスはメソッドにおけるステートメントの協調度合いを表し, メソッドが単一機能であるか, 複数機能を含んでいる可能性があるかを考える際の指標となるため，本研究では上記の凝集度メトリクスを用いてメソッドのインライン化の候補をみつける.

手順 3: 凝集度が高い順に関数対を提示

この手順では, 開発者がメソッドのインライン化対象を選択する手間を減らすために, 凝集度を測定した関数対を, 凝集度が高い順に提示を行う.

4. 利用シナリオ

本章では，３章で説明した提案手法に基づいて開発するツールの利用シナリオを紹介する．我々は提案手法に基づいて，Eclipse プラグインを開発する予定である．Eclipse は現在広く使われている統合開発環境である．そのため，提案手法を Eclipse プラグインとして開発することにより，提案手法を多くの人に利用してもらうことが期待できる．開発する Eclipse プラグインの動作画面を図 **4** に示す. まず, 開発者は, 図 4 の (1) に示すように, Eclipse の画面にあるパッケージエクスプローラーからメソッドのインライン化を実施するプロジェクトを右クリックをし, 選択する. 図 4 の (2) に示すように, 対象プロジェクトを選択し，右クリックして表示されるメニューの中で「Inline Method」メニューを選択する. これによって, 対象とするプロジェクトに対して提案手法が適用される. つまり, 3.2 節で説明したようにまず, ソースコードから呼び出し関係のある関

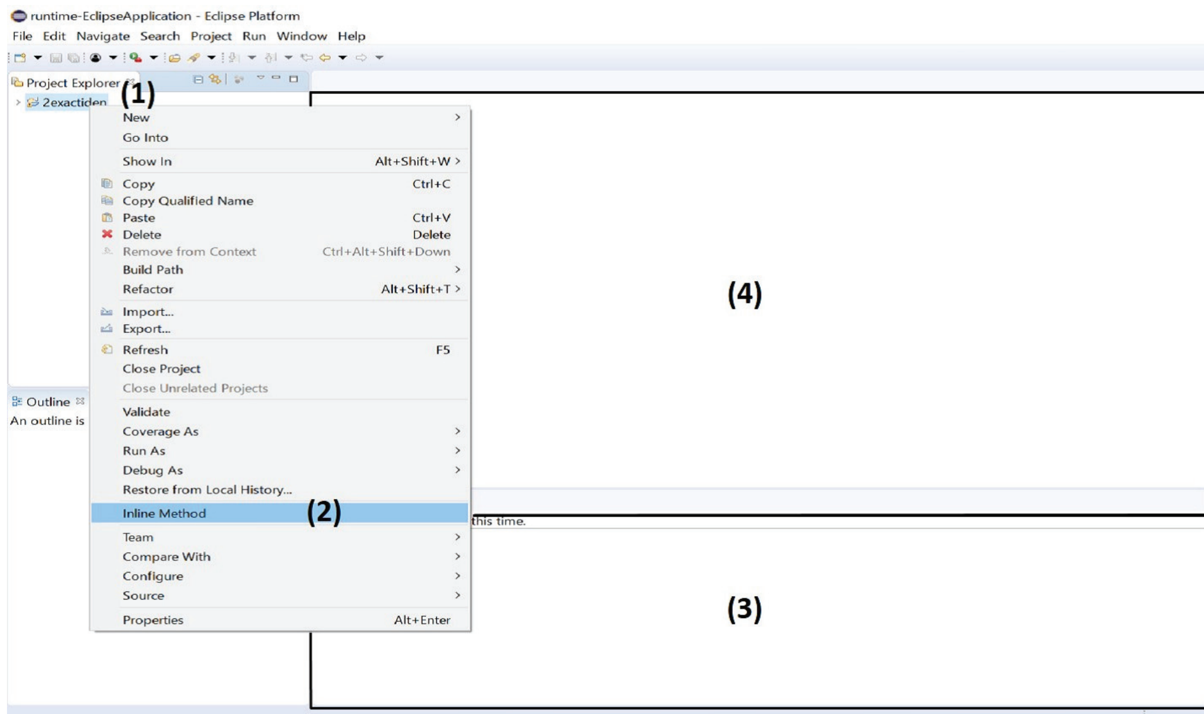

図 4 ツール画面

Fig. 4 An Screenshot of Developing Tool.

数対が抽出され, その関数対に対して凝集度メトリクスが測定される．その後, (3) の部分に, 呼び出し関係のある関数対の情報 (呼び出し元のメソッドの名前と, 呼び出し先のメソッドの名前) と, インライン化実施後の凝集度の一覧が表示される．このとき, これらの一覧は凝集度が高い順に表示される．開発者が一覧の中から 1 組を選択すると, (4) の部分のエディタに, 選択したものに対応する呼び出し元メソッド, 呼び出し先メソッドのソースコードが表示される．開発者は, 表示された呼び出し元メソッド, 呼び出し先メソッドを見ることで, メソッドのインライン化を実施するかどうかを判断することができ, メソッドのインライン化を実施することができる．

5. 関連研究

本章では, メソッド抽出に対する既存研究をを紹介する．

三宅らは, メソッド抽出の必要性を評価する凝集度メトリクス COB(Cohesion Of Blocks) を提案し, メソッド抽出が必要であるメソッドの特徴を調査した．また, 提案した凝集度メトリクスがメソッド抽出に対して有効であることを示した [10].

Tsantalis らは，リファクタリング支援ツール JDeodorant を開発している [11]．JDeodorant は，ソースコードから不吉な臭い [1] を特定し，そのコード片に適用すべきリファクタリングを開発者に提案する

後藤らは, 過去にメソッド抽出が実施されたメソッドの特徴を調査した．メソッド抽出が実施されたメソッドに対して, 凝集度メトリクスである $Tightness, Coverage, Overlap$ を用いて, メソッドの凝集度と, メソッドが抽出されたメソッドのサイズを計測した．その結果, メソッド抽出が実施されたメソッドはメソッドのサイズが大きく, 凝集度が低い傾向があることを明らかにした [1].

また，Silva らは，メソッド抽出リファクタリングの対象メソッドを特定し，それらをメソッド抽出を適用すべき度合い順で開発者に提示する手法を提案したる [8]．この手法は，メソッドから抽出可能な全てのコード片に対して，メソッド抽出されるコード片と抽出元コード片の依存関係が弱いコード片を，ランキングの上位に順位付ける．

これらの研究は, メソッド抽出の支援を行う研究である．メソッド抽出とメソッドのインライン化は, 数多くあるリファクタリングパターンの中で, 実施回数が多いため, 開発者のメソッドのインライン化作業を支援するツールが必要である．しかし, メソッドのインライン化を支援するツールの開発は行われていない．そのため, 本研究では, メソッドのインライン化の支援手法を提案した．

6. まとめと今後の課題

本研究では, プログラムスライスに基づく凝集度メトリクスを用いて, メソッドのインライン化を実施すべき関数対の推薦手法を提案した．提案手法では, 以下の手順を行う．呼び出し関係のある関数対を抽出し, 抽出した関数対を絞り込む．その後, インライン化を実施した後のメソッドに対して凝集度メトリクスをを計算し, 凝集度が高いも

© 2017 Information Processing Society of Japan

のから順に関数対を提示する．また, 提案手法の利用シナリオについて説明した．

今後の課題としては, まず, ツールの実装が挙げられる．現在, 提案手法のツールは実装中であり, Eclipse 上で, 対象プロジェクトを選択すると, 「Inline Method」のメニューが表示される機能まで実装した．今後は, 呼び出し関係のある関数対を抽出し, 絞り込んだ後の凝集度を計算し, 開発者に提示する機能を実装する予定である．次に, 提案手法によるメソッドのインライン化の支援の有効性を調査する予定である．具体的には, オープンソースソフトウェアプロジェクトから実際に開発者が実施したメソッドのインライン化が含まれているデータセットを用いて, 開発する支援ツールの正確性を評価する．このデータセットは, リファクタリングの実施履歴を調査した研究 [9] で用いられたものである．また, 開発するツールを実際に開発者に使用してもらい, アンケートを行う予定である．このアンケートでは, 開発者が考えているメソッドのインライン化実施対象と, ツールが提示したメソッドのインライン化の候補がどれくらい一致しているかを評価する予定である．

謝辞　本研究は JSPS 科研費 26730036，15H06344 の助成を受けたものです．

参考文献

[1] 後藤　祥，吉田則裕，藤原賢二，崔　恩瀞，井上克郎：メソッド抽出リファクタリングが行われるメソッドの特徴調査，コンピュータソフトウェア，Vol. 31, No. 3, pp. 318–324 (2014).

[2] Boehm, B. W.: Software Engineering, *IEEE Trans. Comput.*, Vol. 25, No. 12, pp. 1226–1241 (1976).

[3] Chidamber, S. R. and Kemerer, C. F.: A Metrics Suite for Object Oriented Design, *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6, pp. 476–493 (1994).

[4] 田中大樹，崔　恩瀞，吉田則裕，藤原賢二，飯田　元：プロセスメトリクスを用いたメソッド抽出事例の調査と予測モデルの構築，電子情報通信学会技術研究報告，Vol. 116, No. 512, SS2016-73, pp. 79–84 (2017).

[5] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).

[6] 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).

[7] Murphy-Hill, E., Parnin, C. and Black, A. P.: How We Refactor, and How We Know It, *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 5–18 (2012).

[8] Silva, D., Terra, R. and Valente, M. T.: Recommending Automated Extract Method Refactorings, *Proc. of ICPC*, pp. 146–156 (2014).

[9] Silva, D., Tsantalis, N. and Valente, M. T.: Why We Refactor? Confessions of GitHub Contributors, *Proc. of FSE*, pp. 858–870 (2016).

[10] 三宅達也，肥後芳樹，井上克郎：メソッド抽出の必要性を評価するソフトウェアメトリクス，電気情報通信学会，Vol. 108, No. 173, pp. 73–78 (2008).

[11] Tsantalis, N. and Chatzigeorgiou, A.: Identification of extract method refactoring opportunities for the decomposition of methods, *The Journal of Systems and Software*, Vol. 84, pp. 1757–1782 (2011).

[12] Weiser, M.: Program Slicing, *Proc. of ICSE*, pp. 439–449 (1981).