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.

スライスに基づく凝集度を用いたメソッド抽出支援手法の実験的評価

山口　佳久† 　吉田　則裕†† 　後藤　　祥† 　井上　克郎†

† 大阪大学大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 1–5
†† 奈良先端科学技術大学院大学情報科学研究科　〒 630–0192 奈良県生駒市高山町 8916–5
E-mail: †{y-kaku,a-gotoh,inoue}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp

あらまし　ソフトウェアの品質向上は，ソフトウェア開発や保守の効率を高めるために重要な活動である．リファクタリングはソフトウェアの品質を保つために重要な作業であるが，複雑な作業を伴うため，開発者には多くの知識や経験が要求される．本研究では，プログラムスライスに基づく凝集度を用いたメソッド抽出リファクタリング支援手法について評価実験を行った．その結果，手法の有効性の有無，有効に働く場合の条件について知見を得た．
キーワード　リファクタリング，メソッド抽出，凝集度

An Empirical Evaluation of Slice-based Cohesion Approaches to Supporting Extract Method

Kaku YAMAGUCHI†, Norihiro YOSHIDA††, Akira GOTO†, and Katsuro INOUE†

† Graduate School of Information Science, Osaka University
Yamadaoka 1–5, Suita, Osaka 565–0871 Japan
†† Graduate School of Information Science, Nara Institute of Science and Technology
Takayama-cho 8916-5, Ikoma, Nara 630–0192 Japan
E-mail: †{y-kaku,a-gotoh,inoue}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp

Abstract　Software quality improvement is one of the important activities to enhance the efficiency of development and software maintainability. Refactoring makes software easier to understand and change. However, the criteria of understandability and maintainability are different among developers. In this study, we performed the evaluation of the technique supporting extract method refactoring used by slice-based cohesion. As a result, we found the efficacy of the technique and conditions which the technique work effectively.
Key words　refactoring, extract method, cohesion metric

1. ま え が き

近年，ソフトウェアは業務管理システムなど，様々な分野で利用されており，大規模化及び複雑化が進んでいる．それに伴い，ソフトウェアの開発や保守のコストも増大している．そのため，ソフトウェアの開発や保守を効率的に行うことができるように設計することが求められるが，設計段階でこのような構造を決定することは難しい．そこで開発及び保守の段階で構造を修正することができるリファクタリングが注目されている．

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った状態で，内部の構造を改善し，ソフトウェアの品質を高める作業である [2]．リファクタリングには様々なパターンが存在し，代表的なものとして既存のメソッドの一部を，新しいメソッドとして抽出するメソッド抽出と呼ばれるリファクタリングパターンが存在する [2]．このパターンを用いることで，複数の機能を持つメソッドや長すぎるメソッドを，機能単位の小さなメソッドに分割することができる．

しかし，手作業でリファクタリングを行うには 3 つの問題がある [2]．まず第 1 に，どのようなソースコードに対してリファクタリングを行うかという厳密な基準が存在しておらず，開発者には多くの経験と知識が要求される．第 2 に，大規模及び複雑化したソフトウェアの場合，手作業でリファクタリングを行うのは困難である．このような問題を解決するために，リファクタリング作業を支援する方法が必要になる．

メソッド抽出リファクタリングを行うためにはまず，メソッドとして抽出する範囲を決定する作業が必要となる．既存研究では，凝集度を用いてソースコードを機能ごとのまとまりに分割し，それをメソッドとして抽出することで，この作業を支援

している [11]．しかし，既存研究では，各メトリクスが有効に働く条件や各メトリクスに基づいて提示されるメソッド抽出候補の特徴について，評価が行われていない．そのため，開発者はメトリクスを選択するための知識を持たないまま，試行錯誤をする必要があり，効率的にリファクタリングを行うことが困難であると言える．

そこで本研究では，各凝集度メトリクスが有効に働く条件や，各メトリクスに基づいて提示されるメソッド抽出候補の特徴について調査を行った．オープンソースソフトウェアから有用なメソッド抽出リファクタリングの事例を収集し，評価対象の手法によって得られるメソッド抽出候補との比較を行い，メソッド抽出リファクタリング支援手法の評価を行った．その結果，各メトリクスが有効に働く条件や，各メトリクスに基づいて提示されるメソッド抽出候補の特徴について知見を得ることができた．

以降，2. 章では関連技術について説明する．3. 章では，本研究において評価対象とする手法について説明する．4. 章では，本研究の動機について述べる．5. 章では評価方法について説明する．6. 章では，結果及び考察について述べる．7. 章ではまとめと今後の課題について述べる．

2. 関 連 技 術

2.1 メソッド抽出リファクタリング

メソッド抽出リファクタリングとは，既存のメソッドの一部を，新しいメソッドとして抽出する作業のことである．メソッド抽出リファクタリングの主な用途としては，複数の機能を持つメソッドや長すぎるメソッドの一部を抽出して，機能ごとに分けられた短いメソッドに分割することが挙げられる．分割することにより以下のような利点が生まれる [2]．

- 機能ごとに分けられたメソッドは，再利用性が向上する．
- 抽出したメソッドに適切な命名を行うことで，可読性が向上する．
- メソッドの粒度が細かくなるので，オーバーライドを行いやすい．

メソッド抽出リファクタリングは，他のリファクタリングが行われる前段階で頻繁に利用される，普遍的なリファクタリングパターンの 1 つである [2]．そのため，既存の研究においても，様々な自動化手法が提案されている [7], [8], [10]．しかし，リファクタリングはどのようなソースコードに行うべきかという厳格な基準は存在しない．Fowler はリファクタリングがいつ行われるかについての正確な基準を設けるのではなく，過去の経験や知識を基にリファクタリングが行われる可能性のある兆候 (コードの不吉な匂い) を定義している [2]．

2.2 プログラムスライスを用いた凝集度メトリクス

凝集度とは，モジュール内の構成要素が特定の機能を実現するために，協調している度合いを表す尺度である．凝集度が高いほどモジュールの可読性，保守性が高いといわれている [6]．凝集度を用いて，プログラム理解を支援する手法が既存研究で提案されている [12]．また，Weiser はメソッドや関数の凝集度を評価するためにプログラムスライスを利用した 5 つの凝集度メトリクスを提案している [9]．これらの凝集度メトリクスは Ott らによって定式化，及び実験がなされ，実験により Tightness，Overlap，Coverage と呼ばれる 3 つのメトリクスの有用性が高いことが示された [4], [5]．Tightness，Overlap，Coverage の定義式を式 1，2，3 に示す．式において，メソッドを M，$length(M)$ をメソッド M の文の数，V_0 を M における返り値などの出力変数の集合，SL_{int} を V_0 中の全変数に対する後ろ向きスライスの積集合とする．

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

$$Overlap(M) = \frac{1}{V_0}\sum_{x \in V_0}\frac{|SL_{int}|}{|SLx|} \tag{2}$$

$$Coverage(M) = \frac{1}{V_0}\sum_{x \in V_0}\frac{|SL_x|}{length(M)} \tag{3}$$

3. 凝集度に基づくリファクタリング支援手法

後藤らはプログラムスライスを用いた凝集度メトリクスを使用して，メソッド抽出リファクタリングを支援する手法を提案している [11]．後藤らの手法は，2.2 節で述べた 3 つのメトリクスを応用した，FTightness，FCoverage，FOverlap という 3 つのメトリクスを使用している．それぞれの定義式を式 4，5，6 に示す．式において，メソッドを M，$length(M)$ をメソッドの文の数，V_i を M における引数の集合，V_0 を M における返り値の集合，FSL_x を引数 x を起点にした前向きスライス，BSL_x を変数 x を起点にした後ろ向きスライス，SL_{int} を V_i 中の全変数に対する前向きスライスと V_0 中の全変数に対する後ろ向きスライスの積集合とする．

$$FTightness(M) = \frac{|SL_{int}|}{length(M)} \tag{4}$$

$$FOverlap(M) = \frac{1}{|V_0|+|V_i|}\left(\sum_{x \in V_i}\frac{|SL_{int}|}{|FSLx|} + \sum_{x \in V_0}\frac{|SL_{int}|}{|BSLx|}\right) \tag{5}$$

$$FCoverage(M) = \frac{1}{|V_0|+|V_i|}\left(\sum_{x \in V_i}\frac{|FSL_x|}{length(M)} + \sum_{x \in V_0}\frac{|BSL_x|}{length(M)}\right) \tag{6}$$

凝集度に基づくリファクタリング支援手法では，対象となるメソッドを与えると，まずそのメソッド内でメソッドとして抽出可能なコード片をすべて列挙する．そして，それらのコード片に対して，プログラムスライスを用いた凝集度メトリクスの値を計算する．これらのメトリクスはメソッドを対象としているが，ここではコード片の凝集度を計算する必要がある．そのため，コード片をメソッドとして抽出したとして，そのメソッドに対して凝集度を計算し，その結果をコード片の凝集度としている．

図 1 に凝集度の計算例を示す．凝集度の計算はまず，スライシング基準を定める．スライシング基準は，引数となる変数と，それらの変数を最初に参照している文の組，返り値となる変数と return 文の組とする．図 1 の FSL_a はスライシング基準が (2,a) である前向きスライスに含まれる頂点の集合であ

	FSL_a	FSL_b	BSL_c	SL_{int}
1int method(int a,int b){				
2 int a1=a;	2		2	
3 int b1=b;		3	3	
4 int c = a1;	4		4	
5 if(b1 > c)	5	5	5	5
6 c = b1;	6	6	6	6
7 return c;	7	7	7	7
8}				

$$\text{FTight}nes = 0.5$$
$$\text{FOverlap} = 0.833$$
$$\text{FCoverage} = 0.6167$$

図 1　抽出されたコード片の凝集度の計算例

る．図 1 の FSL_b はスライシング基準が (3,b) である前向きスライスに含まれる頂点の集合である．図 1 の BSL_c はスライシング基準が (7,c) である後ろ向きスライスに含まれる頂点の集合である．これらのスライスを用いて，全てのスライスの積集合 SL_{int} を求める．求めたスライスを用いて凝集度を計算すると図 1 のように $FTightness = 0.5$，$FOverlap = 0.833$，$FCoverage = 0.6167$ となる．

これをすべてのメソッド抽出可能なコード片に対して行い，計算した凝集度毎にメソッド抽出候補の順位付けを昇順で行う．

4. 本研究の動機

メソッド抽出リファクタリングが必要なメソッドの特定方法は，LOC(Lines Of Code) などのソフトウェアメトリクスを使用すれば，ある程度自動化することは可能である．一方で，メソッドとして抽出する範囲を決定する作業は，開発者にとってコストが大きい．

具体的な理由としては，メソッドを機能ごとに分割するには，開発者がソースコードを読んで，処理内容を理解する必要がある点が挙げられる．また，メソッド抽出の際に，メソッド抽出しようとしている範囲のコードと，その範囲外のコードとのデータの依存や制御フローの依存を考慮する必要がある [3]．以上の理由から，メソッド抽出範囲を決定する作業について支援する手法が重要であるといえる．既存研究では，メソッド抽出範囲を開発者に提示する手法が提案されている [11]．

しかし，この手法は，メソッド抽出リファクタリング支援手法としての有効性評価が行われていない．そのため，対象によっては，可読性が向上するメソッド抽出候補を提示しない場合が考えられる．

5. 評 価 方 法

本研究では，対象とする手法の有効性を評価することを目的としている．そのため，評価実験では凝集度を用いたメソッド抽出リファクタリング支援手法による結果を，本研究の評価基準に基づいて評価した．実際にメソッド抽出が行われた範囲と，手法を適用することで得られるメソッド抽出範囲の類似度が高いほど，有効性が高いと考えた．Bellon らは，複数のコードクローン検出ツールが提示するクローン候補の比較，評価を行っている [1]．既存研究では OK 値，GOOD 値，Precision，Recall というメトリクスが定義されている．OK 値，GOOD 値はコード片の重複の度合いを表すメトリクスであり，Bellon がコードクローンであると定めた範囲と，ツールを用いて得られた結果を，これらのメトリクスを用いて比較している．また，Precision はコードクローン検出ツールが検出するコードクローンのうち正解の割合を表し，Recall は正解集合のうち，コードクローン検出ツールが検出した割合を表す．

本研究では，既存研究で定義された GOOD 値，Precision，Recall を参考にした評価基準を定義した．5.1 節では，GOOD 値，$Precision_{good}$，$Recall_{good}$ について説明する．

5.1 評 価 基 準

5.1.1 GOOD 値

本研究で用いる評価の定義式を式 7 に示す．GOOD 値において，CF_1 を支援手法を使用して得られるメソッド抽出候補，CF_2 を正解集合に含まれるメソッド抽出範囲，$contained$ を一方のコード片において，他方のコード片が含まれている割合，$overlap$ を 2 つのコード片に共通するコードの割合を表す．

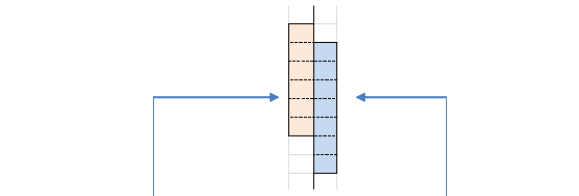
$$\begin{aligned} &GOOD(CF_1, CF2) \\ &= max(contained(CP_1.CF_1, CP_2.CF_1), \\ &\qquad contained(CP_2.CF_1, CP_1.CF_1))) \end{aligned} \tag{7}$$

GOOD 値の例を図 2 に示す．図の左が手法によって得られたメソッド抽出候補，右が正解集合から得られたメソッド抽出範囲である．ブロック 1 つがソースコードのステートメントに相当する．2 つのコード片を比較して GOOD 値を求める．図 2 の場合は $GOOD = \frac{5}{8}$ になる．

5.1.2 $Precision_{good}$ と $Recall_{good}$

本研究で用いる評価の定義式を式 8，9 に示す．$Precision_{good}$ と $Recall_{good}$ において，P を対象とするプログラム，M をメソッド抽出リファクタリング支援手法で使用したメトリクス，$GOODCands(P, M, x)$ をメトリクス M を使用したメソッド抽出リファクタリング支援手法を用いて得られた，正解集合内の要素 x に対して $GOOD$ 値 $\geqq 0.7$ であるメソッド抽出候補の集合，$Cands(P, M, x)$ をメトリクス M をも使用したメソッド抽出リファクタリング支援手法を用いて得られた，正解集合内の要素 x に対するメソッド抽出候補の集合，$Refs(P)$ を正解集合から得られるメソッド抽出の集合．$GOODRefs(P, M, x)$ をメトリクス M を使用したメソッド抽出リファクタリング支援手法を用いて得られた $GOOD$ 値 $\geqq 0.7$ である Refs(P) の集合を表す．

$$\begin{aligned} &Precision_{good}(P, T) \\ &= \frac{1}{|Refs(P)|} \sum_{x \in Refs(P)} \frac{|GOODCands(P, T, x)|}{|Cands(P, x)|} \end{aligned} \tag{8}$$



$$GOOD(CF_1, CF_2) = \frac{5}{8}$$

図 2　GOOD 値の例

$$Recall_{good}(P,T) = \frac{|GOODRefs(P,T)|}{|Refs(P)|} \quad (9)$$

5.2　適 用 実 験

評価実験のオープンソースソフトウェアには，jEdit(注1) を用いた．また，オープンソースソフトウェアを解析して得られる，正解集合に含まれるメソッド抽出リファクタリングは，以下のような条件を満たす．

- あるクラスにおいて「行の削除及び行の追加が行われたメソッド」（以降，M_1 と呼ぶ）と「新規に追加されたメソッド」（以降，M_2 と呼ぶ）が存在する．
- M_1 に M_2 の呼び出しが追加されている．
- M_1 において，削除された行と M_2 の類似度 S が閾値以上である．

これらの条件を満たすメソッドを目視で確認し，メソッド抽出が行われたと判断できた事例を正解集合とした．以降では，I_X を正解集合のうち類似度 S が X 以上の事例から構成される集合とする．例えば，$I_{0.9}$ は類似度 0.9 以上のメソッド抽出事例の集合である．

6.　結果と考察

実験を行い，図 3〜図 7 のような結果が得られた．グラフは，リファクタリング支援手法で用いられるメトリクスが，メソッド抽出すべきと判定する閾値を変化させたときの Precision，もしくは Recall の推移を表す．四角の点のグラフが Precision，三角の点のグラフが Recall を示す．横軸がメソッド抽出候補であると手法が判定するための，凝集度メトリクスの閾値を示す．

Tightness に関して，$I_{1.0}$，$I_{0.9}$，$I_{0.8}$，$I_{0.7}$，$I_{0.6}$ の場合のすべてにおいて，フィルタリングしない場合がもっともよい Precision が得られた．Recall に関しても常に減少しているので，Recall の観点から見ても Tightness はメソッド抽出には不向きであると考えられる．

Coverage に関して，$I_{1.0}$ のみ異なるが，フィルタリングする値を 0.3 にすると，Precision の値が最も高くなる．Recall に関しては，フィルタリングする値を増加させると Recall が大幅

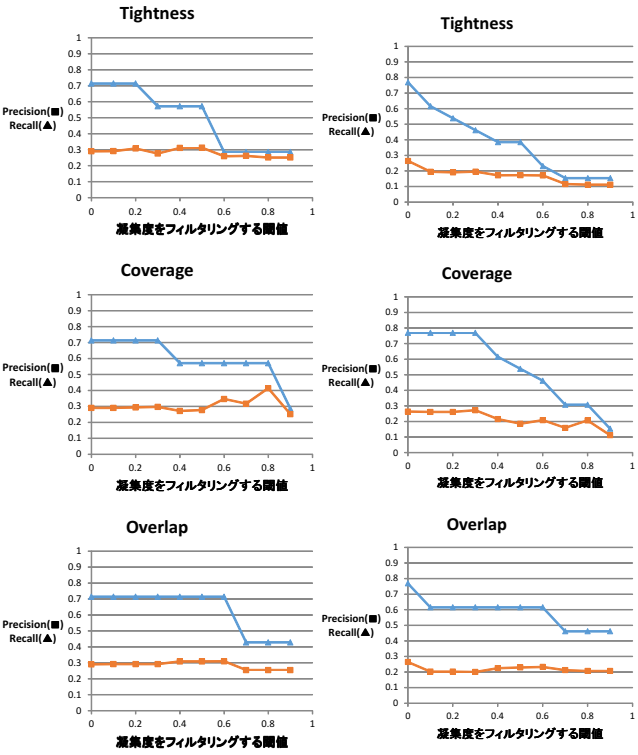


図 3　$I_{1.0}$ に対する precision 及び recall

図 4　$I_{0.9}$ に対する precision 及び recall

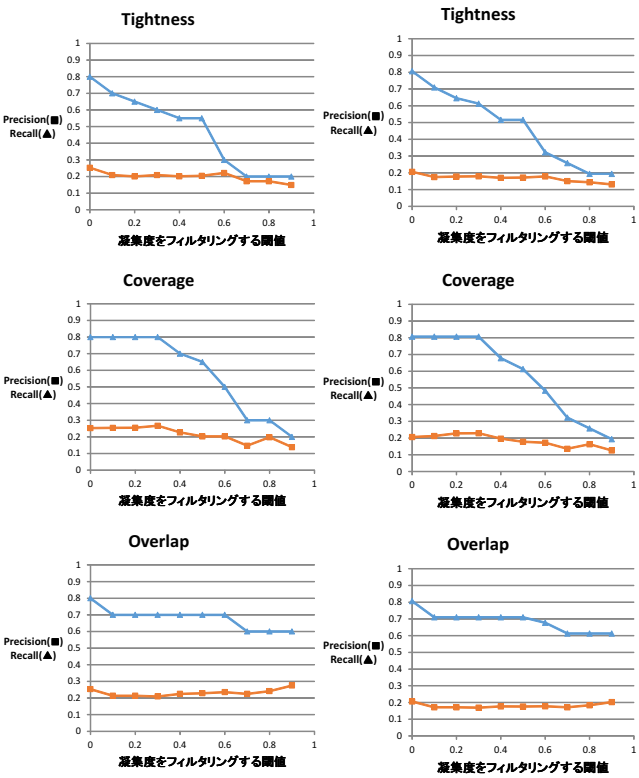


図 5　$I_{0.8}$ に対する precision 及び recall

図 6　$I_{0.7}$ に対する precision 及び recall

(注1)：http://www.jedit.org/

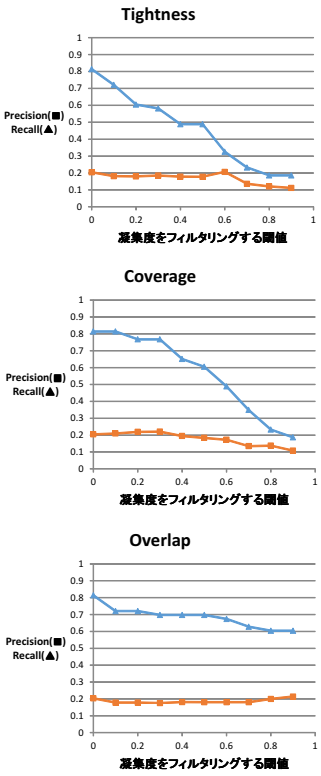


図 7　$I_{0.6}$ に対する
precision 及び recall

に減少する．

Overlap に関して，フィルタリングする値を 0，もしくは 0.9 に近づけるほど Precision が高くなる傾向が見られる．Recall は Tightness，Coverage ほど増減は変化せず，Tightness や Coverage に比べて相対的に高い値を維持している．

これらの結果から，Precision と Recall が最大になるような値を設定することができれば，Coverage メトリクスを用いたメソッド抽出を有効に使用できると考えられる．また，解になるメソッド抽出範囲を見つけたい場合には Recall メソッドが重要なので，Overlap メトリクスを用いたメソッド抽出が有用であると考えられる．次に I_X に関して，Overlap メソッドは I_X の X の値が低いほど，Recall の値が高くなっていることがわかる．つまり，メソッドとして抽出した範囲が後に，追加や修正される可能性が高いものは，Overlap メトリクスを用いたメソッド抽出が有用であると考えられる．

7. あとがき

本研究では，既存手法の有効性評価を目的として，プログラムスライスを用いた凝集度に基づくメソッド抽出リファクタリング支援手法の評価を行った．評価では，書籍やオープンソースソフトウェアから有用なメソッド抽出リファクタリングの事例を収集し，評価対象の手法によって得られるメソッド抽出候補との比較を行った．その結果，プログラムスライスを用いた凝集度に基づくメソッド抽出リファクタリング支援手法において，使用する凝集度メトリクスにより，提示されるメソッド抽出候補の特徴や，有効に働く場合の条件に差異が存在することがわかった．

今後の課題として，正解集合の改良が挙げられる．今回の評価実験ではオープンソースソフトウェアのリポジトリを調査し，実際にメソッド抽出リファクタリングが行われた事例を正解集合としている．これは，十分成熟したオープンソースソフトウェア上で修正を加える開発者は常に正しい修正を行う，という考えに基づいている．そのため，開発者が気付くことができないメソッド抽出リファクタリングは正解集合には含まれていない．この開発者が気付くことができないが，メソッド抽出リファクタリングを行ったほうがよい部分を正解集合に含むことができれば，より正確に有効性を評価することができる．

また既存研究を適用したツールを実装し，開発者にアンケート調査をすることが考えられる．本研究では，1 つの正解集合に対して比較，評価を行っているので，アンケート調査を行うことで，多面的な観点から手法の評価ができると考えられる．

謝　辞

本研究において，データ提供をしていただきました奈良先端科学技術大学院大学ソフトウェア設計学講座，藤原賢二氏に感謝します．

文　献

[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *IEEE Trans. Soft. Eng*, Vol. 33, No. 9, pp. 577–591, 2007.

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

[3] E. Murphy-Hill and Andrew P. Black. Breaking the barriers to successful refactoring: observations and tools for extract method. In *Proc. of ICSE '08*, pp. 421–430, 2008.

[4] L. M. Ott and J. M. Bieman. Program slices as an abstraction for cohesion measurement. *Information and Software Technology*, Vol. 40, No. 11-12, pp. 681–699, 1998.

[5] L.M. Ott and J.J. Thuss. Slice based metrics for estimating cohesion. *In Proc. of METRICS '93*, pp. 71–81, 1993.

[6] W. P. Stevens, G. J. Myers, and L. L. constatine. Structured design. *IBM Systems Journal*, Vol. 13, No. 2, pp. 115–139, 1974.

[7] 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ックスの提案. 電子情報通信学会論文誌, Vol. J92-D, No. 7, pp. 1071–1073, 2009.

[8] N. Tsantalis and A. Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *Journal of Systems and Software*, Vol. 84, No. 10, pp. 1757–1782, 2011.

[9] M Weiser. Program slicing. In *Proc. of ICSE '81*, pp. 439–449, 1981.

[10] 丸山勝久. 基本ブロックスライシングを用いたメソッド抽出リファクタリング. 情報処理学会論文誌, Vol. 43, No. 6, pp. 1625–1637, 2002.

[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. 2, 2013.

[12] 平山力地, 吉田則裕, 飯田元. スライスに基づく凝集度を用いて自動分割を行うプログラム理解支援手法. 電子情報通信学会技術報告 SS2012-31, Vol. 112, No. 164, pp. 127–132, 2012.