特別研究報告

題目

メソッド抽出リファクタリング支援手法の有効性評価

指導教員
井上 克郎 教授

報告者
山口 佳久

平成 25 年 2 月 12 日

大阪大学 基礎工学部 情報科学科

平成 24 年度 特別研究報告

メソッド抽出リファクタリング支援手法の有効性評価

山口 佳久

内容梗概

ソフトウェアの品質向上はソフトウェアの開発や保守の効率を高めるために重要な活動である．リファクタリングとは，ソフトウェアの外部的振る舞いを保ちつつ，内部の構造を改善する作業を指し，ソフトウェアの品質を向上させるために有効である．

リファクタリングはソフトウェアを理解しやすく，変更を容易にするために行う作業であるが，理解の容易性や保守性などの基準は開発者の主観に依る部分が大きい．そのため，どのような場合ににリファクタリングを行うか，厳密な基準は存在しない．さらに，リファクタリング作業は変数の依存関係の調査など，複雑な作業を伴うため，開発者には多くの知識や経験を要求される．したがって，大規模ソフトウェアに対してリファクタリングを行う時には，熟練した開発者であっても，膨大な時間を要する．

リファクタリングの代表的なパターンとして，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リファクタリングが存在する．メソッド抽出リファクタリングは，メソッドを機能単位に分割する場合や，長すぎるメソッドを複数の短いメソッドに分割する場合に行われる．メソッド抽出リファクタリングは頻繁に使用されているため，その作業を支援する手法が，既存研究によって複数提案されている．

しかし，既存研究で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おらず，開発者によいメソッド抽出候補を提示するとは限らない．メソッド抽出リファクタリング支援手法が有効に働く条件を認識することで，開発者に理解しやすいメソッド抽出候補を提示できると考えられる．そこで，本研究では代表的なリファクタリングパターンの１つであるメソッド抽出リファクタリングを支援する手法について評価実験を行った．

評価実験では，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，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，各手法の有効性の有無や，有効に働く場合の条件について知見を得た．

主な用語

リファクタリング
メソッド抽出
凝集度

目 次

1 まえがき **3**

2 背景 **5**

2.1 リファクタリング 5

2.2 メソッド抽出リファクタリング 5

2.3 凝集度に基づくリファクタリング支援手法 7

2.3.1 プログラムスライス 7

2.3.2 プログラムスライスを用いた凝集度メトリクス 9

2.3.3 プログラムスライスを用いた凝集度に基づくリファクタリング支援手法 10

2.4 既存のメソッド抽出リファクタリング支援手法の問題点 11

3 評価実験 **13**

3.1 既存研究による評価方法 13

3.1.1 OK 値と GOOD 値 13

3.1.2 Precision と Recall 14

3.2 本研究での評価方法 15

3.3 実験準備 16

3.4 結果 16

3.5 考察 17

4 あとがき **24**

謝辞 **25**

参考文献 **26**

1　まえがき

近年，ソフトウェアは生産や販売などの業務管理システムや，銀行の預金口座を管理するシステムなど，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．この作業を支援する手法は，既存研究で提案されている [9, 14]．しかし，既存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おらず，対象によっては開発者によいメソッド抽出候補を提示しない場合が考えられる．手法が有効に働く条件を認識することで，ソースコードの可読性，保守性が上がるようなメソッド抽出候補を提示してくれる手法を適用でき，開発者に理解しやすいメソッド抽出候補を提示できると考えられる．

そこで本研究では，メソッド抽出リファクタリング支援手法の有効性を評価することを目的とし，対象の手法が開発者の可読性，保守性が向上するメソッド抽出候補を提示するかを調査した．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，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，各手法の有効性の有無や，有効に働く場合の条件について知見を得た．

以後，2 章では本研究の背景として，リファクタリング，メソッド抽出リファクタリング，凝集度に基づくリファクタリング支援手法，既存のメソッド抽出リファクタリング支援手法

の問題点について述べる．3 章では既存研究による評価方法，本研究での評価方法，結果，考察について述べる．4 章ではまとめと今後の課題について述べる．

2　背景

本研究の背景としてリファクタリング，リファクタリングパターンの 1 つであるメソッド抽出リファクタリング，評価対象の手法について簡単に述べる．

2.1　リファクタリング

リファクタリングとは “外部からみた振る舞いを保ちつつ，理解や修正が簡単になるようにソフトウェアの内部構造を改善していくこと” である [2]．Fowler は，開発者が設計の全体を理解せずに開発を行うとコードの構造を崩し，コードを読んで設計を把握することが困難になると述べている．そこでリファクタリングを定期的に行い，コードの構造を保つことの重要性を説いている [2]．

リファクタリングにはいくつかのパターンがあるが，代表的なリファクタリングとして，メソッド抽出やメソッドの引き上げなどが挙げられる．それらの中から本研究で対象としているメソッド抽出リファクタリングについて説明する．

2.2　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は以下の手順で行われる [2, 13]．

手順 1 メソッド抽出リファクタリングが必要なメソッドを特定する．

手順 2 抽出範囲を決定し，メソッドとして抽出する．

手順 3 変更後のプログラムの外部的振る舞いが保たれているか検証する．

図 1 にメソッド抽出リファクタリングの例を示す．左図がメソッド抽出リファクタリング前で，右図がメソッド抽出リファクタリング後である．リファクタリング前の，四角で囲まれた部分がポイント計算する機能なので，メソッドとして抽出している．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，まとまりのある機能かどうかの指標であるモジュール性が高まり，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，開発者にとって処理の理解が容易になり，可読性が向上する．

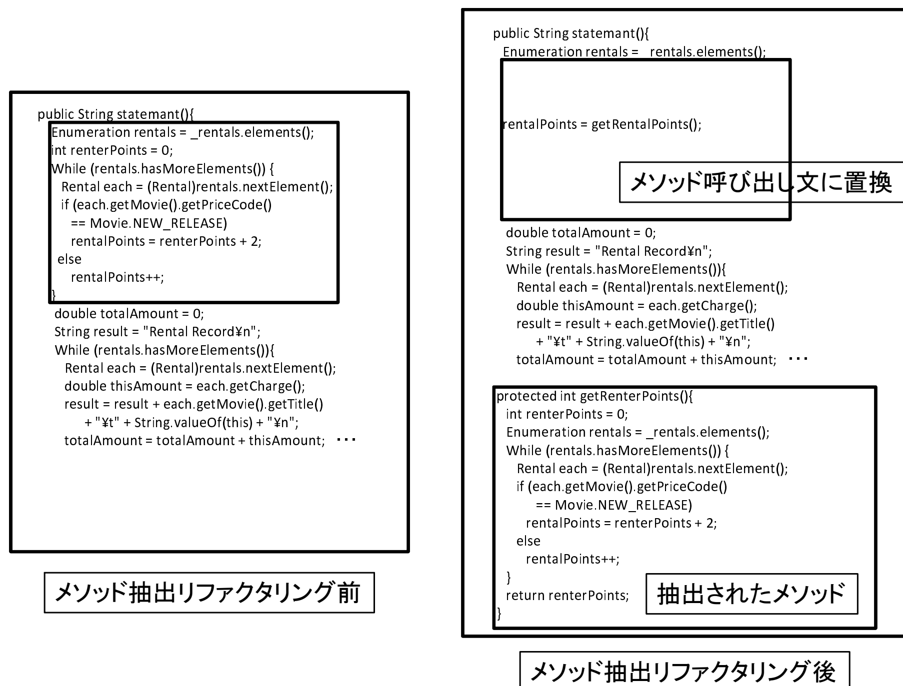


図 1: メソッド抽出リファクタリング [2]

- 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]．そして，メソッド抽出リファクタリングは効果的であり，頻繁に適用されるリファクタリングである．そのため，既存の研究においても，様々な自動化手法や Eclipse[1] のような支援ツールが提案されている [8, 9, 13].

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密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].

以下ではメソッド抽出を行う必要性がある対象の例を述べる [2, 7, 11].

長すぎるメソッド 長いメソッドほど可読性，再利用性が低下する．長すぎるメソッドはメソッド抽出によって短いメソッドに分割することが望ましい．

凝集度の低いメソッド 凝集度とはモジュール内の要素の協調度合いを示す指標である．一般に凝集度の高いモジュールは可読性，再利用性が高い．このため凝集度の低いメソッ

[1] http://eclipse.org/

ドはメソッド抽出を行い，互いに協調していない部分を分離することが望ましい．

同一の構文が連続して出現している場合 GUIの設定など，同一の構文が連続して出現している場合，同一の構文が処理する内容は類似すると考えられる．このためメソッド抽出を行い，同一の構文のブロックを一つのメソッドとすることが望ましい．

複数の文が同一のデータフローに含まれる場合 データフロー部分は1つの機能であると考えられる．(同じ変数が使われてたらそこの部分はメソッドとしてまとめたほうがよい的なことを言いたいがうまくまとめたい．) このためメソッド抽出を行い，複数のブロックが同一のデータフローに含まれる部分を分離することが望ましい．

2.3 凝集度に基づくリファクタリング支援手法

本節では，本研究で評価対象としている手法の1つである，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ついて説明する．以降では，プログラムスライス，プログラムスライスを用いたメトリクス，スライスベースの凝集度に基づくリファクタリング支援手法について述べる．

2.3.1 プログラムスライス

プログラムスライスとは，プログラム中のあるステートメントと変数の組に対して，その組に関連のあるステートメントの集合のことである [12]．次に，プログラムスライスを求めるために必要な，プログラム依存グラフとプログラムスライシングについて説明する．

プログラム依存グラフ プログラム依存グラフ（以下PDGとする）とは，プログラム中のステートメント間の依存関係を有向辺で表したグラフである．PDGの辺をたどることにより，プログラムスライスを抽出することができる．図2に例を示す．

通常の文を丸の頂点，制御文とメソッドの入口をひし形の頂点で表し，数字はその頂点が表しているプログラム中のステートメントの行番号である．依存関係にはデータ依存関係と制御依存関係が存在する．

データ依存関係 ステートメント $s1$ において変数 v が定義され，その変数が変更されることなくステートメント $s2$ に到達する経路が1つ以上存在する場合，$s1$ から $s2$ においてデータ依存関係があるという．

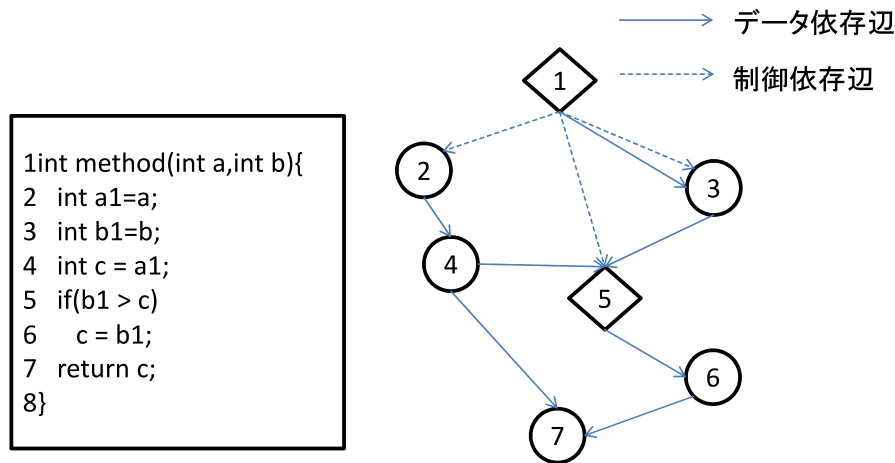


図 2: PDG の例

制御依存関係　ステートメント $s1$ が分岐 (または繰り返し) 命令であり，$s1$ の結果によってステートメント $s2$ が実行されるかが直接決まる場合，$s1$ から $s2$ に制御依存関係があるという．

プログラムスライシング　プログラムスライシングとは，PDG を用いてプログラムスライスを求める手法のことである [12]．プログラムスライシングでは基準となる文と変数の組を定め，そこから PDG 上の依存関係をたどることによって，基準となる組との依存関係があるステートメントの集合を抽出する．また，プログラムスライシングは PDG 上で基準点から依存関係をたどる方向によって前向きスライシング，後ろ向きスライシングの 2 種類がある．

前向きスライシングの例を図 3 に，後ろ向きスライシングの例を図 4 に示す． 図 3 では，ステートメントと変数の組 (2, a1) がスライシング基準であり，塗りつぶされている頂点が，前向きスライスに含まれる頂点である．前向きスライシングでは，スライシング基準となる頂点から依存関係がある頂点をスライスに加え，さらにその頂点から依存関係がある頂点を再帰的にスライスに加えていくことでプログラムスライスを求める．図 4 では，ステートメントと変数の組 (6, c) がスライシング基準であり，塗りつぶされている頂点が，後ろ向きスライスに含まれる頂点である．後向きスライシングでは，スライシング基準となる頂点への依存関係をもつ頂点をスライスに加え，さらにその頂点への依存関係をもつ頂点を再帰的にスライスに加えていくことでプログラムスライスを求める．またスライシングにはメソッドの入口を表す頂点は含めない．

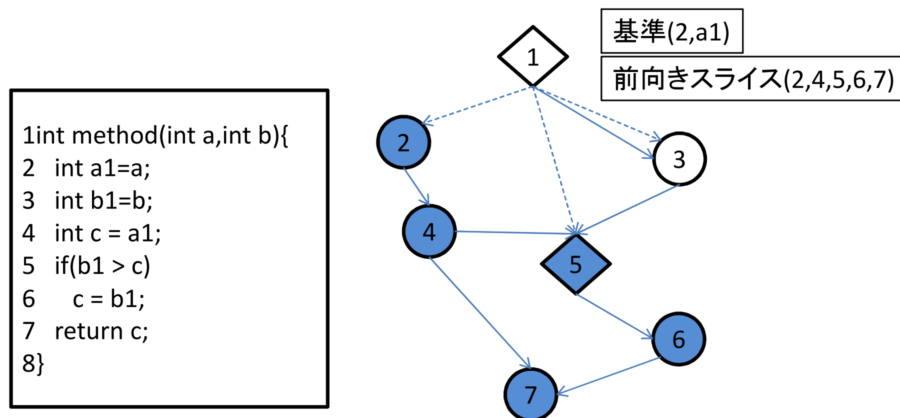


図 3: 前向きスライシング

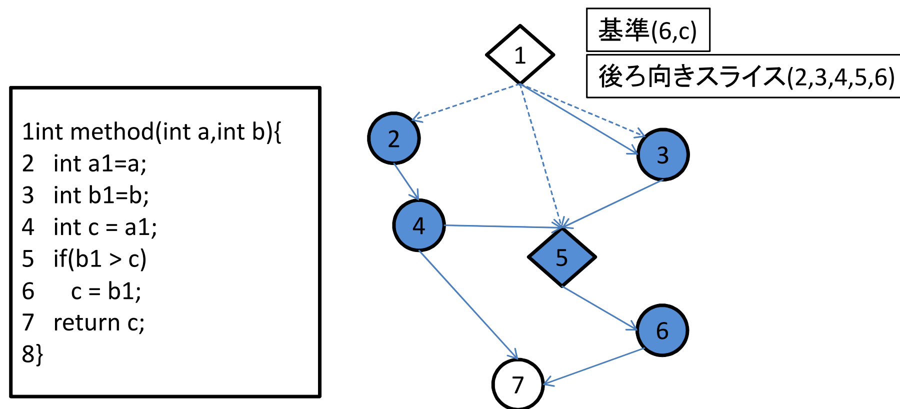


図 4: 後ろ向きスライシング

2.3.2 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 [7]．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 [15]．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 [12]．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 [6, 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．

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

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

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

Tightness はメソッド内の文の数と全スライスに含まれる文の数の比率を表す．Overlap は各スライスに含まれる文の数と全スライスに含まれる文の数の比率の平均を表す．Coverage はメソッド内の文の数と各スライスに含まれる文の数の比率の平均を表す.

Meyers らはプログラムスライスを用いたメトリクスの実験と評価を行っている [3]．実験の結果，プログラムスライスを用いたメトリクスで凝集度を測ることで，品質の低いメソッドの特定や改善に有効であることが述べられている．

2.3.3 プログラムスライスを用いた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 [14]．後藤らの手法は，2.3.2 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．

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

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

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

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，

	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}				

$$\mathrm{FTigh}tnes = 0.5$$
$$\mathrm{FOverlap} = 0.833$$
$$\mathrm{FCoverage} = 0.6167$$

図 5: 抽出されたコード片の凝集度の計算例

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 5 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 5 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る．図 5 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 5 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 5 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．

2.4 既存のメソッド抽出リファクタリング支援手法の問題点

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，処理内容を理解しなくてはならない点が挙げられる．また，メソッド抽出の際に，メソッドとして抽出しようとしている範囲のコードと，その範囲の外のコードとのデータ依存や制御依存を考慮する必要がある [4]．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，対象メソッドの処理内容を何らかの基準で判断し，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 [9, 11]．

しかし，いずれの手法も，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，手法によっては，開発者の可読性，保守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．

3　評価実験

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．

3.1　既存研究による評価方法

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]．コードクローンとはソースコード中に存在する互いに一致または類似したコード片を指す．また，互いに一致または類似したコードクローンの対をクローンペアと呼ぶ．3.1 節では既存研究の評価基準である OK 値と GOOD 値，Precision と Recall について述べる．

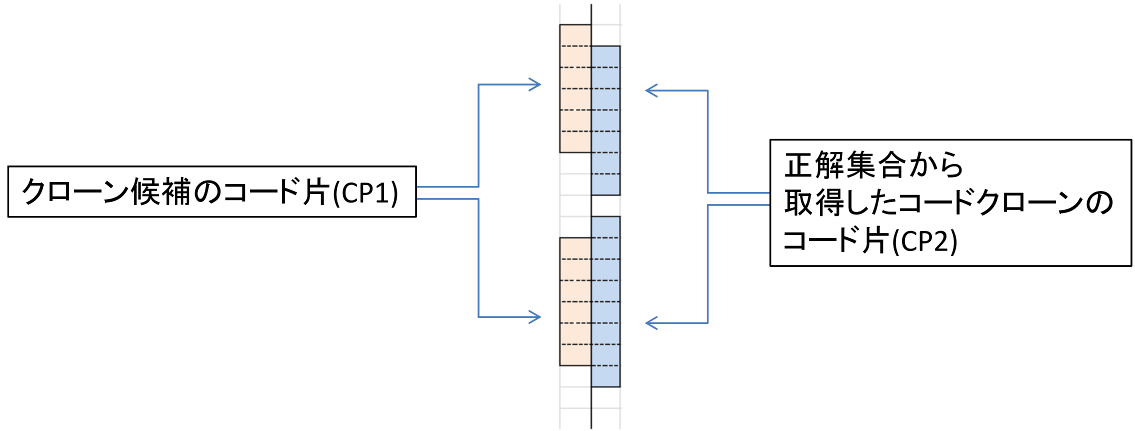
3.1.1　OK 値と GOOD 値

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る．既存研究では，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．それぞれの定義式を式 7，8 で示す．式において，CP_1 をツールを使用して得られるクローンペア，CP_2 を正解集合に含まれるクローンペア，CF_1，CF_2 をクローンペアを構成するコード片，*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とする．

$$\begin{aligned} & OK(CP_1, CP_2) \\ = \; & min(overlap(CP_1.CF_1, CP_2.CF_1), overlap(CP_1.CF_2, CP_2.CF_2)) \end{aligned} \tag{7}$$

$$\begin{aligned} & GOOD(CP_1, CP2) \\ = \; & min(max(contained(CP_1.CF_1, CP_2.CF_1), contained(CP_2.CF_1, CP_1.CF_1)), \\ & max(contained(CP_1.CF_2, CP_2.CF_2), contained(CP_2.CF_2, CP_1.CF_2))) \end{aligned} \tag{8}$$

OK 値，GOOD 値の例を図 6 に示す．図の左がツールが検出したクローンペア，右が正解集合から得られるクローンペアである．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 OK 値，GOOD 値を求める．図 6 の場合は $OK = \frac{5}{6}$，$GOOD = \frac{5}{8}$ になる．



$$GOOD(CP_1, CP_2) = min\left(\frac{5}{8}, \frac{6}{8}\right) = \frac{5}{8}$$

$$\mathrm{OK}(CP_1, CP_2) = min\left(max\left(\frac{5}{6}, \frac{5}{7}\right), max\left(\frac{6}{6}, \frac{6}{8}\right)\right) = \frac{5}{6}$$

図 6: OK 値，GOOD 値の例

3.1.2 Precision と Recall

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．それぞれの定義式を式 9，10 に示す．式において，P を対象とするプログラム，T をコードクローン検出ツール，τ をクローンの種類を表すクローンタイプ，$DetectedRefs(P,T,\tau)$ をツール T を用いて得られた OK 値 ≥ 0.7 または，$GOOD$ 値 ≥ 0.7 であるクローンペアの集合，$Cands(P,\tau)$ をコードクローン検出ツールを用いて得られたクローンペアの集合，$Refs(P,T,\tau)$ を正解集合から得られるクローンペアの集合を表す．

$$Precision(P,T,\tau) = \frac{|DetectedRefs(P,T,\tau)|}{|Cands(P,\tau)|} \quad (9)$$

$$Recall(P,T,\tau) = \frac{|DetectedRefs(P,T,\tau)|}{|Refs(P,\tau)|} \quad (10)$$

3.2　本研究での評価方法

本研究では，3.1 節で説明した評価基準を基にして，メソッド抽出支援手法の評価を行う．本研究で用いる評価の定義式を式 11，12，13，14，15，16 に示す．OK 値，GOOD 値において，CF_1 をツール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．GOODPrecision と GOODRecall において，P を対象とするプログラム，T をメソッド抽出リファクタリング支援手法，$GOODCands(P,T,x)$ をメソッド抽出リファクタリング支援手法 T を用いて得られた，正解集合内の要素 x に対して $GOOD$ 値 $\geq$ 0.7 であるメソッド抽出候補の集合，$Cands(P,T,x)$ をメソッド抽出支援手法 T 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．$GOODRefs(P,T,x)$ をメソッド抽出リファクタリング支援手法 T を用いて得られた $GOOD$ 値 $\geq$ 0.7 である Refs(P) の集合を表す.

OKPrecision と OKRecall において，P を対象とするプログラム，T をメソッド抽出リファクタリング支援手法，$OKCands(P,T,x)$ をメソッド抽出リファクタリング支援手法 T を用いて得られた OK 値 $\geq$ 0.7 であるメソッド抽出候補の集合，$OKRefs(P,T,x)$ をメソッド抽出リファクタリング支援手法 T を用いて得られた OK 値 $\geq$ 0.7 である Refs(P) の集合を表す.

$$OK(CF_1, CF_2) = overlap(CF_1, CF_2) \quad (11)$$

$$GOOD(CF_1, CF2) = max(contained(CP_1.CF_1, CP_2.CF_1), contained(CP_2.CF_1, CP_1.CF_1))) \quad (12)$$

$$GOODPrecision(P,T) = \frac{1}{|Refs(P)|} \sum_{x \in Refs(P)} \frac{|GOODCands(P,T,x)|}{|Cands(P,x)|} \quad (13)$$

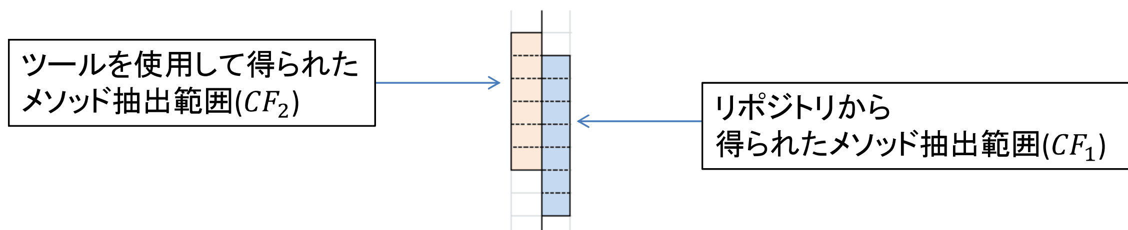
$$GOODRecall(P,T) = \frac{|GOODRefs(P,T)|}{|Refs(P)|} \quad (14)$$

$$OKPrecision(P,T) = \frac{1}{|Refs(P)|} \sum_{x \in Refs(P)} \frac{|OKCands(P,T,x)|}{|Cands(P,x)|} \quad (15)$$

$$OKRecall(P,T) = \frac{|OKRefs(P,T)|}{|Refs(P)|} \quad (16)$$

(17)

OK 値，GOOD 値の例を図 7 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



$$GOOD(CF_1 CF_2) = \frac{5}{8}$$
$$\mathrm{OK}(CF_1, CF_2) = max\left(\frac{5}{6}, \frac{5}{7}\right) = \frac{5}{6}$$

図 7: OK 値，GOOD 値の例

メントに相当する．2 つのコード片を比較して OK 値，GOOD 値を求める．図 7 の場合は $OK = \frac{5}{6}$，$GOOD = \frac{5}{8}$ になる．

3.3 実験準備

評価実験のオープンソースソフトウェアには，columba を用いた．また文献の例も用いて評価実験を行った [10]．この文献は開発者のリファクタリング作業向上を目的とした文献であり，基本となるリファクタリングが数多く掲載されているため，実験対象とした．使用したメソッドを表 1 に示す．

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

- メソッド抽出リファクタリングがリビジョン単位で行われている
- 同一クラス内で「行の削除及び行の追加が行われたメソッド」(以降，M1 と呼ぶ) と「新規追加が行われたメソッド」(以降，M2 と呼ぶ) が存在する．
- M1 の各メソッドの追加された行において M2 の呼び出しが追加されているもの

3.4 結果

実験を行い，以下のような結果が得られた．図 8 は各メソッドの行数とプロジェクト毎の行数の平均である．図 9，10 は，columba プロジェクト内のメソッド抽出リファクタリン

グの OKPrecision と OKRecall，図 11，12 は，文献内のメソッド抽出リファクタリングの OKPrecision と OKRecall のグラフである．図 13，14 は，columba プロジェクト内のメソッド抽出リファクタリングの GOODPrecision と GOODRecall，図 15,16 は，文献内のメソッド抽出リファクタリングの GOODPrecision と GOODRecall のグラフである．

これらのグラフは，メソッド抽出リファクタリング支援手法が，メソッド抽出候補であると判定するために必要な，凝集度の閾値を変化させたときの OKPrecision，OKRecall，GOODPrecision，GOODRecall の変化をプロジェクト毎に集計している．また，FTightness，FCoverage，FOverlap の 3 種類の凝集度を計測しているので，それぞれについて，OKPrecision，OKRecall，GOODPrecision，GOODRecall を求めた．x 軸は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．y 軸は各凝集度に対する OKPrecision，OKRecall，GOODPrecision，GOODRecall を示す．

3.5 考察

OK 値ベースにおける考察 OKPrecision と OKRecall を，凝集度をフィルタリングする閾値ごとの変化について考察する．2 つのプロジェクトにおいて，FOverlap はフィルタリングする凝集度の値に関係なく，開発者がよいと思うメソッド抽出候補を得ることができた．Overlap は各スライスに含まれる文の数に対する，全スライスに含まれる文の数の割合に依存するため，値が極端な値に偏りやすい．そのため，凝集度でフィルタリングした場合でも，他の 2 つの凝集度メトリクスよりもよい OKPrecision と OKRecall が得られたと考えられる．この結果より，FOverlap は明らかにメソッド抽出候補に成り得ないものを除く手法としては有効であることがわかる．また，FTightness と FCoverage はフィルタリングする閾値を上昇させると，必ずしも OKPrecision と OKRecall があがらないということが分かっ

表 1: 実験対象のメソッド

プロジェクト	クラス	メソッド
columba	MailSearchProvider	query
columba	TableModelThreadedView	createHashmap
columba	VCardParser	read
文献内の例	Matcher	Matcher
文献内の例	MatcherNo2	MatcherNo2
文献内の例	Template	template
文献内の例	Report	report

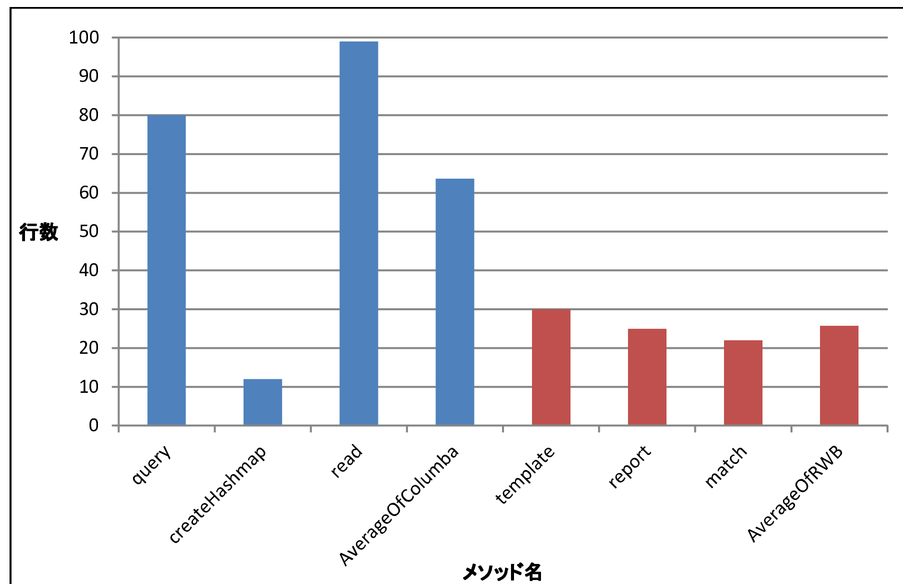
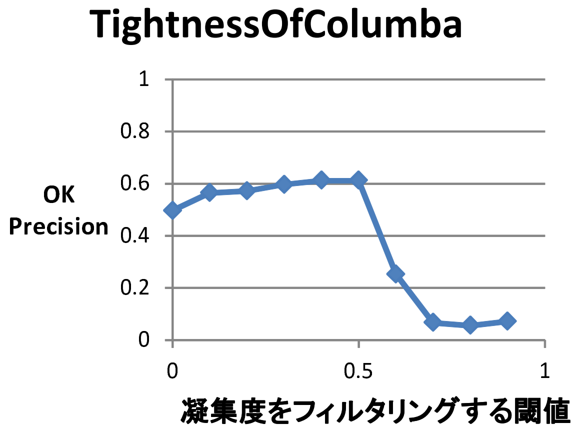
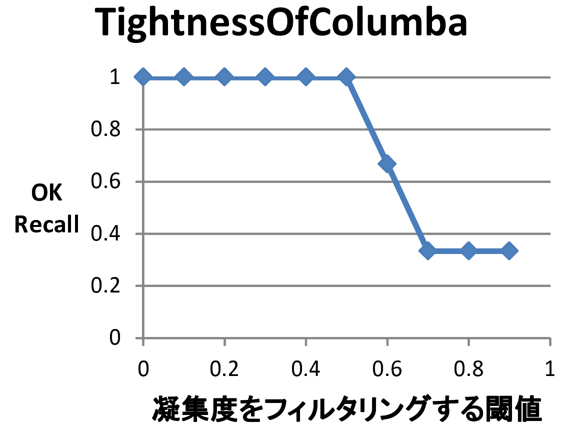


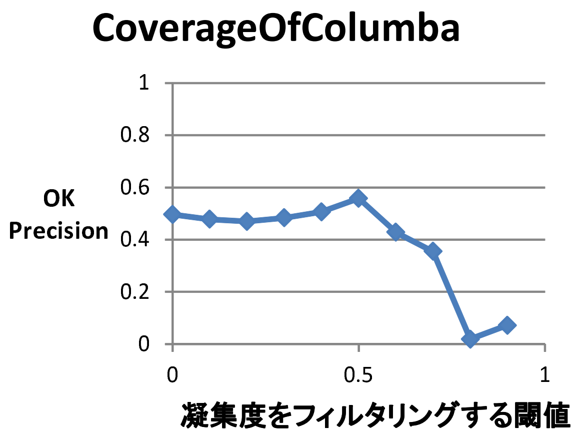
図 8: メソッドと行数の関係

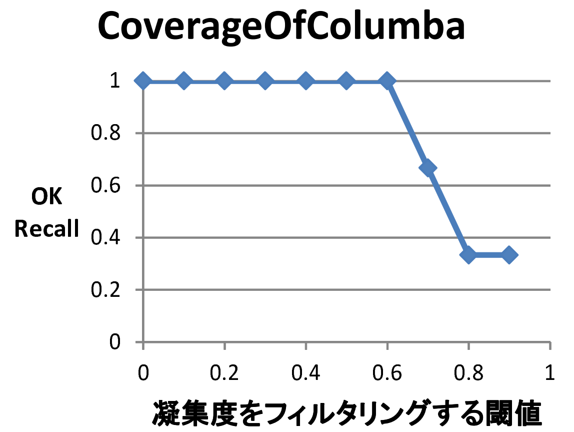
た．つまり，開発者がよいと思う候補を得るためにはフィルタリングする閾値を変化させる必要がある．FTightness と FCoverage はそれぞれ，メソッドの文の数に対する全スライスに含まれる文の数の比率，メソッドの文の数に対する各スライスに含まれる文の数の比率の平均を表し，どちらもメソッドに対するスライスの平均を凝集度の値としているため，共通したような結果が得られたと考えられる．FTightness と FCoverage は凝集度に対してフィルタリングをかけた場合，0.5〜0.6 の時，OKPrecision と OKRecall についてよい値が得られた．

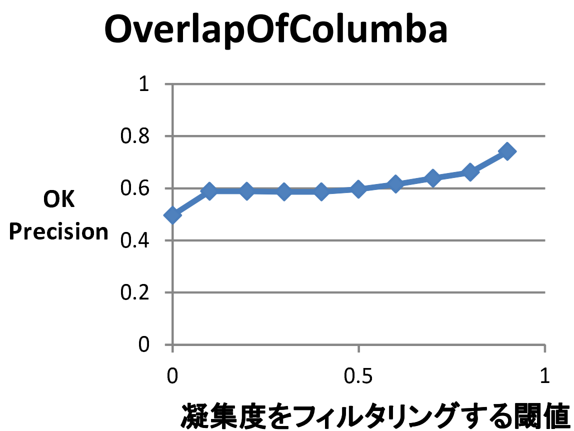
GOOD 値ベースにおける考察　GOODPrecision と GOODRecall を，凝集度をフィルタリングする閾値ごとの変化について考察する．2 つのプロジェクトにおいて，FOverlap は OK 値ベースと同様に，フィルタリングする凝集度の値に関係なく，開発者がよいと思うメソッド抽出候補を得ることができた．この結果より，FOverlap は明らかにメソッド抽出候補に成り得ないものを除く手法としては有効であることがわかる．また，FTightness も OK 値ベースと同様に，フィルタリングする閾値を上昇させると，必ずしも GOODPrecision と GOODRecall があがらないということが分かった．つまり，開発者がよいと思う候補を得るためにはフィルタリングする閾値を変化させる必要があるということが考えられる．FTightness は凝集度に対してフィルタリングをかけた場合，0.1 の時，GOODPrecision と GOODRecall について，よい値が得られた．FCoverage は GOOD 値ベースの場合，凝集度によるフィルタリングによる効果は見られなかった．











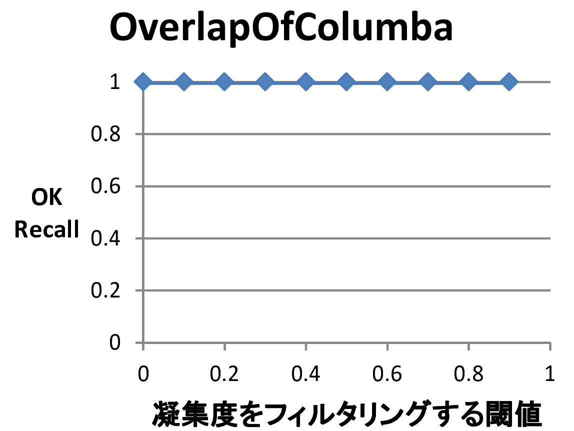
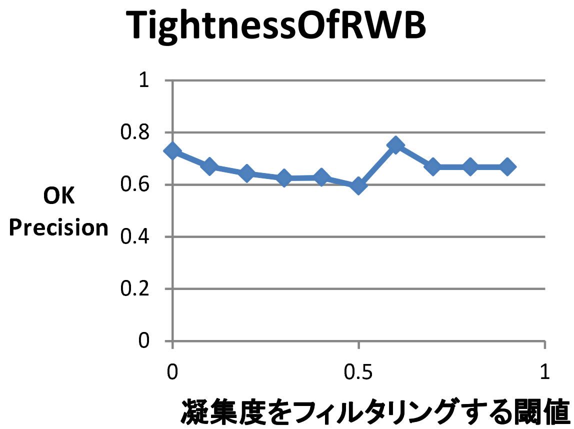
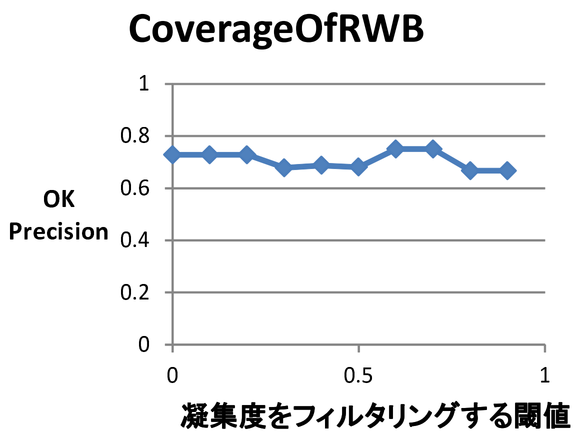


図 9: columba プロジェクトの OK 値ベースの Preicision

図 10: columba プロジェクトの OK 値ベースの Recall





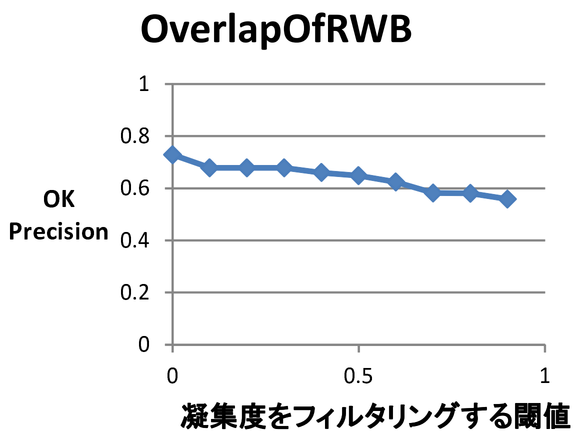
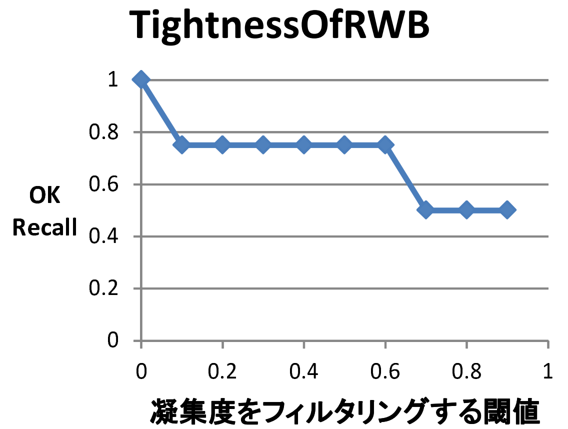
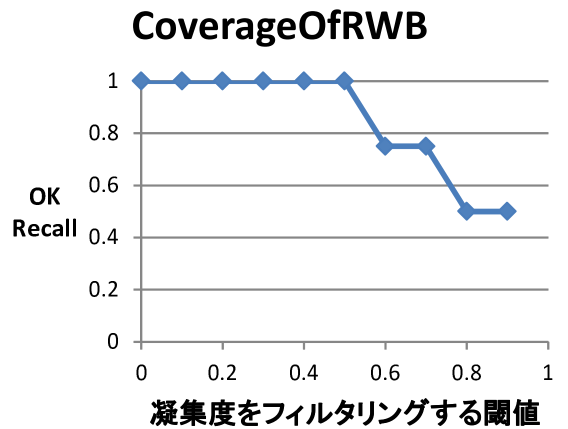


図 11: 文献の例における OK 値ベースの Precision





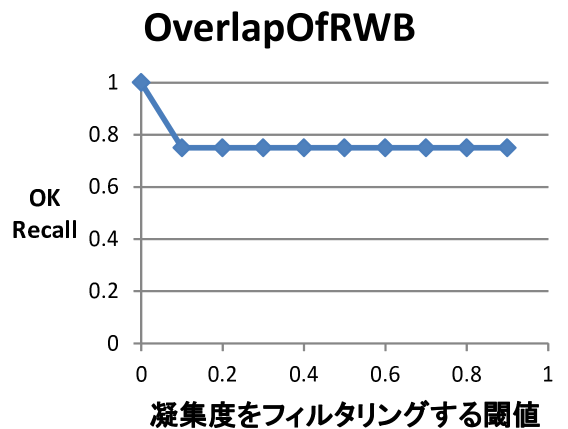
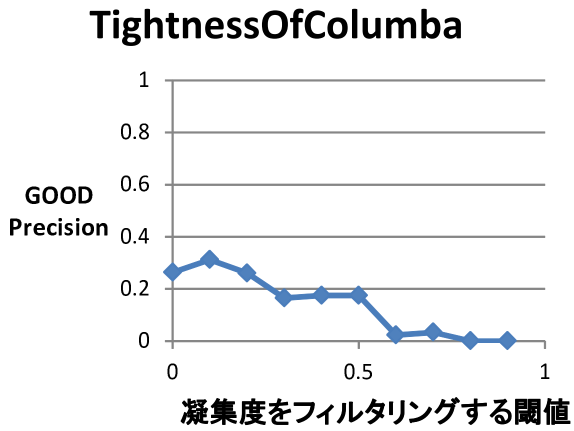
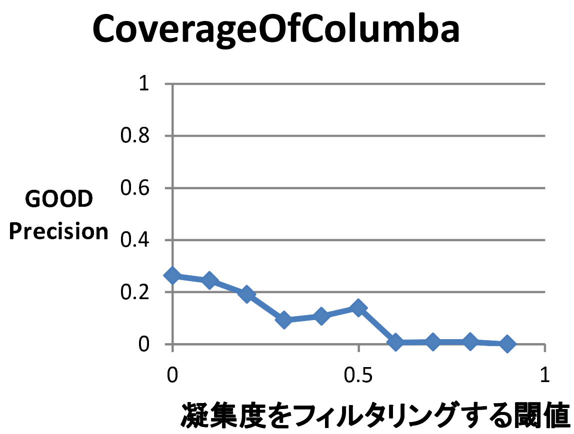


図 12: 文献の例における OK 値ベースの Recall





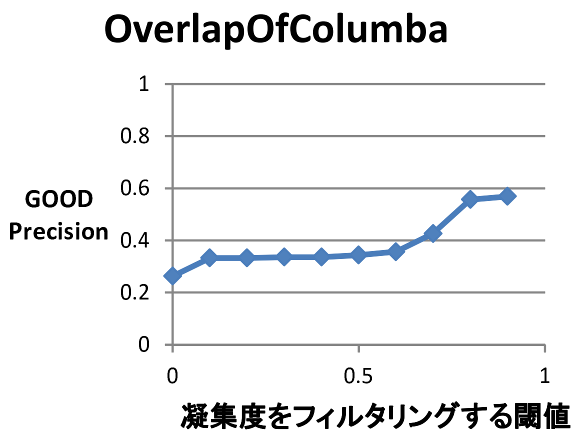
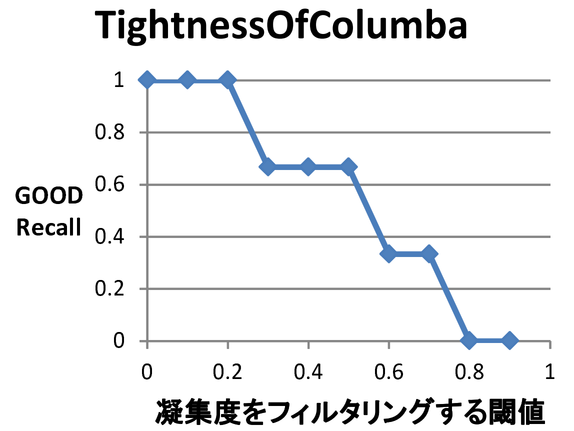
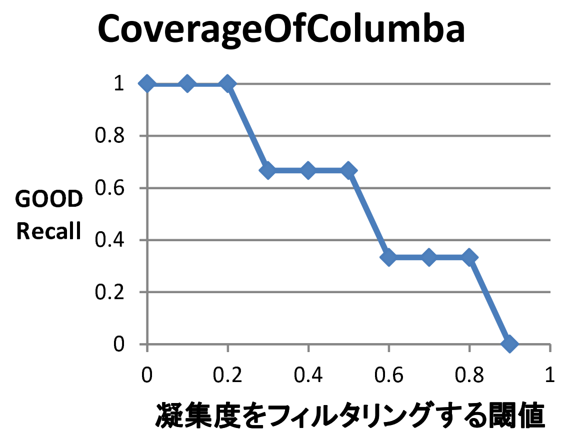


図 13: columba プロジェクトの GOOD 値ベースの Preicision





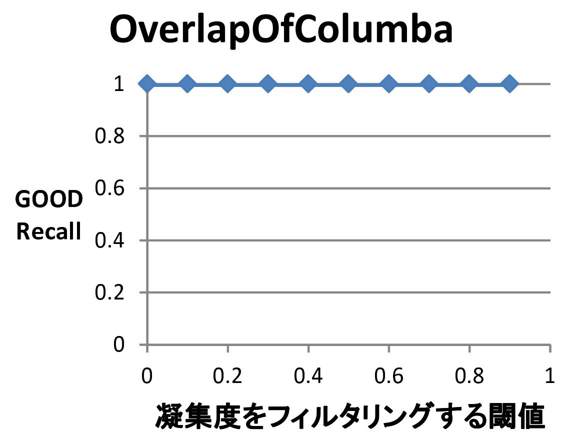
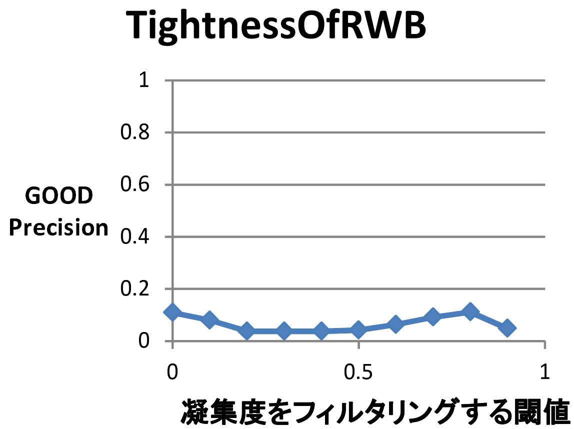
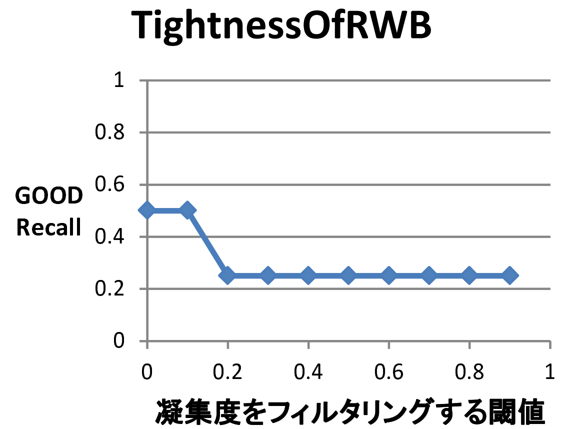
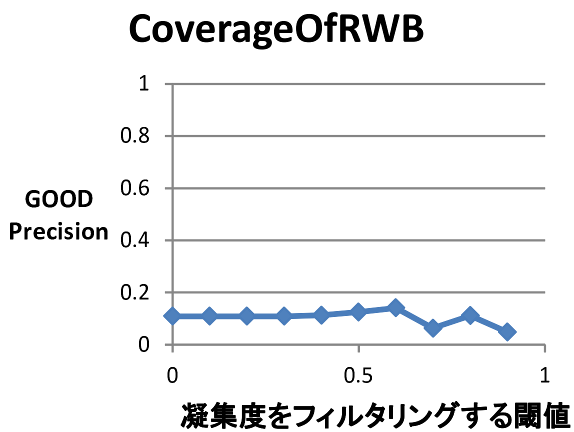
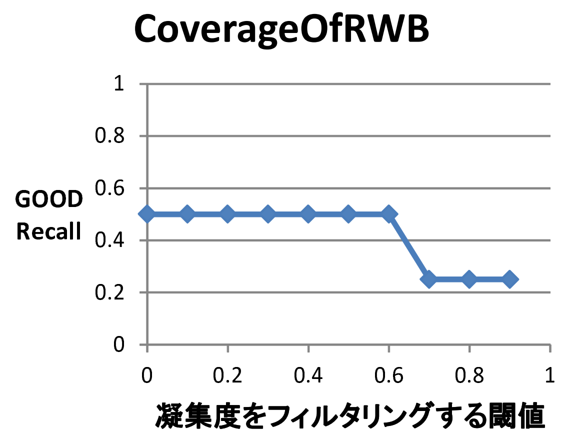


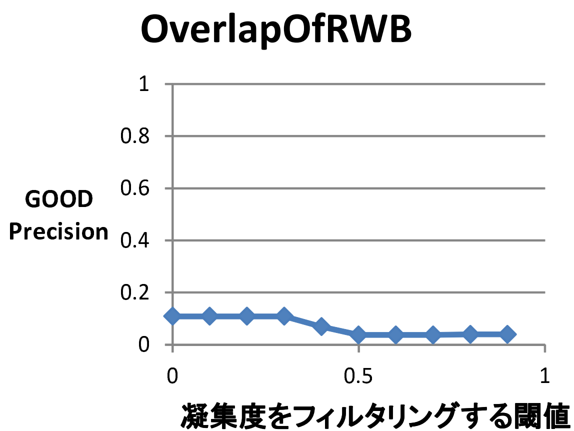
図 14: columba プロジェクトの GOOD 値ベースの Recall











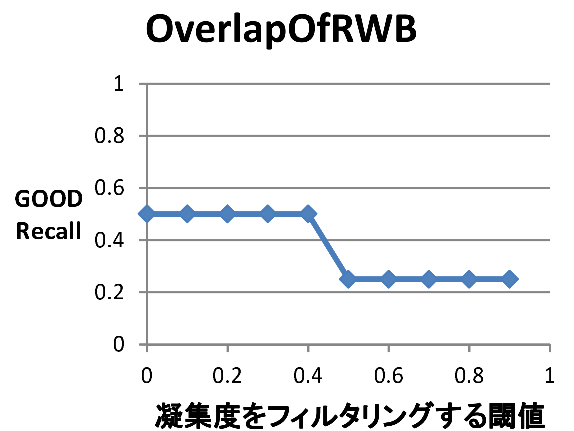


図 15: 文献の例における GOOD 値ベースの Recall

図 16: 文献の例における GOOD 値ベースの Recall

行数の考察　図 8 より columba の対象メソッドは文献の例の対象メソッドと比較して約 2.5 倍の行数がある．メソッドの行数が少ないと，メソッド抽出の候補が少なくなり，意味的なまとまりをもたない候補もメソッドの行数に応じて少なくなる．したがって，凝集度に対してフィルタリングをかけた場合でも変化があまり見られなかったと考えられる．つまり，短いメソッドに対して，凝集度をフィルタリングする閾値を変化させたとしても効果の表れる程度が少ないことが分かる．逆に，長いメソッドについては，メソッド抽出の候補が多くなり，意味的なまとまりをもたない候補も多くなる．したがって，凝集度に対して全くフィルタリングをかけない場合と比べ，フィルタリングをかけたほうが高い Precision の値を得ることができた．

4　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，有効性や，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

本研究を進めるに当たり，指導教員である井上克郎教授には，常に適切な御指導御鞭撻を賜りました．井上教授の元で研究生活を送ることができ，本論文を執筆できましたことに厚くお礼申し上げます.

松下誠准教授には問題点の提示や，疑問点に対する解決策など，多くの御助言を頂き，研究の難しさや面白さを教えていただきました.

石尾隆助教には本研究の中間報告会の際に，私が気づいていなかったような問題点を的確に指摘して頂き，本研究に対する知識を深めることができました.

奈良先端科学技術大学院情報科学研究科ソフトウェア設計学講座吉田則裕助教には本研究の構成，並びに執筆に至るまで御多忙の中，貴重な時間を割いて頂き，御指導頂きました．ソフトウェア工学の知識が乏しかった私が，本論文を執筆できたのも吉田則裕助教の御指導のおかげであると感じております.

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ソフトウェア設計学講座井垣宏特任准教授には本研究の内容，構成において熱心な御指導頂きました．教員の皆様方には感謝しきれないほどの御指導を頂きました．心から深く感謝しております.

また，井上研究室の先輩である崔 恩瀞 氏，井岡 正和 氏，後藤 祥 氏には，研究の相談，本論文の修正など親身になって対応していただきました．研究生活を有意義に過ごすことができましたのも先輩方のおかげだと思っています．本当に感謝しています.

また，本研究のためのデータ提供等のご協力や助言を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，平山力地氏に感謝しています.

最後に，楽しくも大変な研究生活の中で，寝食を共にした井上研究室の皆様に感謝の意を表すとともに，私の謝辞とさせていただきます.

参考文献

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

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

[3] T. M. Meyers and D Binkley. An empirical study of slice-based cohesion and coupling metrics. *ACM Trans. Softw. Eng. Methodol.*, Vol. 17, No. 1, pp. 2:1–2:27, December 2007.

[4] E. Murphy-Hill and Andrew P. Black. Breaking the barriers to successful refactoring: observations and tools for extract method. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pp. 421–430. ACM, 2008.

[5] 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.

[6] L.M. Ott and J.J. Thuss. Slice based metrics for estimating cohesion. *In Proceedings of the First International Software Metrics Symposium*, pp. 71–81, 1993.

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

[8] 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.

[9] N. Tsantalis and A. Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *J. Syst. Softw.*, Vol. 84, No. 10, pp. 1757–1782, October 2011.

[10] W.C. Wake. *Refactoring Workbook.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1 edition, 2003.

[11] X. Wang, L. Pollock, and K. Vijay-Shanker. Automatic segmentation of method code into meaningful blocks to improve readability. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pp. 35–44, 2011.

[12] M Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE '81, pp. 439–449, 1981.

[13] 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.

[14] 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.

[15] 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.