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.

組み合わせて実施されたリファクタリングの調査

雑賀　翼†　崔　恩瀞†　後藤　祥†　吉田　則裕††　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 番 5 号
†† 名古屋大学 大学院情報科学研究科
〒 464-8601 名古屋市千種区不老町
E-mail: †{t-saika,ejchoi,a-gotoh,inoue}@ist.osaka-u.ac.jp, ††yoshida@ertl.jp

あらまし　リファクタリングを支援するツールが研究されているが，組み合わせて実施されるリファクタリングについてはあまり研究されておらず, 既存ツールでは複数のリファクタリングをまとめて実施することができない. ソフトウェア開発履歴中のリファクタリングを対象に, 90 秒以内に連続して実施されたリファクタリングの組み合わせを調査した結果, 3 つのパターン (Move, Rename),(Rename, Rename),(Extract, Move) の実施される頻度が高いことが分かった．さらに, それらの組み合わせがどのような作業を行っているかを, リファクタリングの実施履歴の詳細から調べ, その調査結果からリファクタリングの組み合わせをまとめて実施するための, Eclipse のリファクタリング機能の改善を考案した.

キーワード　リファクタリング, ソフトウェア開発履歴, 統合開発環境, オブジェクト指向プログラミング，利用履歴分析，ソフトウェア保守

An investigation of refactoring performed in combination

Tsubasa SAIKA†, Eunjong CHOI†, Akira GOTO†, Norihiro YOSHIDA††, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1-5, Suita, Osaka 565-0871 Japan
†† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
E-mail: †{t-saika,ejchoi,a-gotoh,inoue}@ist.osaka-u.ac.jp, ††yoshida@ertl.jp

Abstract　Numerous tools that support refactoring have been proposed. However, existing refactoring support tools do not support consecutively co-occurred refactoring. This study investigated frequency on refactoring that were occurred within 90 seconds by analyzing refactoring history of data set in software projects. As a result, it turns out that refactoring pairs (Move, Rename), (Rename, Rename) and (Extract, Move) were more frequently occurred than other refactoring. Furthermore, this study investigated detailed co-occurred refactoring work and devised suggestions for improving Eclipse's automated refactoring features in order to support co-occurred refactoring based on the results of the investigation.

Key words　Refactoring, Software development history, IDE, Object-oriented programming, Usage analysis, Software maintenance

1. ま え が き

リファクタリングとは, ソフトウェアの外部から見た振る舞いを変えずに, 内部の構造を整理する作業のことである [1]. リファクタリングの主な目的は, ソースコードの可読性や保守性の向上である. また, リファクタリングを実施することで, 機能の追加や, バグの発見が容易になり, ソフトウェア開発の効率を向上させることができる [2]. しかし, 誤って実施されたリファクタリングによって, ソフトウェアに新たなバグを発生させる場合がある. バグが発生しないようにリファクタリングを実施するために, 様々なリファクタリング支援ツールが提案されている [3]〜[5].

例えば, 利用者が多い統合開発環境の Eclipse にもリファクタリング機能が備わっている. しかし, Murphy-Hill らの調査に

よると, 実際のソフトウェア開発では約 9 割のリファクタリングがツールを使わずに行われている [6]. リファクタリングの種類, プログラミング言語, 開発環境などによっては対応するツールが存在しない場合もあるが, 対応するツールがあるにも関わらず, その利用率が低い原因としては, 既存のツールが実際の開発者のリファクタリングに合っていないと考えられている.

開発者のリファクタリングに即したツールを開発するために, 複数の研究者達が開発者のリファクタリングの動向を調査してきた [6]〜[8]. Murphy-Hill らは同じ種類のリファクタリングが連続して実施されることが多いことを明らかにした [6]. このようにリファクタリングは一度にまとめて実施されることが多い一方で, 既存のツールではリファクタリングを 1 つ 1 つ実施しなくてはならない. これが Eclipse のリファクタリング機能があまり利用されない理由の 1 つであり, 連続して複数のリファクタリングを実施可能なツールを開発するべきと彼らは主張している.

一方, 異なる種類のリファクタリングがまとめて実施される頻度も高く, シームレスな移行が可能なようにツールを改善する必要がある. しかし, どの種類のリファクタリングの組み合わせが頻繁に実施されているかはまだ知られいない. 本研究では, 一度にまとめて実施される異なる種類のリファクタリングについて, どの種類の組み合わせが実施されるのかと, その頻度を明らかにする. そのために, Eclipse の利用履歴と Mylyn プロジェクトの開発履歴を用いて, リファクタリング履歴の調査を行った. まず, 連続して実施されたリファクタリングを判別し, そして, 連続して実施された異なる種類のリファクタリングの組み合わせを調査した. それから, 実施された頻度の高いリファクタリングの組み合わせについて, どのような目的で連続したリファクタリングが実施されたのかを調査した.

調査の結果, 実施された頻度が高いリファクタリングの組み合わせは, Rename と Move と Extract の組み合わせだということが分かった. これらのリファクタリングを連続して実施できるようなツールを作成することで, 開発者はより効率良くリファクタリングを実施できると考えられる.

以降,2 節では本研究の背景を説明し, 3 節では調査対象のデータセットと調査手順を説明する. 4 節ではその調査結果を説明し, 5 節でそれに対する考察を行う. 6 節では本研究のまとめと今後の課題を述べる.

2. 背　　景

2.1 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から見た振る舞いを変えずに, 内部の構造を整理する作業のことである [1]. つまり, リファクタリングではソフトウェアの機能そのものは変更せず, コードを理解しやすいようにソフトウェアの構造を変化させる. 本研究で対象としている言語は Java だが, それ以外にもオブジェクト指向言語ならばリファクタリングを実施することができ, オブジェクト指向の利点であるコードの再利用性を高めることが出来る [9].

リファクタリングの利点の一つは, ソフトウェアの設計を改善できることである. 設計の良し悪しは開発の効率にも関係するが, リファクタリングによって設計を改善することで, ソフトウェア開発の効率を向上させることがある [1]. リファクタリングによってソフトウェアの設計をいつでも改善できるため, 設計の完成を待たずにコーディングを開始することが出来る.

また, プログラムは時間が経つことで劣化する. つまり, 機能の追加やバグの修正などの保守作業によって, 本来の設計から離れたプログラムになり, 理解が難しくなってしまう. リファクタリングはコードを読みやすくし, 設計を改善することで, 機能の追加などやバグの修正などの保守作業の効率を向上させるため, リファクタリングによってプログラムの劣化を防ぐことができる [1].

2.2 Eclipse のリファクタリング機能

Eclipse はリファクタリング実施を支援する機能を提供している. この章では, Eclipse のリファクタリング機能のうち, Rename, Move, Extract について紹介する. これらのリファクタリングは実施される頻度が高いものである [10].

Eclipse のリファクタリング機能では, 基本的には個々のリファクタリングが連携していない.

a） Rename

メソッド, フィールド, ローカル変数, パッケージ, クラスなどの識別子の名前変更を行うリファクタリングで, それに合わせてその識別子に対する全ての参照も修正する必要がある.

Eclipse の Rename のリファクタリング機能では, 名前の変更と同時に全ての参照を自動的に更新するかを選択できる.

b） Move

静的なメソッドやフィールド, パッケージ, クラスなどを別のパッケージまたはクラスに移動するリファクタリングで, それに合わせてそれらに対する全ての参照も修正する必要がある.

Eclipse の Move のリファクタリング機能では, 移動先を選択するダイアログが表示され, そこで同時に全ての参照を自動的に更新するかを選択できる.

c） Extract

コードの一部を抽出して, 新しいメソッドやローカル変数, 定数フィールド, インタフェース, クラスなどとして作成するリファクタリングである. 抽出するコードの種類によって作業が大きく異なるために, 別々な種類のリファクタリングとされることが多い.

例えばインタフェースを抽出するＥｃｌｉｐｓｅのリファクタリング機能では, 抽出するコードを選択するためのダイアログを表示し, その設定に基づいてインタフェースの作成や参照による置き換えなどを自動的に行う.

2.3 関 連 研 究

Murphy-Hill らは, 利用率の低いリファクタリング支援ツールを改善することを目的として, 開発者が Eclipse のリファクタリング機能を使って実施したリファクタリングの詳細履歴を調査し, 同じ種類のリファクタリングが連続して実施されることを明らかにした [6]. 彼らの研究では, 60 秒以内に実施されたリファクタリングを, 連続して実施されたリファクタリングと見なしていた. この 60 秒という時間間隔は彼らの経験から導

かれたものである．

彼らの研究は, 同じ種類のリファクタリングが連続して実施されることを明らかにしたが, 互いに異なる種類のリファクタリングが連続して実施されるかどうかは調査されていない．異なる種類のリファクタリングが連続して実施される割合も高いのであれば, 実施される頻度の高い組み合わせについて調査するべきである．

3. 調 査 手 法

3.1 調査対象のデータセット

本研究は Users と Mylyn のデータセットを調査した．本研究で調査した 2 つのデータセットは, それぞれ異なる開発者を対象にこの Mylyn プラグインを使って記録された, Eclipse のリファクタリング機能の使用履歴である．Mylyn とは Eclipce のタスク管理プラグインで, 2009 年 11 月の機能変更までは, この Mylyn プラグインを使ってプログラマのリファクタリングの履歴を, リファクタリングを実施した時間を含めて正確に記録することが出来た．表 1 はそれぞれのデータセットの概要である．Users データセットは, 2005 年 7 月から 2005 年 9 月までの 41 人のリファクタリング実施記録である．リファクタリングの総実施回数は 3494 回で, リファクタリングの種類は 22 種類である．Mylyn データセットは, 2006 年 2 月から 2009 年 8 月までの 8 人のリファクタリング実施記録である．リファクタリングの総実施回数は 4637 回で, リファクタリングの種類は 19 種類である．以降, それぞれのデータセットについて詳しく説明する．

3.1.1 Users データセット

Users データセットは, Eclipce の利用者である 41 人の開発者の, リファクタリング履歴である．このデータセットは Gail Murphy らが, 職業や所属組織が異なる開発者を対象に, ソフトウェア開発履歴を収集したものである [10].

このデータセットが対象とした 41 人の開発者は様々な背景を持っており, リファクタリングの専門家ではないため, 一般的なソフトウェア開発の傾向が調査できると考えられる．

この Eclipse の利用履歴のデータの形式は, Mylyn のプラグインによってコミット毎に分けて作成された XML ファイルである．この中には, 開発環境の設定の変更や, エディタでの編集履歴, その他のコマンドの実施履歴などが残されている．リファクタリングの実施履歴はコマンドの実施履歴に含まれる．

本研究では, Users の全てのコマンド実施履歴から, コマンドの種類がリファクタリングと判断できるものを抽出した．表 2 で, Users データセットに実施履歴が含まれるリファクタリングの種類と, それぞれの実施された回数と割合を示す．

表 1　データセットの概要

データセット	開発者数	期間	リファクタリング総実施回数	リファクタリング種類数
Users	41	2005 年 7 月 ～2005 年 9 月	3494	22
Mylyn	8	2006 年 2 月 ～2009 年 8 月	4637	19

3.1.2 Mylyn データセット

Mylyn データセットは, Mylyn という Eclipce のタスク管理プラグインを開発している, Mylyn プロジェクトの保守履歴である．Mylyn プロジェクトでは Mylyn プラグインを使って保守作業を記録している．また, Mylyn の開発者はリファクタリングの専門家とは限らないが, 開発しているものがタスク管理プラグインであり Eclipse の機能には詳しく, 一般的な開発者よりもリファクタリングについては詳しいと推測できる．Murphy-Hill らはこのリファクタリングの実施履歴がデータベースにまとめた [6]．このデータベースには, 実施されたリファクタリング毎に, 種類と実施された日時, そしてリファクタリングの対象などの詳細情報が含まれている．本研究ではこのデータベースを使用した．

表 3 に, Mylyn データセットに実施履歴が含まれるリファクタリングの種類と実施された回数と割合を示す．

3.2 調 査 手 順

Users と Mylyn の 2 つのデータセットのリファクタリング実施履歴を対象として, 連続して実施された頻度の高い互いに異なるリファクタリングの組み合わせを明らかにするため, 以下の手順で調査を行った．

（1） 連続して実施されたリファクタリングを判断する

（2） 連続して実施された互いに異なるリファクタリングの組み合わせを調査する

（3） 実施された頻度の高いリファクタリングの組み合わせについてその作業内容を調査する

表 2　Users に含まれるリファクタリングの種類と実施回数

リファクタリングの種類	実施回数	実施割合
Rename	1992	57.0%
Extract Local Variable	449	12.9%
Extract Method	305	8.7%
Move	180	5.2%
Inline	174	5.0%
Promote Local Variable	95	2.7%
Extract Constant	59	1.7%
Modify Parameters	40	1.1%
Pull Up	36	1.0%
Convert Local To Field	29	0.8%
Convert Anonymous To Nested	29	0.8%
Introduce Parameter	25	0.7%
Extract Interface	24	0.7%
Change Method Signature	16	0.5%
Move Static Member	12	0.3%
Convert Member Type to Top Level	8	0.2%
Encapsulate Field	8	0.2%
Use Supertype	6	0.2%
Externalize Strings	4	0.1%
Change Type	1	0.0%
Generalize Type	1	0.0%
Infer Type Arguments	1	0.0%
合計	3494	100.0%

まず, 本研究ではデータセットから得られる限られた情報の中，リファクタリングが実施された時間に基づいて, 連続して実施されたリファクタリングを判別した．従って, あるリファクタリングが実施されてから, 90 秒以内に次のリファクタリングが実施されれば, それらは連続して実施されたと判断した．本研究で定めた一定時間内で行われたリファクタリングが, 開発者が一つの作業として実施したリファクタリングであるとは限らない．この問題に対しては, 実施された頻度の高いリファクタリングの組み合わせについて, 実行履歴の詳細を確認し, それらが関連性のあるものかを調べることで対応する．

次に, 連続して実施される異なる種類のリファクタリングの組み合わせの調査を行った．全ての連続して実施されたリファクタリングについて, 組み合わせ毎に実施された回数を調査した．ただし, 同じ種類の組み合わせでも実施された順序が異なれば別の組み合わせとした．そして, 頻度の高いリファクタリングの組み合わせについて, 支援方法を考察するために, 詳細な作業内容を調査した．ただし, Users のリファクタリングの実施履歴にはリファクタリングの対象などの情報が含まれていなかったため, 作業内容の調査は Mylyn のリファクタリング履歴に対してのみ行った．具体的な調査方法は, 実施される頻度の高い上位 10 個のリファクタリングの組み合わせについて, 実施された履歴の詳細を各 10 個ずつ調査し, リファクタリングの対象と, その前後に実施されたリファクタリングを調べた．そして, 対象が同じもの, 関連するもの, 関連がわからないものに分けた．関係の有無については同じパッケージやクラスに属するか, 対象の名前が類似しているかで判断した．

表 3　Mylyn のリファクタリングの種類と実施回数

リファクタリングの種類	実施回数	実施割合
Rename	2401	51.8%
Move	691	14.9%
Extract Method	305	6.6%
Extract Local Variable	254	5.5%
Extract Constant	245	5.3%
Move Static Member	235	5.1%
Change Method Signature	191	4.1%
Inline	110	2.4%
Convert Member Type to Top Level	44	0.9%
Infer Type Arguments	30	0.6%
Extract Interface	29	0.6%
Encapsulate Field	29	0.6%
Convert Anonymous To Nested	23	0.5%
Extract Superclass	16	0.3%
Promote Local Variable	15	0.3%
Pull Up	14	0.3%
Push Down	3	0.1%
Use Supertype	1	0.0%
Introduce Parameter	1	0.0%
合計	4637	100.0%

4. 調 査 結 果

4.1 節では 2 つのデータセットについて, 連続して実施された頻度の高い, 互いに異なる種類のリファクタリングの組み合わせを調査した結果を示す．4.2 節では Mylyn データセットに含まれるリファクタリングの詳細な履歴から, 頻度の高いリファクタリングの組み合わせについて支援方法を考察するために, リファクタリングの組み合わせの作業内容について調査した結果を示す．

4.1　頻度の高い異なる種類のリファクタリングの組み合わせ

表 4 が Users の調査結果で, 表 5 が Mylyn の調査結果である．表には実施されたものを回数の多い順に 10 組のリファクタリングを示している．また, この表の欄のリファクタリング 1 は組み合わせの中で先に実施されたリファクタリングを表し, リファクタリング 2 がその次に実施されるリファクタリングを表している．3 種類以上のリファクタリングが連続して実施される回数は 5 回未満であった．

Rename, Move, Extract は実施される頻度の多いリファクタリングで, それらの種類のリファクタリングを含めた組み合わせが多いことが分かる．

また, 2 つのデータセットで出現する回数の順位は異なっても, 同じような組み合わせが多いことがわかる．順位が異なるのは, どのリファクタリングが多く実施されるかが主に影響していると考えている．例えば, Users の方では Extract Local Variable, Extract Method, Inline などの実施回数が多く, Mylyn の方では Move, Extract Interface, Extract Constant などの実施回数が多い．

表 6 は, Mylyn について Rename を対象の種類で分けた場合の, 連続して実施される頻度の高い異なる種類のリファクタリングの組み合わせである．表には実施されたものを回数の多い順に 10 組のリファクタリングを示している．対称の種類による分類とはつまり, Move をクラスやパッケージの移動, Move Static Member を静的なフィールドの移動と分けている．また, Rename もローカル変数, フィールド, メソッドなどに対象で分けている．Rename Type というのはクラスの名前変更である．ここでは, 異なる種類の対象への Rename と Rename の組み合わせを, 異なる種類のリファクタリングとして扱っている．表 6 から, Rename は対象の要素の種類に関係なく, 関連する要素にまとめて実施される場合が多いことがわかる．

4.2　頻度の高いリファクタリングの組み合わせの作業内容

Mylyn データセットで, 連続して実施された回数が多い順に, 10 個の互いに異なる種類のリファクタリングの組み合わせについて, その作業内容をリファクタリング実施履歴の詳細から調査した．表 7 は, 作業内容を調査したリファクタリングの組み合わせと, それぞれのリファクタリングが互いに関連するかどうかを示したものである．ただし, 組み合わせの順序を無視し, Move と Rename を対称の種類で分類した場合とする．

実施回数はその組み合わせが実施された数を, 調査数はリファクタリングの実施履歴の詳細を調べた数を表している．同じ対象とは, 調査数のうちリファクタリングの対象が同じであった

表 7　頻度の高いリファクタリングの組み合わせの作業内容

リファクタリング 1	リファクタリング 2	実施回数	調査数	同じ対象	関係あり
Move	Rename Type	58	10	4	1
Rename Field	Rename Method	48	10	0	5
Rename Type	Rename Method	32	10	0	4
Move	Rename Package	29	10	1	7
Rename Type	Rename Field	26	10	0	3
Move Static Member	Rename Field	15	10	6	0
Extract Interface	Move	14	10	7	1
Rename Local Variable	Rename Field	12	10	0	6
Move	Move Static Member	12	10	0	2
Move	Rename Field	12	10	0	0

数である. 当然, リファクタリングの対象の種類が異なる場合には, 同じ対象になることはない. 関係ありとは, リファクタリングの対象に関係がありそうなものの数である. 関係の有無については同じパッケージやクラスに属するか, 名前が類似しているかで判断した.

以降, 各リファクタリングの組み合わせについてその作業内容を説明する.

（1） (Move, Rename) の作業内容

Move と Rename の組み合わせの作業内容から, ある要素を移動したとき, それに合わせて名前を変更する場合が多いことが判明した. Move と Rename の対象が同じ場合だと, パッケージ, クラス, フィールドなどを移動した後に, その移動した要素の名前を変更していた. また, Move と Rename の対象が関係する場合だと, クラス, フィールド, メソッドなどの移動の後に, その宛先のパッケージやクラスの名前を変更していた.

（2） (Rename, Rename) の作業内容

異なる種類の要素を対象とした, Rename と Rename の組み合わせの作業内容から, Rename は対象の種類に関係なく, 関連する要素を対象に連続して実施される場合が多いことが判明した. また, 連続して変更される名前は単語が共通するなど類似していることが多い. 例えば, ローカル変数名とフィールド名の Rename では, コンストラクタや getter, setter のローカル変数名と, それに対応するフィールドの名前を連続して変更していた.

（3） (Extract Interface, Move) の作業内容

Extract と Move の組み合わせの作業内容から, インタフェースを抽出した後に、それを適当なパッケージに移動することが多いことが判明した.

表 4　Users の連続して実施された頻度の高い互いに異なる種類のリファクタリングの組み合わせ.

リファクタリングの種類		
リファクタリング 1	リファクタリング 2	実施回数
Extract Method	Rename	35
Extract Local Variable	Rename	22
Extract Local Variable	Extract Method	19
Rename	Extract Method	17
Rename	Move	15
Extract Method	Inline	14
Extract Local Variable	Inline	14
Inline	Extract Local Variable	14
Extract Local Variable	Promote Local Variable	12
Move	Rename	12

表 5　Mylyn の連続して実施された頻度の高い互いに異なる種類のリファクタリングの組み合わせ.

リファクタリングの種類		
リファクタリング 1	リファクタリング 2	実施回数
Move	Rename	88
Rename	Move	62
Extract Interface	Move	12
Extract Constant	Rename	11
Extract Constant	Move	10
Extract Method	Rename	9
Extract Method	Extract Local Variable	6
Extract Local Variable	Rename	6
Rename	Change Method Signature	5
Inline	Rename	5

5. 考　　察

本調査の結果, 連続して実施される互いに異なる種類のリファクタリングのうち, 実施される頻度の高い組み合わせがわかった. これらの頻度の高い組み合わせに対して行った作業内容の調査をもとに, 連続して実施されるリファクタリングの支援方

表 6　Move と Rename を対称の種類で分類した場合の, Mylyn の連続して実施された頻度の高い互いに異なる種類のリファクタリングの組み合わせ.

リファクタリングの種類		
リファクタリング 1	リファクタリング 2	実施回数
Rename Type	Move	30
Move	Rename Type	28
Rename Field	Rename Method	26
Rename Method	Rename Field	22
Move	Rename Package	19
Rename Type	Rename Method	16
Rename Method	Rename Type	16
Rename Type	Rename Field	13
Rename Field	Rename Type	13
Extract Interface	Move	12

法を考える.

以降, 現状の Eclipse のリファクタリング機能の仕様上の, 連続して実施する際の問題点を解消する支援方法を考案する.

（1） (Move, Rename) の支援

現状の Eclipse のリファクタリング機能の仕様では, Move と Rename のリファクタリングは連携していないため, それぞれ別々に実施するとダイアログが 2 回も表示されることになる. この組み合わせに対する支援方法としては, まず Move と Rename の対象が同じ場合には, Move のダイアログに新しい名前を入力する項目を追加して, まとめて実施できるようにすることが考えらる. また Move と Rename の対象が関係する場合には, Move の後に, 関係する要素へ Rename を推薦することが考えられる. ただし, 一度に複数のクラスを Move するときがあるので, その場合にはそれぞれのクラスに対して個別に名前を変更できるようにするべきである.

（2） (Rename, Rename) の支援

この組み合わせに対する支援方法としては, ある要素を Rename するとき, 名前が類似した要素をまとめて Rename できるようにすることが考えられる. 現状の Eclipse のリファクタリング機能の仕様では, クラスを対象とした Rename のダイアログでは, 類似の名前の要素をまとめて Rename できる. しかし, フィールドやメソッドなどの Rename にはそのような支援がないため, 同様に類似の名前の Rename をまとめてできるようにすることが考えられる.

（3） (Extract Interface, Move) の支援

Extract Interface のダイアログで Java ファイルを作る場所を指定できるようにすれば, 続けて Move する手間は無くなると考えられる. ただし, 現状の Eclipse のリファクタリング機能の仕様では, Java ファイルの Move はパッケージエクスプローラーからドラッグ＆ドロップを用いて実施出来るため, この組み合わせについては現状の Eclipse の機能のみで十分な支援だ出来ているとも考えられる.

6. まとめと今後の課題

本研究では, 互いに異なる種類のリファクタリングが高い頻度で組み合わせて実施されると推測して, ソフトウェア開発履歴中の, 連続して実施された頻度の高いリファクタリングの組み合わせを調査した. ただし, リファクタリングが 90 秒以内に続けて実施された場合を連続と見なした. 調査の結果, 連続して実施された頻度の高いリファクタリングの組み合わせは, (Move,Rename),(Rename,Rename),(Extract,Move) であることが判明した. そして, それらのリファクタリングの組み合わせについて, その作業内容を調査し, その結果から一度にまとめて実施するための Eclipse のリファクタリング機能の改善を考案した. 今後の課題としては, 多くの Bad-smell を含む古いソフトウェア・プロジェクトと良く保守されたソフトウェア・プロジェクトの間で, 連続して実施されるリファクタリングを比較する予定である. Bad-smell とは, リファクタリングを実施することが推奨されるコードの不吉な臭いのこと [1] であり, Bad-smell の組み合わせ [11], あるいはコードクローンの量 [12] が, 連続して実施されるリファクタリングの数を増加させるかどうかを明らかにしたい. また, ソフトウェア産業でのリファクタリングの実態調査 [13] で, どのようなリファクタリングが連続して実施されるかにも興味がある. そして, 本研究の考案に基づいてリファクタリング支援ツールを開発する予定である.

謝辞 本稿の執筆に際して，貴重なご意見をくださった大阪大学 大学院情報科学研究科 春名 修介 特任教授に感謝いたします．本研究は JSPS 科研費 25220003 の助成を受けたものです.

文　献

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

[2] G. Bavota, B.D. Carluccio, A.D. Lucia, M.D. Pentaand, R. Oliveto, and O. Strollo, “When does a refactoring induce bugs? an empirical study,” Proc. of SCAM, pp.104–113, 2012.

[3] S.R. Foster, W.G. Griswold, and S. Lerner, “Witchdoctor: IDE support for real-time auto-completion of refactorings,” Proc. of ICSE, pp.222–232, 2012.

[4] X. Ge, Q.L. DuBose, and E. Murphy-Hill, “Reconciling manual and automatic refactoring,” Proc. of ICSE, pp.211–221, 2012.

[5] R. Tairas and J. Gray, “Increasing clone maintenance support by unifying clone detection and refactoring activities,” Inf. Softw. Technol., vol.54, no.12, pp.1297–1307, 2012.

[6] E. Murphy-Hill, C. Parnin, and A.P. Black, “How we refactor, and how we know it,” IEEE Trans. Softw. Eng., vol.38, no.1, pp.5–18, 2012.

[7] E. Choi, N. Yoshida, and K. Inoue, “An investigation into the characteristics of merged code clones during software evolution,” IEICE Trans. Inf. & Syst., vol.E97-D, no.5, 2014. to appear.

[8] E. Murphy-Hill and A.P. Black, “Breaking the barriers to successful refactoring: observations and tools for extract method,” Proc. of ICSE, pp.421–430, 2008.

[9] L. Tokuda and D. Batory, “Evolving object-oriented designs with refactorings,” Proc. of ASE, vol.8, no.1, pp.89–120, 2001.

[10] G.C. Murphy, M. Kersten, and L. Findlater, “How are java software developers using the Eclipse IDE?,” IEEE Softw., vol.23, no.4, pp.76–83, 2014.

[11] A. Yamashita and L. Moonen, “Exploring the impact of inter-smell relations on software maintainability: an empirical study,” Proc. of ICSE, pp.682–691, 2013.

[12] T. Kamiya, S. Kusumoto, and K. Inoue, “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.

[13] M. Kim, T. Zimmermann, and N. Nagappan, “A field study of refactoring challenges and benefits,” Proc. of FSE, pp.1–11, 2012.