特別研究報告

題目

開発履歴中の連続して実施されたリファクタリングの分析

指導教員
井上 克郎 教授

報告者
雜賀　翼

平成 26 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

平成 25 年度 特別研究報告

開発履歴中の連続して実施されたリファクタリングの分析

雑賀　翼

内容梗概

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. リファクタリングはソフトウェア開発で広く使われており, 様々なツールによる支援が研究されている. しかし, 現状のリファクタリング支援ツールでは, 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ず, リファクタリングを一つ一つ実施しなくてはいけない.

この問題を解決するために, 複数のリファクタリングの組み合わせを支援するツールが必要である. しかし, このツールを実現するためには, 優先して支援するリファクタリングの組み合わせを明らかにする必要がある.

そこで本研究では, 複数のリファクタリングをまとめて支援するツールの開発を目的とし, 実際のソフトウェア開発履歴中の, リファクタリングが実施された履歴を調査の対象として, 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るかを調査した. 本研究では, ツールで支援するべきリファクタリングの組み合わせとして, 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. 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て, その調査結果から支援手法を考案した.

調査の結果, 連続して実施された頻度の特に高いリファクタリングの組み合わせは, (Move, Rename), (Rename, Rename), (Extract, Move) であった. 特に, (Move, Rename) の組み合わせに関しては, 移動した要素の名前を移動先に合わせて変更するという作業内容が多かった. この作業を支援方法する方法としては, Move した要素には Rename を推奨する, または, Move と同時に Rename を実施できるようにすることが考えられる.

主な用語

リファクタリング
ソフトウェア開発履歴
統合開発環境

目 次

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 [4]. 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 [1]. ただし, リファクタリングの方法を誤ると, 既に機能しているソフトウェアに新たなバグを発生させることがある.

リファクタリング支援ツールを用いることで, 人の手によるミスを減らすことができるため, 多くの研究者がリファクタリング支援ツールの使用を推奨している [3, 5, 13]. 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 [12]. 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, 既存のツールが使いにくいということが考えられている.

使いやすいリファクタリングのツールを開発するために, 複数の研究者達がプログラマのリファクタリングの動向を調査してきた [2, 11, 12]. Murphy-Hill らの調査で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されている [12]. 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組み合わせは, Rename と Move と Extract のリファクタリング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行うことができると考

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を説明する. 4 節では調査手法を説明し, 5 節でその調査結果を示す. 6 節では本研究のまとめと今後の課題を述べる.

2　背景

この節では, まずリファクタリングがどのような技術であるかを説明し, Eclipse のリファクタリング機能の紹介をする. そして, 既存研究である, Murphy-Hill らの連続して実施される同じ種類のリファクタリングに関する研究を紹介する.

2.1　リファクタリング

リファクタリングを一言で説明すると, コードを整理する作業である. 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行うことができ, 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 [14].

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 [4]. 最初からソフトウェアの設計を完璧なものとするのは困難だが, 後からリファクタリングによってソフトウェアの設計が改善できるため, 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, コードは理解し辛く本来の設計から離れたものになってしまう. 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 [4].

リファクタリングを行うべき状況の例としては, 重複したコードが 2 箇所以上に存在する場合が分かりやすい. このようなソースコード中の同一, あるいは類似したコードの断片のことは, コードクローンと呼ばれる [7]. リファクタリングはコードクローンを含むソースコードを改善するのに有効な手段である. コードクローンのリファクタリングに要する時間は短く, その後の修正回数を削減する効果がある [6].

2.2　Eclipse のリファクタリング機能

Eclipse のリファクタリング機能を利用するには, まず, エディタまたはビューからリファクタリングの対象となるプログラム要素を選択し, その状態からいくつかの方法でリファクタリングを開始する. ただし, ここでのビューはパッケージ・エクスプローラーやアウトラインなど, プログラム要素が表示され, 選択可能なものに限られる. リファクタリングを開始する方法の一つは図 1 に示した, Eclipse のメニューバーにあるリファクタリングのメニュー

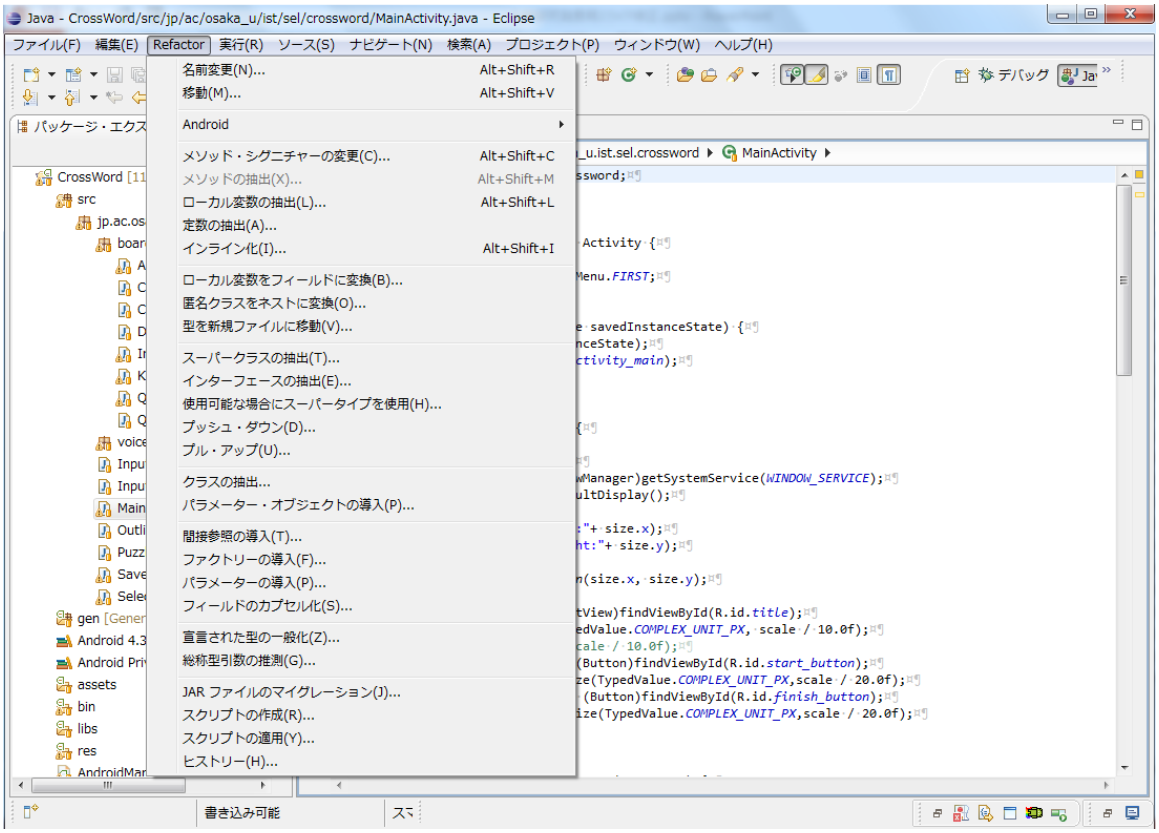

図 1: Eclipse のメニューバーにあるリファクタリングのメニュー

から, 実施するリファクタリングの種類を選択することである. また他の方法としては, 図 2 に示した, 右クリックで表示されるコンテキストメニューにも, リファクタリングのサブメニューがあり, そこからリファクタリングを開始できる. 他には, 一部のリファクタリングの種類は, 対応するショートカットキーを押すことでもリファクタリングを開始できる.

以降, 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, 実施される頻度の特に高いものである [10].

Rename 識別子の名前変更を行うリファクタリングで, その識別子に対する全ての参照も修正される. メソッド, フィールド, ローカル変数, パッケージ, クラスなどのほとんど全ての識別子の名前を変更することができる.

ビュー上のプログラム要素を対象にして Rename を開始すると, 新しい名前を入力するためのダイアログが表示される. このダイアログは対称の要素の種類に応じて異なり, 図 3 に示すのはクラスを対象にした Rename で表示されるダイアログで, 類似する名前のものをまとめて Rename することが出来る. さらに, 図 4 のようなダイアログを開いて, 類似の名前

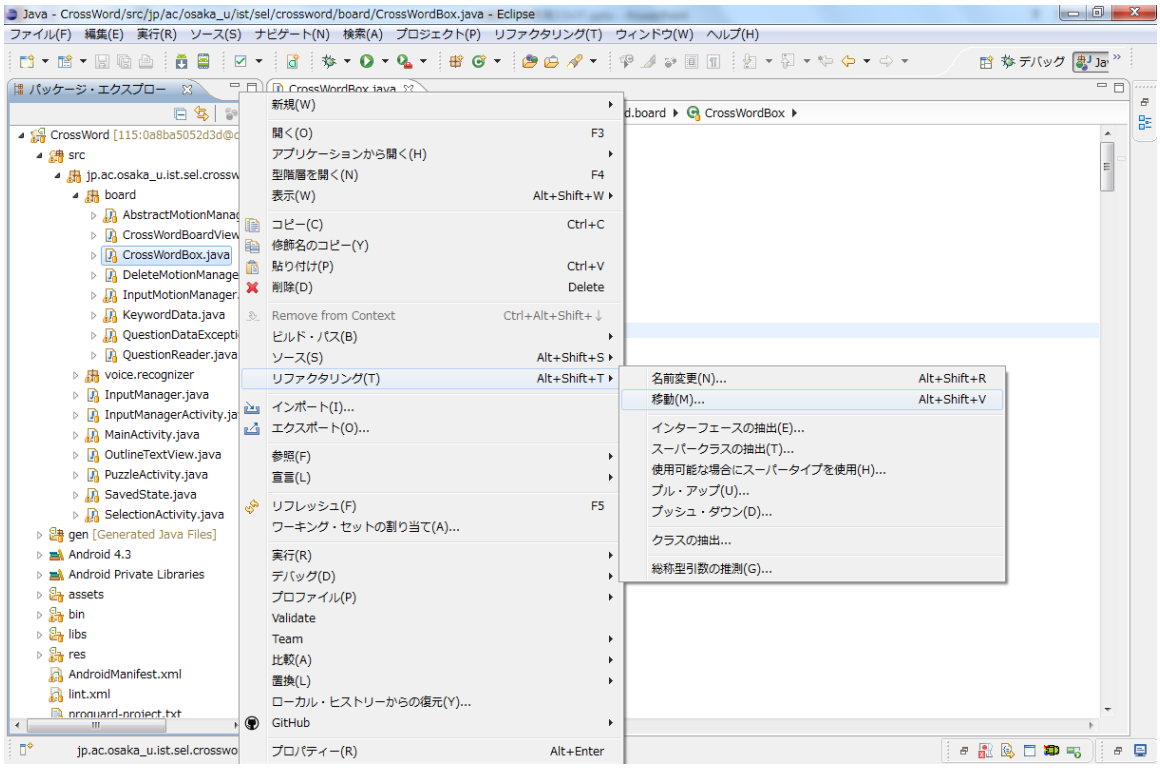

図 2: Eclipse のコンテキストメニューにあるリファクタリングのサブメニュー

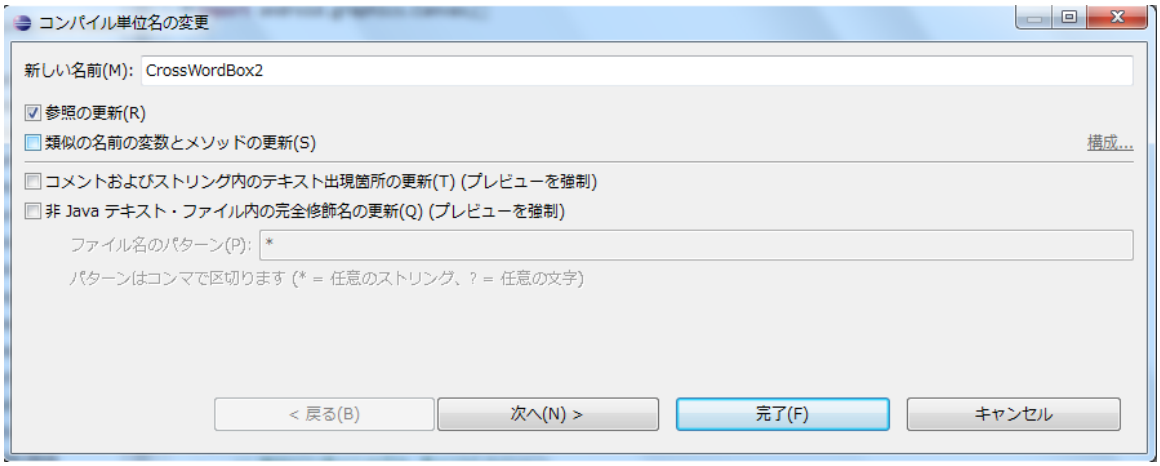

図 3: クラスを対象とした Rename のダイアログ

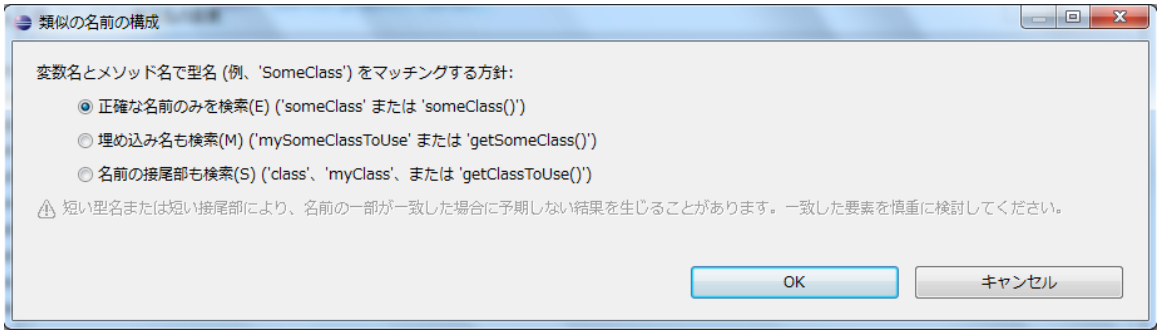

図 4: 類似の名前の要素を検索する条件を設定するダイアログ

の要素を検索する条件を設定できる. 一方, フィールドやメソッドなどの Rename から類似の名前のクラスやフィールド, メソッドの Rename することはできない. また, エディタ上のプログラム要素を対象にして Rename を開始すると, 図 5 のようにダイアログを表示せずにエディタ上で新しい名前を入力できる.

Move　プログラム要素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, その要素に対する全ての参照も修正される. 静的なメソッドやフィールド, パッケージ, クラスなどの要素が対象とできる. パッケージエクスプローラでドラッグ&ドロップを用いてクラスやフィールドなどを Move することが出来る. しかし, パッケージはドラッグ&ドロップでは同じフォルダ内の別のパッケージの中へと Move することができず, 元のパッケージの下にそのパッケージのコピーを作るという動作になっている.

Move のリファクタリングを行うときにはクラスへの参照を変更するためのダイアログが

図 5: Eclipse のエディタ上での Rename

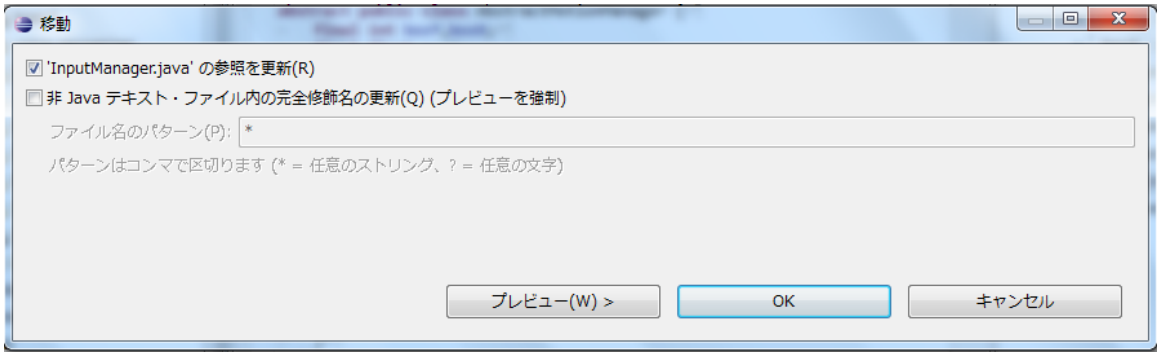

図 6: ドラッグ＆ドロップで実施した場合の, クラスを対象とした Move のダイアログ

表示される. パッケージエクスプローラーでドラッグ＆ドロップして実施した場合では図 6 のようなダイアログが表示される. リファクタリングメニューから実施した場合では, 同時に移動先も入力する必要があり, 図 7 のようなダイアログが表示される. 一方, 複数のクラスをまとめて移動することが可能で, その場合のダイアログの表示は 1 回のみである. また, 移動先に同名のクラスがあった場合は, 移動するもので上書きするか移動をキャンセルするかの 2 択となる.

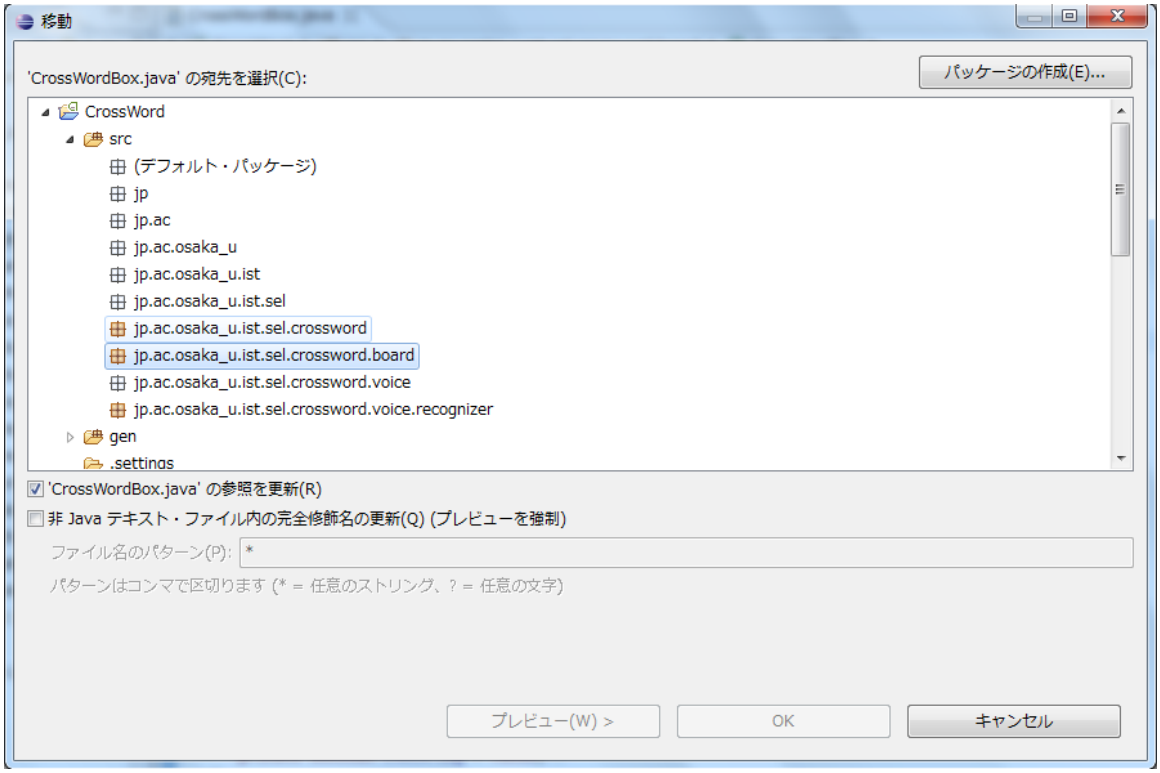

図 7: メニューから実施した場合の, クラスを対象とした Move のダイアログ

Extract　コードの一部を抽出して, 新しいプログラム要素として作成するリファクタリングである. 抽出する要素の種類によって作業が大きく異なるため, 別々な種類のリファクタリングとされることが多い. どの種類の要素に対する Extract でも, 抽出の方法を設定するダイアログが表示される. 以下, 抽出する要素の種類に分けて紹介する. これらの他にもクラスの抽出などがある.

- Extract Method
  複雑なメソッドの一部を抽出し, 別のメソッドとすることで整理する.
- Extract Local Variable
  式の代わりとなる新しいローカル変数を作成し, その参照で置き換える.
- Extract Constant
  数値や文字列の代わりとなる新しい静的な定数フィールドを作成し, その参照で置き換える.
- Extract Interface
  クラスのメソッドから新しいインタフェースを作成し, 元のクラスをその実装クラスとする.

2.3 連続して実施された同じ種類のリファクタリング

Murphy-Hill らの研究 [12] は, 開発者のリファクタリング履歴を調査し, 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. この研究では, Toolsmiths と Mylyn, Users という 3 つの開発履歴を対象に調査を行っている. これらの履歴は, Eclipse のリファクタリング機能を使って実施されたリファクタリングの詳細を記録している. それぞれのデータセットの概要を以下に示す.

- Toolsmiths : Eclipse のリファクタリング機能の開発者 4 人を対象とした, 2005 年 12 月から 2007 年 8 月までの開発履歴で, リファクタリングの総実施回数は 2331 回である.
- Mylyn : Mylyn という Eclipse のプラグインの開発者 8 人を対象とした, 2006 年 2 月から 2009 年 8 月までの開発履歴で, リファクタリングの総実施回数は 5048 回である.
- Users : 様々な背景を持った 41 人の開発者を対象とした, 2005 年後半の開発履歴で, リファクタリングの総実施回数は 3027 回である.

Murphy-Hill 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している. 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. 時間間隔は連続するリファクタリングの調査の結果に大きく影響することが判明している. 図 8 は, Murphy-Hill らの論文 [12] の p.8 の図である. 彼らの調べたそれぞれのデータセットについて, 連続して実施された同じ種類のリファクタリングの割合が, 時間間隔の設定によってどのように変化したかを示したものである. 横軸は連続かどうかを判定するための時間間隔である. 縦軸は時間間隔毎の, データセット中の全てのリファクタリングに対する, 連続して実施された同じ種類のリファクタリングの割合である. 赤色の線が Toolsmiths, 青色の線が Mylyn, 紫色の線が Users のデータセットの結果をそれぞれ示している. 彼らの調査に使われた時間間隔である 60 秒のところには縦線が引かれており, 4 割前後のリファクタリングが同種のものと連続して実施されたことがわかる. また, データセットによって割合が大きく異なることがわかる.

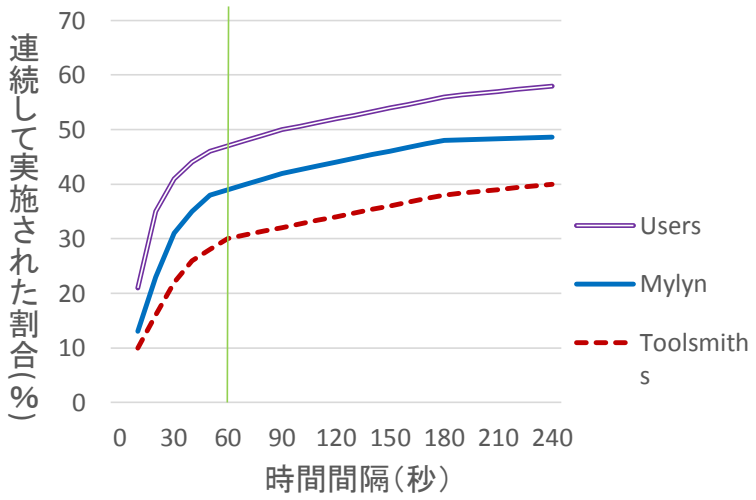

図 8: 同じ種類のリファクタリングが連続して実施された割合 [12]

表 1 は, Murphy-Hill らの論文 [12] の p.7 の表の一部を抜粋したものである. それぞれのデータセットについて, リファクタリングの種類毎に実施された割合と, 同じ種類のものと連続して実施された割合を示したものである. ただし, 横線で区切られた下の 4 つのリファクタリングの種類は, Toolsmiths と Mylyn の履歴の期間の途中から記録され出した種類で, Users の履歴の期間ではその種類が記録されていなかったため「*」としている. また, 連続する割合が「-」となっているのは, その種類のリファクタリングが実施されたのが 1 回以下であることを表す. この表から開発者によってリファクタリングの傾向が異なるのがわかる. 例えば, どのデータセットでも Rename の実施された割合が最も高いが, User では 61.5%であるのに対して, Toolsmiths では 28.7%と差が大きい. これは, Toolsmiths の開発者は Eclipse のリファクタリング機能の保守を行っているため, 当然ながらリファクタリングに詳しく, かつツールを熟知しているため, Rename のような良く知られているリファクタリングだけでなく, 他の種類のリファクタリングも頻繁に実施するためと考えられる. 一方, User の開発者は一般的な Eclipse の利用者であるため, 結果はリファクタリングのツールの作成の経験がない人の傾向を表していると考えられる. Mylyn の開発者は Eclipse のプラグインの開発を行っていることから, Toolsmiths の開発者ほどではなくとも, Users の開発者よりは, Eclipse のリファクタリング機能に詳しいと推測できるので, Toolsmiths と User の間くらいの結果となっていることが納得できる.

表 1: リファクタリングの実施された割合と同種のものと連続して実施された割合 [12]

	Toolsmiths		Mylyn		Users	
リファクタリングの種類	実施割合	連続割合	実施割合	連続割合	実施割合	連続割合
Rename	28.7%	42.2%	53.6%	42.4%	61.5%	54.2%
Extract Local Variable	24.4%	22.4%	5.2%	21.9%	10.6%	32.9%
Inline	15.0%	37.8%	2.2%	47.3%	4.5%	38.0%
Extract Method	12.0%	10.0%	6.3%	8.5%	8.6%	22.0%
Move	6.3%	34.0%	19.0%	47.9%	5.6%	57.3%
Change Method Signature	4.0%	28.0%	3.8%	38.2%	1.8%	36.4%
Convert Local To Field	3.9%	13.0%	0.4%	45.5%	0.9%	37.0%
Introduce Parameter	1.8%	48.8%	0.1%	-	0.5%	68.8%
Extract Constant	0.9%	27.3%	5.5%	32.7%	2.7%	59.3%
Convert Anonymous To Nested	0.8%	0.0%	0.5%	0.0%	0.6%	36.8%
Move Member Type to New File	0.6%	0.0%	1.1%	7.3%	0.4%	41.7%
Pull Up	0.5%	0.0%	0.3%	11.8%	1.2%	11.1%
Encapsulate Field	0.5%	72.7%	0.6%	58.6%	0.1%	50.0%
Extract Interface	0.1%	0.0%	0.6%	6.9%	0.5%	0.0%
Generalize Declared Type	0.1%	0.0%	0.0%	-	0.1%	50.0%
Push Down	0.1%	-	0.1%	66.7%	0.1%	-
Infer Generic Type Arguments	0.0%	-	0.6%	41.9%	0.1%	0.0%
Use Supertype Where Possible	0.0%	-	0.1%	-	0.1%	0.0%
Introduce Factory	0.0%	-	0.0%	-	0.1%	-
Extract Superclass	0.3%	0.0%	0.3%	0.0%	*	*
Extract Class	0.1%	0.0%	0.0%	-	*	*
Introduce Parameter Object	0.0%	-	0.0%	-	*	*
Introduce indirection	0.0%	-	0.0%	-	*	*
計	100.0%	29.7%	100.0%	38.7%	100.0%	47.3%

しかし, 彼らの研究では, 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支援するべきである.

3　調査手法

この節では本研究の調査手法を説明する. まず, 調査対象のデータセットの説明を行い, その次に, 調査の手順を説明する.

3.1　調査対象のデータセット

ソフトウェア開発履歴を記録するツールには, Mylyn という Eclipce のタスク管理プラグインがある. 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. 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. 表 2 はそれぞれのデータセットの概要である. Users のデータセットは, 開発者数は 41 人で, データの期間は 2005 年 7 月から 2005 年 9 月である. 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 のデータセットは, 開発者数は 8 人で, データの期間は 2006 年 2 月から 2009 年 8 月である. 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 のデータセット

一つ目のデータセットは, 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 らの研究 [10] で使われたもので, 職業や所属組織が異なる開発者を対象に, 開発履歴を収集している.

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. 表 3 は G. Murphy らの論文 [10] の p.77 の表である. 開発者の背景, 具体的には開発者の職業の分類と, 所属する組織の規模および分類を, 人数の割合で示している.

表 2: データセットの概要

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

表 3: 開発者の分類（G. Murphy, 2006, p.77）

職業	割合
アプリケーション開発者	65%
研究者	13%
アプリケーション設計者	12%
マネージャー/CIO/CTO	4%
その他	6%

組織規模	割合
個人	19%
職員が 50 人未満	32%
職員が 50 人から 500 人	26%
職員が 500 人より多い	23%

部門	割合
ソフトウェア製造	48%
大学・研究所	19%
金融/小売	13%
報道/交通/ネットワーク	7%
政府	5%
その他	8%

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．表 4 は Rename の実施履歴の一例である．XML のタグ<interactionEvent>の中身が一つの動作の履歴のまとまりであり, この表はそこに含まれる XML のタグとその中身を示している．まず, タグ<kind>は履歴の種類を表していて, これがコマンドの実施履歴だと分かる．その下の, <date>と<endDate>がコマンドの実施された時刻を表している．その次の<originId>がコマンドの実施されたユーザインタフェースを示していて, これを見て実施されたコマンドの種類を判断することが出来る．この例では, item.label.Refactor はリファクタリングのメニューのことで, そこから Rename が選択されたことが分かる．また, Alt+Shift+T と Alt+Shift+R はそれぞれ, リファクタリングメニューを開くのと, Rename を選択を行うショートカットキーである．残念ながらどのリファクタリングの履歴にも, そのリファクタリングの対象とされたものに関する情報は含まれておらず, 前後の<interactionEvent>からその情報を得ることも出来なかった．

調査のため 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 5 で, Users の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 4: Rename の実施履歴の例

タグ	タグの中身
kind	command
date	2005-07-13 10:58:52.258 PDT
endDate	2005-07-13 10:58:52.258 PDT
originId	item.label.Refactor Alt+Shift+T/Rename... Alt+Shift+R
structureKind	null
structureHandle	null
navigation	null
delta	menu
interestContribution	1

表 5: 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%

3.1.2 Mylyn のデータセット

二つ目のデータセットは, Mylyn という Eclipce のタスク管理プラグインを開発している, Mylyn プロジェゥトのソフトウェア開発履歴である. 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 らの研究 [12] で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られており, 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 6 に, Mylyn の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. また, Mylyn データセットではリファクタリングの対象が記録されているため, Rename と Move, Inline リファクタリングを, 対象とした要素の種類毎に分割することができる. その場合のリファクタリングの種類と実施された回数と割合を表 7 で示す.

表 6: 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%

表 7: Mylyn の Rename と Move, Inline を対称の種類で分類した場合の, 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 Type	817	17.6%
Rename Method	719	15.5%
Move	684	14.8%
Rename Field	543	11.7%
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%
Rename Local Variable	119	2.6%
Rename Package	90	1.9%
Rename Resource	95	2.0%
Inline Local Variable	58	1.3%
Convert Member Type to Top Level	44	0.9%
Inline Constant	42	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%
Rename Enum Constant	13	0.3%
Inline Method	10	0.2%
Move Resource	6	0.1%
Rename Compilation Unit	5	0.1%
Push Down	3	0.1%
Use Supertype	1	0.0%
Introduce Parameter	1	0.0%
Move Method	1	0.0%
合計	4637	100.0%

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

まず, データセットのリファクタリング実施履歴から, 連続して実施されたリファクタリングを判別する. ただし, データセットから得られる情報は限られており, リファクタリングが実施された時間のみから判断しなければならない. 従って, Murphy-Hill らの研究 [12] と同様に, あるリファクタリングが実施されてから, 一定時間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する. 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.

また, リファクタリングが連続して実施されたかを判断する時間間隔を決定するため, 時間間隔の設定が調査にどのような影響を及ぼすのかを調べる必要がある. 時間間隔を決定するために, それぞれのデータセットについて, リファクタリングが連続して実施される割合が時間間隔の設定によってどのように変化するかを調べた. 連続して実施されるリファクタリングの種類を, 同じ種類のみに限定した場合と, 異なる種類も含めた場合で調べた. これにより, 時間間隔による影響の大きさ, 変化の仕方を知ることができる. また, 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頻度を知ることができる. 他にも, 連続して実施されたリァクタリングの種類や, リファクタリングが連続して実施された数が, 時間間隔によってどのように変化するかを調べた. 特定のリファクタリングの種類が, ある時間間隔では連続して実施されず, それよりも少し長い時間間隔では連続して実施されれば, 最初の時間間隔はそのリファクタリングを実施するのには時間が足りないと考えられる.

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 10 から 15 個見て, 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.

4　調査結果

4.1　時間間隔の変化による影響

4.1.1　リファクタリングが連続して実施される割合

２つのデータセットについて, データセット中のリファクタリングが連続して実施された割合を調査した結果を, 調査で設定した時間間隔毎のグラフとして図 9 に示す.

グラフの横軸がリファクタリングが連続して実施されたかを判断する時間間隔で, 縦軸がリファクタリングが連続して実施された割合である. 縦線は 60 秒と 90 秒, 120 秒を示している. Users のデータを紫色で, Mylyn のデータを青色で示している. 破線が連続して実施された同種のリファクタリングのみの場合で, 実線がそれに加えて連続して実施された互いに異なる種類のリファクタリングも含めた場合である. 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た割合は, どちらのデータセットでも全体の 10%から 15%であり, 同じ種類のリファクタリングと同様に, 支援が必要であると考えられる.

Users の結果と Mylyn の結果を比較すると, 同じ種類のみの場合でも異なる種類を含めた場合でも, User の方が割合は全体的に約 10%高い. また, 時間間隔による変化は似ているので, 時間間隔はどちらも同じに設定して良いと考えられる.

連続して実施されたリファクタリングをなるべく多く抽出するべきであるが, 時間間隔があまりに長いとほとんどのリファクタリングが, 連続して実施されたと判断されてしまう. 互いに関連の無いリファクタリングが, 連続して実施されたと判断されてしまう問題は, リファクタリングの実施履歴の詳細を調べることで対処するが, 長すぎず短すぎずの時間間隔に設定にするべきである. 60 秒より短い時間間隔では割合の変化が激しいため, リファクタリングを実施するのに十分ではないと推測される. 以降, 60 秒と 90 秒, 120 秒の 3 つの時間間隔を比較し, これらの中から時間間隔を決定する.

4.1.2　連続して実施されたリファクタリングの種類

表 8 は Users の, 表 9 は Mylyn のデータセット中の各リファクタリングの種類について, 60 秒, 90 秒, 120 秒の時間間隔毎に連続して実施された回数と連続する割合を調べたものである. また, Mylyn のデータセットで Rename と Move, Inline を対象の種類で分類した場合が表 10 である. 連続回数とは, その種類のリファクタリングが連続して実施された回数である. 連続割合とは, その種類のリファクタリングが実施された回数に対する, 連続して実施された回数の割合である. 表のリファクタリングの種類の順番は, 連続して実施された回数が多いほど上になるように並べているが, 時間間隔が変わっても, その順番はほとんど変わっていない. つまり, リファクタリングの種類と時間間隔による変化が関係していない. もし,

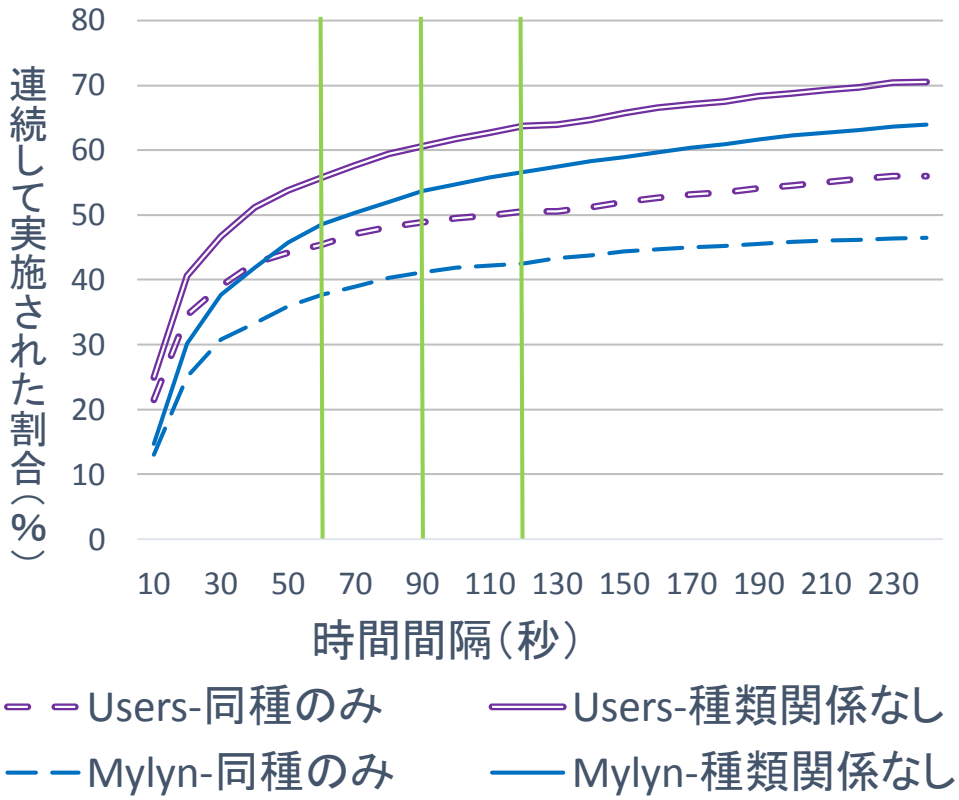

図 9: リファクタリングが連続して実施される割合

ある種類のリファクタリングを適用するのに十分な時間間隔でなければ, そのリファクタリングが連続して実施された数が極端に少なくなると推測できるが, 時間間隔が 60 秒でもそのようなことにはなっていない.

4.1.3 一度に連続して実施されたリファクタリングの数

表 11 は Users の, 表 12 は Mylyn のデータセット中で一度に連続して実施されるリファクタリングの数を, 60 秒, 90 秒, 120 秒の時間間隔毎に示したものである. 表を見ると, 2 つのリファクタリングが連続して実施された数が多く, 連続するリファクタリングの数が多いほど実施された数は少ないことがわかる. 実施された数の多い, 連続するリファクタリングの数が少ないところでは, 時間間隔による変化は僅かしかないため, 全体としても大きくは変化しないと考えられる.

以上より, 60 秒と 90 秒, 120 秒の 3 つの時間間隔では, 連続して実施されたリファクタリングの傾向は, 大きくは変わらないと推測できる. 3 つの時間間隔の中でどれが最も調査に適しているか, 優劣を定めることは出来ないため, 中間である 90 秒を, 連続して実施された組み合わせの調査での時間間隔とする.

表 8: Users のリファクタリングの種類毎の連続して実施された回数と割合の時間間隔による変化

	時間間隔					
	60 秒		90 秒		120 秒	
リファクタリングの種類	連続回数	連続割合	連続回数	連続割合	連続回数	連続割合
Rename	1138	57.1%	1209	60.7%	1265	63.5%
Extract Local Variable	203	45.2%	229	51.0%	244	54.3%
Extract Method	134	43.9%	169	55.4%	182	59.7%
Move	125	69.4%	128	71.1%	131	72.8%
Inline	118	67.8%	125	71.8%	129	74.1%
Promote Local Variable	70	73.7%	71	74.7%	74	77.9%
Extract Constant	31	52.5%	33	55.9%	35	59.3%
Modify Parameters	24	60.0%	29	72.5%	32	80.0%
Pull Up	19	52.8%	21	58.3%	24	66.7%
Convert Anonymous To Nested	16	55.2%	19	65.5%	19	65.5%
Convert Local To Field	15	51.7%	19	65.5%	21	72.4%
Introduce Parameter	22	88.0%	22	88.0%	22	88.0%
Extract Interface	9	37.5%	15	62.5%	16	66.7%
Change Method Signature	6	37.5%	8	50.0%	11	68.8%
Move Member Type to New File	3	25.0%	3	25.0%	5	41.7%
Convert Nested To Top	4	50.0%	4	50.0%	4	50.0%
Encapsulate Field	5	62.5%	5	62.5%	5	62.5%
Use Supertype	6	100.0%	6	100.0%	6	100.0%
Externalize Strings	0	0.0%	0	0.0%	0	0.0%
Change Type	1	100.0%	1	100.0%	1	100.0%
Generalize Type	1	100.0%	1	100.0%	1	100.0%
Infer Generic Type Arguments	0	0.0%	0	0.0%	0	0.0%

表 9: Mylyn のリファクタリングの種類毎の連続して実施された回数と割合の時間間隔による変化

	時間間隔					
	60 秒		90 秒		120 秒	
リファクタリングの種類	連続回数	連続割合	連続回数	連続割合	連続回数	連続割合
Rename	1267	52.8%	1394	58.1%	1463	60.9%
Move	377	54.0%	431	61.7%	455	65.2%
Move Static Member	165	70.2%	169	71.9%	179	76.2%
Extract Constant	98	40.0%	105	42.9%	110	44.9%
Change Method Signature	82	42.9%	92	48.2%	99	51.8%
Extract Local Variable	66	26.0%	84	33.1%	95	37.4%
Inline	63	57.3%	64	58.2%	66	60.0%
Extract Method	51	16.7%	61	20.0%	63	20.7%
Encapsulate Field	19	65.5%	20	69.0%	21	72.4%
Convert Member Type to Top Level	18	40.9%	20	45.5%	21	47.7%
Extract Interface	15	51.7%	15	51.7%	17	58.6%
Infer Type Arguments	13	43.3%	14	46.7%	15	50.0%
Promote Local Variable	10	66.7%	10	66.7%	11	73.3%
Convert Anonymous To Nested	4	17.4%	6	26.1%	8	34.8%
Push Down	3	100.0%	3	100.0%	3	100.0%
Pull Up	2	14.3%	3	21.4%	3	21.4%
Introduce Parameter	1	100.0%	1	100.0%	1	100.0%
Extract Superclass	0	0.0%	0	0.0%	1	6.3%
Use Supertype	0	0.0%	0	0.0%	0	0.0%

表 10: Mylyn のの Rename と Move, Inline を対称の種類で分類した場合の, リファクタリングの種類毎の連続して実施された回数と割合の時間間隔による変化

	時間間隔					
	60 秒		90 秒		120 秒	
リファクタリングの種類	連続回数	連続割合	連続回数	連続割合	連続回数	連続割合
Rename Type	422	51.7%	460	56.3%	484	59.2%
Move	371	54.2%	423	61.8%	447	65.4%
Rename Method	359	49.9%	401	55.8%	414	57.6%
Rename Field	322	59.3%	349	64.3%	363	66.9%
Move Static Member	165	70.2%	169	71.9%	179	76.2%
Extract Constant	98	40.0%	105	42.9%	110	44.9%
Change Method Signature	82	42.9%	92	48.2%	99	51.8%
Extract Local Variable	66	26.0%	84	33.1%	95	37.4%
Rename Local Variable	62	52.1%	70	58.8%	74	62.2%
Rename Package	53	58.9%	63	70.0%	73	81.1%
Extract Method	51	16.7%	61	20.0%	63	20.7%
Inline Constant	39	92.9%	40	95.2%	40	95.2%
Rename Resource	37	38.9%	38	40.0%	41	43.2%
Inline Local Variable	20	34.5%	20	34.5%	22	37.9%
Encapsulate Field	19	65.5%	20	69.0%	21	72.4%
Convert Member Type to Top Level	18	40.9%	20	45.5%	21	47.7%
Extract Interface	15	51.7%	15	51.7%	17	58.6%
Infer Type Arguments	13	43.3%	14	46.7%	15	50.0%
Promote Local Variable	10	66.7%	10	66.7%	11	73.3%
Rename Enum Constant	10	76.9%	11	84.6%	11	84.6%
Convert Anonymous To Nested	4	17.4%	6	26.1%	8	34.8%
Inline Method	4	40.0%	4	40.0%	4	40.0%
Push Down	3	100.0%	3	100.0%	3	100.0%
Pull Up	2	14.3%	3	21.4%	3	21.4%
Move Resource	2	33.3%	3	50.0%	3	50.0%
Rename Compilation Unit	2	40.0%	2	40.0%	3	60.0%
Introduce Parameter	1	100.0%	1	100.0%	1	100.0%
Move Method	1	100.0%	1	100.0%	1	100.0%
Extract Superclass	0	0.0%	0	0.0%	1	6.3%
Use Supertype	0	0.0%	0	0.0%	0	0.0%

表 11: Users の一度に連続するリファクタリングの数と実施された回数

	時間間隔（秒）		
連続するリファクタリングの数	60	90	120
2	388	380	378
3	144	153	149
4	74	77	73
5	16	30	43
6	16	20	26
7	13	17	20
8	0	0	2
9	4	4	4
10	1	1	1
11	0	2	1
12	0	0	1
13	1	1	1
14	0	0	0
15	0	0	0
16	0	0	0
17	1	1	1
18	0	0	0
19	0	0	0
20	0	0	0
21	0	0	0
22	0	0	0

表 12: Mylyn の一度に連続するリファクタリングの数と実施された回数

	時間間隔（秒）		
連続するリファクタリングの数	60	90	120
2	475	479	482
3	133	137	132
4	58	64	72
5	35	37	40
6	18	22	19
7	12	13	11
8	6	7	7
9	5	12	11
10	3	6	8
11	3	3	4
12	2	3	4
13	2	3	2
14	0	0	2
15	1	1	3
16	1	1	1
17	1	1	2
18	0	0	2
19	0	0	0
20	0	1	1
21	1	1	1
22	0	0	0
23	0	0	0
24	0	0	0
25	0	0	0
26	0	0	0

4.2 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

連続して実施された, 互いに異なる種類のリファクタリングの組み合わせのうち, 実施された頻度の高いものを表 13, 14, 15 に示す. ただし, 連続して実施されたかを判断する時間間隔は 90 秒とした.

表 13 が Users のデータで, 表 14 が Mylyn のデータである. 表のリファクタリングの組み合わせは, 10 回以上実施されたものを回数の多い順に示している. 最初に実施されたものがリファクタリング 1 で, その次に実施されるのがリファクタリング 2 である. リファクタリング 3 は更に連続して実施された場合で, ”-”はリファクタリングが連続して実施されなかったことを示す. 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少なかった.

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 15 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. 表のリファクタリングの組み合わせは, 10 回以上実施されたものを回数の多い順に載せている. 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 15 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.

表 13: 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 3 の”-”はリファクタリングが連続して実施されなかったことを示す

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	リファクタリング 3	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	Rename	Rename	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
Rename	Extract Local Variable	-	11
Extract Method	Extract Local Variable	-	10

表 14: 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 3 の”-”はリファクタリングが連続して実施されなかったことを示す

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	リファクタリング 3	実施回数
Move	Rename	-	88
Rename	Move	-	62
Move	Move	Rename	30
Move	Rename	Rename	25
Rename	Move	Move	21
Move	Rename	Move	18
Rename	Rename	Move	17
Rename	Move	Rename	14
Extract Interface	Move	-	12
Extract Constant	Rename	-	11
Extract Constant	Move	-	10

表 15: Mylyn の Rename と Move, Inline を対称の種類で分類した場合の, 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 3 の”-”はリファクタリングが連続して実施されなかったことを示す

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	リファクタリング 3	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
Move Static Member	Rename Field	-	11
Rename Type	Move	Move	11
Extract Constant	Move Static Member	-	10
Rename Package	Move	-	10
Move	Move	Rename Type	10
Move	Rename Type	Rename Type	10

4.3 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 のデータセットで, 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結果を示す. ただし, Rename と Move, Inline を対称の種類で分類して, 組み合わせの順序を無視した場合とする.

表 16 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. 最初に実施されるのがリファクタリング 1 で, その次に実施されるのがリファクタリング 2 である.

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数である. 関係ありは, 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. 関係不明は, リファクタリング対象の関係が良く分からないものの数である. 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.

表 16: 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	関係不明
Rename Type	Move	58	12	4	3	5
Rename Field	Rename Method	48	10	0	5	5
Rename Type	Rename Method	32	10	0	4	6
Move	Rename Package	29	10	1	7	2
Rename Field	Rename Type	26	10	0	3	7
Move Static Member	Rename Field	15	15	9	0	6
Extract Interface	Move	14	14	10	2	2
Rename Local Variable	Rename Field	12	12	0	7	5
Move	Move Static Member	12	12	0	0	12
Rename Field	Move	12	12	1	0	11

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.

(Rename Type, Move) 同じ対象へリファクタリングが実施された例としては, クラスの名前を変更して他のパッケージへ移動していた. 関係のある作業内容としては, クラスの名前を変更してそのクラスにメソッドやフィールドを移動していた. 他に関係のある作業内容としては, あるパッケージの中のクラスを別のパッケージに移してから, 空のパッケージを削除する例もあった.

(Rename Field, Rename Method) 関係のある作業内容としては, フィールドの名前を変更し, 連続して元の名前と類似する名前のメソッド, 例えばそのフィールドの setter や getter などの名前を変更していた. 名前が類似するとは, 名前の一部あるいは全部が他の名前と共通することである. あるフィールドの名前を変更した場合に, 一貫性を保つため, メソッドの名前の共通する部分を変更していたことが多かった. また, これらのリファクタリングの前後に他のものを対象とした Rename が実施されていたことが多かった.

(Rename Type, Rename Method) 関連のある作業内容としては, クラスの名前を変更し, 名前が類似するメソッドを一貫して名前を変更していた. 名前が類似するメソッドはそのクラス内にあるものだけでなく, 名前を変更したクラスを対象とした他のクラスにある getter の場合もあった.

(Move, Rename Package) 同じ対象へリファクタリングが実施された例としては, パッケージを移動してから名前を変更していた. 関連のある作業内容としては, クラスを移動した先のパッケージまたはその親のパッケージの名前を変更をしていた. 他に関連のある作業内容としては, パッケージの統合があった. 例えば, パッケージ A からパッケージ B にいくつかクラスを移動し, A を削除して, B を A の名前に変更していた. また, これらのの前後に Rename や Move が連続して実施されていたことが多く, クラスとパッケージの整理をまとめて行っていると推測できる.

(Rename Field, Rename Type) クラスの名前を変更してからそのクラスのフィールドの名前を変更する, またはそれを逆順に実施する, クラスの名前を変更してから他のクラスにあるフィールドの名前を変更するという作業内容があった. 関連のある作業内容では, 名前が類似するものを同じ名前の変え方で名前を変更していた. しかし, リファクタリングの対称の名前に類似性が無かったものも多く, それらは関係不明とした.

(Move Static Member, Rename Field)　同じ対象へリファクタリングが実施された例としては, フィールドを他のクラスへ移動して名前を変更していた. また, それを逆順に実施している場合もあった.

(Extract Interface, Move)　同じ対象へリファクタリングが実施された例としては, 抽象したインタフェースを他のパッケージへ移動していた. また, これらのリファクタリングの前後でまた別のクラスを対象にして, インタフェースの抽象と移動が連続して実施されていることが多かった.

(Rename Local Variable, Rename Field)　関連のある作業内容としては, コンストラクタなどのローカル変数の名前を変更してから, そのクラスの名前が類似するフィールドの名前を変更していた. 同じ名前の別々のものを一貫して変更する場合と, 共通の語を含むものを一貫して変更する場合の両方があった. また, これらのリファクタリングの前後ではいくつかの Rename が, 対称の種類や属するクラスなどに関係なく, 実施されていることが多かった.

(Move, Move Static Member)　クラスを移動してからそのクラスへ他からフィールドを移動する, またはクラスを移動してからそのクラスのフィールドを他のクラスへ移動するなどの作業内容がされていたが, 名前に類似性は無く関係不明とした. また, これらのリファクタリングの前後には Move と Move Static Member が多く, コードを整える作業をまとめて行っていると推測できる.

(Rename Field, Move)　同じ対象へリファクタリングが実施された例としては, フィールドの名前を変更してから移動していた. 関係不明とした作業内容では, クラスの移動とそのクラスのフィールドの名前を変更が連続して実施されることが多かったが, 移動したクラスや移動先のパッケージの名前と, 名前変更の対象のフィールドの名前に関係がなかった.

5　考察

5.1　頻度の高いリファクタリングの組み合わせの支援方法

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説明する.

(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. そのため支援方法としては, Move でダイアログが表示されるので, そこでまとめて Rename のリファクタリングを行えるように, 名前入力欄を設けることが考えられる. あるいは, 連続して Rename を実施するというチェックボックスを設け, 続けて Rename のリファクタリングのダイアログを表示することも考えられる. ただし, 複数のクラスを Move するときには, 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.

(Rename, Rename) の支援　クラスの Rename から類似の名前のフィールドとメソッドの Rename ができるのと同じように, フィールドとメソッドの Rename から類似の名前のクラスやフィールド, メソッドの Rename はできるようにすることが考えられる.

(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, 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.

パッケージの Move の支援　パッケージの Move を容易にすることで, リファクタリングを連続して実施することも容易になる. そのために, パッケージエクスプローラでのドラッグ＆ドロップによるパッケージの Move を, クラスの Move と同様の仕様にし, パッケージをドラッグ＆ドロップで同じフォルダ内の別のパッケージの中へと Move することができるようにすることが考えれる.

また, あるパッケージに含まれる全部のクラスを, 他のパッケージに移動してから削除するという, パッケージを統合するような特殊な操作が見られた. これについては, あるパッ

ケージを中身のクラスは別のパッケージに移してから削除するというリファクタリングを, リファクタリングのメニューから実施できるようにすることが考えられる.

5.2 データセットによる結果の違い

Users の結果と Mylyn の結果では, 特にリファクタリングの種類毎の実施された割合とリファクタリングの実施された割合が大きく異なった. この原因としては, データセットの対象である開発者層の違いが最も大きいと考えられる.

Users のデータセットは幅広い層の開発者を対象としており, 彼らはリファクタリングのツールを作成した経験を持たない. Mylyn のデータセットは Eclipse の Mylyn プラグインの開発者を対象としていて, 彼らは Eclipse のリファクタリング機能についてはある程度詳しいはずである. そのため, Users の結果の方がより一般的な開発者のリファクタリングの傾向を表していると推測できる.

しかし, Users のデータセット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詳細な情報が含まれていなかったため, 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作業内容がされているのかを調査することができなかった. Users のような一般的な開発者を対象としたデータセットから, 詳細なリファクタリング実施履歴を調査することは今後の課題である.

また, 開発者の違いによるリファクタリングの傾向についてより詳しく調査し, 明確な特徴を見付けるためには, リファクタリングを使い慣れていない開発者を対象にしたデータセットや, リファクタリングの専門家を対象としたデータセットなど, より多くのデータセットを調査する必要がある. 特に専門家のリファクタリングの傾向がわかれば, 未熟な開発者がリファクタリングを効果的に習得することが出来るようになると期待できる. また, あるプロジェクトについて適度にリファクタリングが実施されているかを判定するなどの利用方法が考えられる.

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の組み合わせを, 一度にまとめて実施することを支援するために, 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, その結果から連続して実施するための支援方法を考案した.

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 [4] であり, Bad-smell の組み合わせ [15], あるいはコードクローンの量 [8] 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 [9] 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, 調査の結果を利用してリファクタリングの連続した実施を支援するためのツールを開発する予定である.

謝辞

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 井上 克郎 教授には研究について大変貴重なご意見を頂きました．また, 研究室の環境を良いものにしてくださっているおかげで, 研究に専念することができました．心より深く感謝いたします．

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 松下 誠 准教授には研究について様々なご意見を頂きました．また, 研究だけでなく学生生活などにも多くの助言を頂きました．心より深く感謝いたします．

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 石尾 隆 助教には研究について貴重なご意見を頂きました．また, 将来的な研究に役立つであろう助言を頂きました．心より深く感謝いたします．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学研究室 吉田 則裕 助教には研究のご指導をして頂き, 研究について様々な面でお世話になりました．終始適切な指示を頂けたおかげで本研究を形にすることが出来たと思います．心より深く感謝いたします．

井上研究室の先輩方には常日頃から様々なご指導, 助言を頂き, 心より感謝いたします．特に博士後期課程 2 年 崔 恩瀞さんと博士前期課程 2 年 後藤 祥さんには, 研究を始めから終わりまで見て頂き, 研究の進め方や論文の書き方など様々な面で助けてくださいました．本当にありがとうございます．

最後に, 井上研究室の皆様には学生生活全般でお世話になりました．皆様のおかげで大きな事故も無く, 研究を進めることが出来たと思います．心より深く感謝いたします．

参考文献

[1] 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. In *Proc. of SCAM*, pp. 104–113, 2012.

[2] E. Choi, N. Yoshida, and K. Inoue. What kind of and how clones are refactored?: a case study of three OSS projects. In *Proc. of WRT*, pp. 1–7, 2012.

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

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

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

[6] 肥後芳樹, 楠本真二, 井上克郎. コードクローンを対象としたリファクタリングの有効性に関する調査. 電子情報通信学会技術研究報告, Vol. 106, No. 201, pp. 37–42, 2006.

[7] 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.

[8] 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.

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

[10] G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the Eclipse IDE? *IEEE SOFTWARE*, Vol. 23, No. 4, pp. 76–83, 2014.

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

[12] 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.

[13] 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.

[14] L. Tokuda and D. Batory. Evolving object-oriented designs with refactorings. *Proc. of ASE*, Vol. 8, No. 1, pp. 89–120, 2001.

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