解　説

変更履歴解析に基づくリファクタリング検出技術の調査

崔 恩瀞　藤原 賢二　吉田 則裕　林 晋平

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変化させることなく，内部の構造を改善するプロセスを指す．研究者・実務者ともに，開発プロジェクトにおいて過去に実施されたリファクタリングを知りたいという要求がある．そこで，リファクタリングの実施を自動的に検出する手法 (リファクタリング検出手法) が数多く提案されている．これらの手法は，多様な国際会議や論文誌において発表されており，研究者や実務者にとって研究成果を概観することは容易ではない．本稿では，リファクタリング検出手法の中でも，盛んに研究が行われている成果物の変更履歴解析に基づく手法を中心に紹介を行う．まず，本稿におけるリファクタリング検出の定義および分類について述べる．その後，成果物の変更履歴解析に基づく手法を紹介し，今後行われる研究の方向性について考察を行う．

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. Not only researchers but also practitioners need to know past instances of refactoring performed in a software development project. So far, a number of techniques have been proposed on the automatic detection of refactoring instances. Those techniques have been presented in various international conferences and journals, and it is difficult for researchers and practitioners to grasp the current status of studies on refactoring detection techniques. In this survey paper, we introduce refactoring detection techniques, especially in techniques based on change history analysis. At first, we give the definition and the categorization of refactoring detection in this paper, and then introduce refactoring detection techniques based on change history analysis. Finally, we discuss possible future research directions on refactoring detection.

1　はじめに

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変化させることなく，内部の構造を改善するプロセスを指す [16] [45]．ソースコードに対してリファクタリングを実施する目的には様々なものがある [16]．例えば，複雑な部分や可読性の低い部分を改善することで，保守性を向上させ，欠陥の混入を予防することがある．実務者・研究者ともにリファクタリングへの関心は高く，数多くの書籍や論文が出版されている [16] [29] [37]．

実務者・研究者ともに，開発プロジェクトにおいて過去に実施されたリファクタリングを知りたいという要求がある．その要求は，大きく 2 つに分類することができる．

- 実務者が，使用中のライブラリやフレームワーク，API に対して実施されたリファクタリングを理解することで，保守対象のソフトウェアにおいて追随するか否かや追随方法の判断に役立てたい [11] [59].
- 研究者が，開発プロジェクトにおけるリファク

A Survey of Refactoring Detection Techniques Based on Change History Analysis.

Eunjong Choi, 大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University.

Kenji Fujiwara, 奈良先端科学技術大学院大学情報科学研究科, Graduate School of Information Science, Nara Institute of Science and Technology.

Norihiro Yoshida, 名古屋大学大学院情報科学研究科, Graduate School of Information Science, Nagoya University.

Shinpei Hayashi, 東京工業大学大学院情報理工学研究科, Graduate School of Information Science and Engineering, Tokyo Institute of Technology.

コンピュータソフトウェア, Vol.32, No.1 (2015), pp.47–59.
[解説論文] 2014 年 4 月 14 日受付.

タリングの実施事例を収集することで，リファクタリングおよびその効果や支援手法に関する実証的研究を行いたい [9] [30].

しかし，リファクタリングによる変更とそれ以外の変更が同時に行われた場合は，リファクタリングによる変更が含まれているか否かの判断に時間を要する [18] [23] [42]．大規模開発プロジェクトの中には，数千回の変更履歴を持つものもあり，それらの変更全てに対して目視でリファクタリング実施の有無を検討することは現実的ではない．

そこで，リファクタリングの実施を自動的に検出する手法 (以降，リファクタリング検出手法) が数多く提案されている．これらの手法は，多様な論文誌や国際会議において発表されている．そのため，これらの手法を概観することは容易ではない．リファクタリング全般に関する研究や技術を概観した論文が 2004 年までにいくつか発表されている [13] [37] [38] が，ほとんどのリファクタリング検出手法は 2004 年以降に発表されている．そのため，これらの文献からリファクタリング検出手法に関する研究の傾向を十分に把握することは難しい．

本稿では，リファクタリング検出手法の中でも，盛んに研究が行われている成果物の変更履歴解析に基づく手法を中心に紹介を行う．まず，2 章において，本稿におけるリファクタリング検出の定義を行う．3 章では，リファクタリング検出手法の大まかな分類を行い，本稿において主に扱う成果物の変更履歴解析に基づく手法について述べる．4 章では，成果物の変更履歴解析に基づく手法の紹介を行い，5 章では今後行われる研究の方向性について議論を行う．最後に，6 章において本稿のまとめを述べる．

2 リファクタリング検出の定義

リファクタリングの詳細は，一般にはリファクタリングカタログにまとめられている．カタログには，特定の種類のリファクタリング操作に対して，その適用のための事前条件や事後条件，適用の具体的な手続きなどが，リファクタリング操作名 (リファクタリング名) を添えた形でまとめられている．ソフトウェアパターンの形で記述されているものもあり，これらはリファクタリングパターンとも呼ばれる．例えば，あるクラスに所属しているメソッドを他のクラスに移動するリファクタリング操作は「メソッドの移動」と呼ばれ，メソッドの移動が適用できるための事前条件などを含めてパターンとしてカタログにまとめられている．リファクタリングカタログは，様々な書籍 [15] [16] [29] やウェブサイトにまとめられている．

本稿におけるリファクタリング検出を次のとおりに定義する．あるソフトウェア成果物の版系列 $(v_0, \ldots, v_{n-1}, v_n)$ から抽出した版の対 $(v_a, v_b)(0 \leq a < b \leq n)$ が与えられたとき，その間に行われた変更の集合を $C = \{c_0, \ldots, c_{m-1}, c_m\}$ とする．このとき，リファクタリングカタログに記載されたリファクタリング操作が，変更の集合 C に含まれる空でない部分集合に存在するかどうか推定することをリファクタリング検出と呼ぶ．一般には，リファクタリング検出を行うツールは，例えば版の対 (v_1, v_2) が入力されたとすると「版の対 (v_1, v_2) 間において，メソッドの引き上げおよびフィールドの移動が行われた」といった情報を出力する．ただし，具体的なリファクタリング名を出力せずにリファクタリングの存在のみを示唆するものもある [53].

リファクタリングは，版の対の中で必ずしも単独で存在せず，他の変更と混在することがある [25] [42] [46]. Görg らは，他のリファクタリングやリファクタリング以外の変更と混在した変更を impure リファクタリングと呼んでいる [20]．pure リファクタリングと異なり，impure リファクタリングに関わる版の対では，必ずしも外部的振る舞いが保存されない．また，Murphy-Hill らは，欠陥修正や機能追加の最中にリファクタリングが実施されていることを指摘しており，こういったリファクタリングを floss リファクタリングと呼んでいる [42] [43]．リファクタリングとそれ以外の変更のタイミングを明確に分ける root-canal リファクタリングと比べ，floss リファクタリングではリファクタリングと他の変更が混在した版の対が生じやすい．Murphy-Hill らは，こういった floss リファクタリングが多数行われていることを報告している [42]．また Herzig らは，多様な種類の変更がこのように「もつれて」存在していることを報告している

[26].

このように，変更の混在は現実にはよく起こるため，リファクタリング検出手法は，版の対 (v_a, v_b) 間に行われた変更の集合に，リファクタリングのための変更だけではなく，バグ修正や機能追加のための変更が含まれていた場合であっても，リファクタリングが実施されたことを判定することが望まれる [25] [42]. 本稿では，こういった混じりのあるリファクタリングの検出についても調査対象に含める．

リファクタリング検出手法は，多くの差分の解析手法と技術的な背景を共有している．例えば，Systematic change [32] の検出など，変更の集合に対して一般的に認知された名前を付加する研究 [32] [33] が行われている．また，ある版におけるコード片と過去におけるコード片との対応をとる起源解析の多くは，プログラム要素の名前がどのように変更されたかを特定する技術を含んでいる [19] [34]．同様に，ソースコードやソフトウェアモデルの差分をわかりやすく取得する手法もあり，これらも一部のリファクタリング検出と同様の解析を行う [14]．本稿では，リファクタリングカタログに示されたリファクタリング操作を検出する手法の調査を主目的とし，こういった差分解析手法の網羅は目指さない．ただし，具体的なリファクタリングパターンを特定せずとも，リファクタリング検出としてプログラムの振る舞い保存を検査している手法は例外的に含むこととする．

3 リファクタリング検出の分類

本章では前章で述べたリファクタリング検出を目的とした手法を大きく 4 つに分類し，本稿で調査対象とする変更履歴の解析に基づく手法について述べる．

Murphy-Hill らは，リファクタリング検出手法を，2 つの観点を用いて 4 つに分類している [41]．彼らによる分類を表 1 に示す．1 つの観点として，それぞれの手法がリファクタリングの実施を発見する際に用いる手がかりについて，明示的にリファクタリングの実施が記録されたものを利用するかどうかで分類している (explict または implicit)．別の観点として，得られるリファクタリングの実施が，主観的な判断によるか客観的な事実に基づくかどうかで分類している (context または fidelity)．以降，本章では表中 A1 から A4 の手法についてそれぞれ紹介する．

表 1 リファクタリング検出方法の分類

	context	fidelity
explicit	A1：コミットログ	A3：ツールログ
implicit	A2：開発者の観察	A4：変更履歴の解析

A1 に分類される手法では，版管理システムに記録されたコミットログを解析することでリファクタリングの実施を発見する [44] [51] [58]．開発者がコミットログにリファクタリングの実施を記録していた場合，そのログを正確に抽出することでリファクタリングの実施を発見することができる．そのため，この手法は「refactor」や「extract」などのリファクタリングに関する単語をコミットログから抽出する．つまり，開発者によるリファクタリング実施の明示的な記録を利用するが，リファクタリングを記録するかどうかの判断や記録される情報は開発者の主観的な判断に基づくという特徴がある．これらの手法は，版管理システムを用いて開発されたソフトウェアの履歴であれば適用可能である．一方で，記録の精度が開発者の性質に依存することと，リファクタリングの具体的な適用箇所について情報を得られないという欠点がある．Murphy-Hill らは，開発者が実際にリファクタリングを実施する頻度に比べて，コミットログにそのことを記録する頻度は少ないと報告している [42]．そのため，開発者のリファクタリング傾向を調査することを目的とした研究にこの手法を利用する場合は，得られるリファクタリングの履歴に偏りがあることに留意する必要がある．

A2 に分類される手法では，研究者が開発者の活動を直接またはツールを使って間接的に観察することでリファクタリングの実施を発見する [6] [40] [48]．具体例として，ツールを用いて開発者が利用している開発環境の画面を定期的に記録し，その記録からリファクタリングを検出することが挙げられる．このような手法で利用する記録にはリファクタリングの実施が明示されていない．また，開発者がリファクタリングを実施したことを研究者が判断するため，得られる結果は個人の主観によるものである．これらの手法は適用範囲が限られるが，必要な情報を詳細に収集することが

可能である．

A3 に分類される手法では，リファクタリング支援ツールのログを収集することでリファクタリングの実施を発見する [12] [39] [52]．ここでのリファクタリング支援ツールとは，統合開発環境が提供するリファクタリング機能に代表されるリファクタリングを自動的に適用するためのものである．開発者がそれらのツールを使用する目的がリファクタリングの実施であることは自明である．これらの手法では，リファクタリングの実施時期と適用したリファクタリングの種類，適用箇所などを詳細に知ることができる．また，リファクタリング支援ツールは外部的な振る舞いを変更しないことを保証しているため，pure リファクタリングに関する情報を収集できることが特徴である．一方で，対象とするリファクタリング支援ツールを利用して適用したリファクタリングの実施しか得られないという制限がある．

A4 に分類される手法では，ソフトウェア開発における成果物の変更履歴を解析することで，リファクタリングの実施を発見する．これらの手法ではリファクタリングの実施を開発者や分析者の主観によって判断せず，ソースコードを代表とする成果物の変更を基に客観的な判断を行う．一方で，開発者がリファクタリングとして意図した変更を必ず抽出できるという保証はない．

本稿では，A4 に属する手法を主な調査対象とする．近年では，企業およびオープンソースソフトウェアのソフトウェア開発において，成果物の変更履歴を記録することは一般的となっている．そのため，これらの手法を適用できる範囲は広く，A1 から A3 に分類される手法よりも多くのリファクタリング実施事例を収集できることが期待される．ライブラリやフレームワークに適用されたリファクタリングに関する情報の収集，リファクタリングの適用事例を用いたリファクタリングおよびその効果の実証的研究を行う上では，より多くの事例を収集できることは有益である．

4 変更履歴に基づく手法の紹介

4.1 分類方法

本稿では 3 章で述べたアプローチの中から，成果物の変更履歴の解析によってリファクタリングを検出するアプローチを分類対象として選択した．その理由は，開発者がリファクタリングを実施すると必ず成果物に反映されることから，成果物の変更履歴解析が他の分類と比較して多くのリファクタリング事例を対象にできると考えたからである．

さらに，これまでに提案されている成果物の変更履歴の解析方法によってリファクタリング検出する技術に関する論文をソフトウェア工学分野の主要な国際会議 (APSEC，ASE，CSMR，FSE，ICSE，ICSM，MSR，OOPSLA，SCAM，WCRE) や論文誌 (IEEE Transactions on Software Engineering，Information and Software Technology，Journal of Systems and Software，Journal of Software: Evolution and Process) から調べ，中身を分析した．その結果に基づき，主として使われている技術を以下の 6 種類に分類した．

- 検出規則に基づく手法
- コードクローン解析に基づく手法
- メトリクスに基づく手法
- 動的解析に基づく手法
- グラフマッチングに基づく手法
- 探索に基づく手法

表 2 は各検出技術がどの手法によってリファクタリングを検出するかを表している．本稿で紹介される技術は，対象国際会議や論文誌の論文の中から上記の 6 つの分類に属する検出手法を重要度に基づいて選別したものである．また，上記の範囲外で発表されている重要な手法についても，著者らが知る限り含めた．以降，4.2 節から 4.7 節まで，各手法に属する検出技術を紹介する．なお，複数の手法を用いてリファクタリングを検出する技術は重複して紹介されることがある．

4.2 検出規則に基づく手法

検出規則とは，各種類のリファクタリングが実施されたかを判断するための基準であり，各種類のリファクタリングの実施前後のソースコードに関する規則をクラスやメソッドの名前やパラメータの追加，削除，移動などの情報を用いて定義したものである．また，ソースコードの変更を正確に検出するために，リ

表 2 リファクタリング検出技術と手法

掲載年	文献	検出規則	コードクローン	メトリクス	動的解析	グラフマッチング	探索
2000	Demeyer [10]			✓			
2004	Antoniol [2]	✓					
2005	Görg [20]	✓					
	Xing [62] [63] [64]	✓					
2006	Advani [1]	✓					
	Weißgerber [61]	✓	✓				
	Dig [11]	✓					
2007	Pérez [47]					✓	✓
	Taneja [59]	✓					
2008	Hayashi [24] [25]	✓					✓
2010	Kim [31], Prete [50]	✓					
2011	Biegel [5]	✓	✓				
	Soares [53]				✓		
	Kehrer [28]					✓	✓
	Thangthumachit [60]						✓
2012	Fadhel [3]						✓
2013	Mahouachi [36]			✓			✓
	Soetens [57]					✓	
	藤原 [17]	✓					

ファクタリングの実施前後のソースコードの類似度が検出規則に含まれる場合もある．例えば Prete らの手法では，メソッドの抽出の検出規則は「リファクタリング前からあるメソッドと追加されたメソッドが類似し，かつリファクタリング前のメソッドから追加されたメソッドへの呼び出し文が追加されている」と定義されている [31] [50]．Prete らの手法では，2 つの版のソースコードからパッケージやクラスなどの構文要素を fact として抽出し，それら fact に基づいて版間の差分を計算する．その差分が事前に定義された検出規則のいずれかに一致する場合，その版間においてリファクタリングが実施されたと判定する (図 1)．検出規則に基づく手法は，版間の変更を直接的かつ宣言的に表現できるため，リファクタリングパターンを基に検出規則を記述しやすいという利点がある．一方，検出規則の検討が不十分であると検出精度が低下するという欠点や，リファクタリング以外を目的とした変更が混在する impure リファクタリングの検出に向かないという欠点がある．

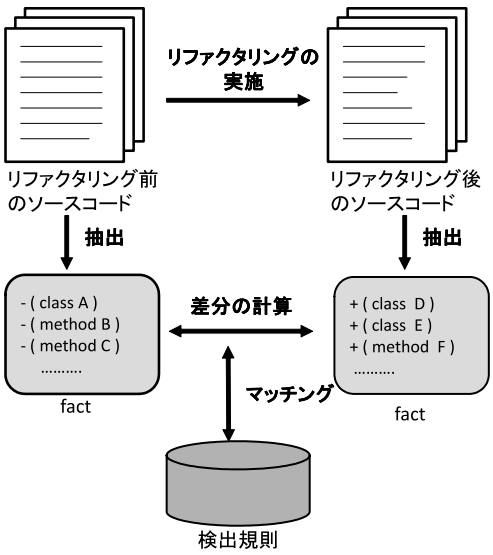

図 1 検出規則に基づく手法の概要

Antoniol らや Advani らの手法では Fowler の各種類のリファクタリングの定義に基づいた簡単な検出規則が用いられている [1] [2]．Antoniol らは各クラスの識別子名の類似度と各クラス単位のリファクタリングが成り立つための条件に基づき，クラスの置き換えやクラスの抽出などのクラス単位のリファクタリングを検出する手法を提案し，Java プログラムのクラスの進化を調査している [2]．この手法では，各クラスから抽出された識別子が TF (Term Frequency)-IDF (Inverse Document Frequency) によって重み付けら

れ，特徴ベクトルに変換される．さらに，クラスの新しい出現などの版間でのクラスの状況による条件とベクトル間のコサイン類似度によりリファクタリングが検出される．この手法を dnsjava の 40 リリースに対して適用したところ，クラスの置き換え，クラスのマージ，クラスの分割，クラスの抽出の検出が確認された．

Advani らは各種類のリファクタリングに関する Fowler の定義に基づいた検出規則を用いて，連続したリリース間でリファクタリングを検出するツールを開発し，各種類のリファクタリング間の関係を調査した [1]．このツールでは，2 つの版のソースコードから抽出されたクラスのエンティティ (例：メソッドやフィールド) が各種類のリファクタリング検出規則にマッチするとリファクタリングとして検出される．このツールを 7 個の OSS に適用したところ，メソッドの移動，フィールドの移動，メソッド名の変更は，他のリファクタリングの一環として実施されることが多いことがわかった．

Görg らや Weißgerber らの手法では，2 つの版間のシグネチャの差分に基づいた検出規則が用いられている [20] [61]．Görg らは 2 つの版間のクラスとメソッドのシグネチャの差分に基づいた検出規則を用いてリファクタリングを検出し，その結果を可視化するツール REFVISE を開発している [20]．REFVISE は 2 つの版間でのクラス名とメソッド名の追加，削除などの情報に基づいた検出規則を用いてリファクタリングを検出し，検出結果をクラス単位で可視化している．Weißgerber らは Görg らの手法を拡張し，2 つの版間のメソッドなどのシグネチャの差分に基づいた検出規則を用いてリファクタリングを検出する手法を提案している [61]．この手法は 2 つの版間のクラス，インターフェース，メソッド，フィールドの差分に基づいた検出規則を用いてリファクタリングを検出し，検出したリファクタリングを，リファクタリングに関わったソースコードの類似度に基づいて分類した．REFVISE は 2 つの版でソースコードが完全に一致しているメソッド対をリファクタリングとして検出するが，Weißgerber らの手法は完全に一致していないメソッドもリファクタリングとして検出することができる．また，REFVISE はクラスとメソッドのシグネチャの差分のみを用いてリファクタリングを検出するが，Weißgerber らの手法はクラス，インターフェース，メソッド，フィールドなどより多くの情報の差分を用いている．

Xing らは各種類のリファクタリングを表すクエリに基づいてリファクタリングを検出するツール UMLDiff を開発している [62] [63] [64]．UMLDiff は 2 つの版からパッケージやクラスやそれらの型，名前，アクセス修飾子などの設計レベルの情報を抽出し，それらの名前の変更や移動，削除による差分を計算する．さらに，計算した差分が各種類のリファクタリングを表すクエリとマッチするとその差分をリファクタリングとして検出する．

Prete らは各種類のリファクタリングの検出規則に基づいて 63 個のリファクタリングを検出するツール Ref-Finder を開発している [31] [50]．Ref-Finder は 2 つの版からパッケージやクラスなどのコード要素やそれらの間の関係，構造に関する依存関係などを抽出し，それらの差分を計算する．さらに，差分が各種類のリファクタリングに対する検出規則 [49] と一致したとき，リファクタリングとして検出する．検出規則は論理式で表現されており，複雑なリファクタリングも基本的なリファクタリングの組み合わせとして検出している．Ref-Finder は条件分岐や例外処理など，詳しくソースコードを解析することにより，UMLDiff より多い種類のリファクタリングを検出している．

また，藤原らは開発履歴に記録されている全ての隣接するリビジョン間からリファクタリング検出を実施するための手法を提案している [17]．この手法は，Hata らが提案している Historage [22] を用いることで，ソースコードからの fact 抽出，リビジョン間における fact のマッチングを高速化している．

検出規則は，コンポーネントの変更履歴からリファクタリングを検出する際にも用いられている．一般的に，コンポーネントに対してリファクタリングが実施される場合，後方互換性を維持するためにリファクタリング前のソースコードが保持される．そのため，リファクタリング前後のソースコードがコンポーネントに混在し，リファクタリングの検出が困難となる．こ

の問題を解決するために，Dig らと Taneja らは，それぞれ検出規則に基づくツールを開発した [11] [59]．Dig らが開発した Eclipse プラグイン RefactoringCrawler [11] は，まず shingles [7] を用いて類似したエンティティを見つける．さらに，エンティティ名の変更に関する構文的規則，およびメソッドの呼び出しなどエンティティ間参照に関する意味的規則を用いてリファクタリングを検出する．また，各種類のリファクタリングの検出順序を定義し，1 つのエンティティからリファクタリングが検出された後も検出処理を継続することで，floss リファクタリングの検出も可能となっている．Taneja らは RefactoringCrawler を拡張し，API の変更履歴からリファクタリングを検出するツール RefacLib を開発した [59]．RefacLib は RefactoringCrawler と同様に，まず 2 つの版の API から類似エンティティを抽出する．そして，構文解析の結果や，エンティティ名の類似度やサイズ，廃止予定のエンティティなどの情報を用いて，リファクタリングを検出する．

4.3 コードクローン解析に基づく手法

ソースコード中から類似したコード片の対，もしくは集合を検出するコードクローン検出ツール [4] [27] を用いて，ある版のコード片がその後の版においてどこに移動・抽出されたかを追跡することで，リファクタリングを検出する研究が行われている．コードクローン検出ツールを用いているため，完全に一致したコード片だけでなく，類似したコード片の移動・抽出を追跡することができる．impure リファクタリングのように多くの種類の変更が行われている場合は，検出精度が低下すると考えられる．

Weißgerber らは，コードクローン検出ツール CCFinder [27] を利用してリファクタリング検出を行う手法を提案している [61]．彼らは，版間におけるコード片の変化を，コード片対の類似性に基づいて，EQUAL(文字列が変化していない)，CLONE(EQUAL ではないが CCFinder が検出する)，NOCLONE(EQUAL, CLONE のいずれでもない) の 3 種に排他的に分類し，それら分類とリファクタリングの有無の関係を調査した．その結果，EQUAL および CLONE に分類された場合のみを検出することで，再現率を大きく低下させることなく，適合率を向上させられることがわかった．また，Biegel らはコード片対の類似度を 3 種類定義し，リファクタリング検出性能の差異を調査している [5]．3 種類の類似度のうち 2 つは，トークン列および抽象構文木の類似性に基づいて定義されており，それぞれコードクローン検出ツール CCFinder および JCCD [4] を用いて計測されている．残りの 1 つは，文字列の類似性に基づいて定義されており，文字列間の距離を表す shingles [7] が定義中で用いられている．調査の結果，いずれのコード類似度を用いた場合も，適合率, 再現率ともに大きな差異はないという結論が得られている．また，特定のコード類似度でなければ検出できないリファクタリングの数は限定的であり，3 種類の類似度間で検出可能なリファクタリングの大半を共有していたことも報告されている．

4.4 メトリクスに基づく手法

メトリクスに基づいてリファクタリングを検出する手法では，2 つの版間で実施されたリファクタリングが版間のメトリクス値の差分により検出される．メトリクスに基づく手法は 4.2 節の検出規則に基づく手法と比較して，軽量な解析しか行わないため高速な検出を実現できるが，その一方で詳細な分析を行わないため検出精度が低いという欠点がある．Demeyer らは Chidamber & Kemerer [8] と Lorenz & Kidd [35] メトリクスからメソッドとクラスのサイズ，クラスの継承に関するメトリクスを選び，それらの組み合わせをヒューリスティックとして用いて子クラスと親クラスの分割とマージやメソッドの分割などに関するリファクタリングを検出する手法を提案している [10]．

また，Mahouachi らは版間の構造に関するメトリクスの差分に基づいてリファクタリングを検出する手法を提案している [36]．彼らの手法は，探索に基づいて，遺伝的アルゴリズムを用いてメトリクスの差分に最も近いリファクタリングの組み合わせを検出する．

4.5 動的解析に基づく手法

リファクタリングの適用前後でプログラムの振る舞

いが保存されることを前提としてリファクタリング検出を行う手法も提案されている．振る舞いの保存を確認するためには適用箇所に関するテストケースを実行し，その出力がリファクタリングの適用前後で変わらないことを調べればよい．現在提案されている，動的解析に基づく手法では pure/impure リファクタリングを混同することなく，pure リファクタリングのみを検出できることが利点である．一方，動的解析に基づく情報だけでは実施されたリファクタリングの種類を判別することができないという欠点がある．

Soares らは，リファクタリングの適用前後においてソフトウェアの振る舞いの変更を検知するためのツール SafeRefactor [55] を用いたリファクタリング検出手法を提案している [53]．SafeRefactor は，版間で変更が加えられていないメソッドに対して単体テストを自動生成する．そして，自動生成したテストを実行し，テストが失敗すれば振る舞いが変更されていることをユーザへ通知する．Soares らの手法では，変更前後のソースコードを SafeRefactor へ入力として与え，振る舞いの保存が確認された場合に，リファクタリングが適用されたとみなす．

4.6 グラフマッチングに基づく手法

プログラムやプログラムの変更をグラフ構造とみなし，リファクタリング操作に相応するパターンが部分グラフとして含まれるかを調べることによりリファクタリングを検出する手法も提案されている．UML クラス図をはじめとして，ソフトウェアの設計はグラフとみなせるものが多いため，モデルリファクタリングを検出する際にグラフマッチングを利用するアプローチは正統的と考える．検出すべきリファクタリングをパターンとして記述できることは，記述の容易さに貢献する．また，ソースコード変更をグラフとして扱うことにより，リファクタリングが他の変更と混ざった場合でも部分グラフとして検出できるという利点もある．一方，モデルが複雑になるとパターンの記述が困難となる点，リファクタリングの種類によってはパターンとして十分に表現できない等の欠点もある．

Kehrer らは EMF モデル上の操作列からリファクタリング等の高レベル変更をモデル差分から抽出することにより，差分の抽象度を上げる手法を提案している [28]．モデルにおいては，版間の操作は，節や辺の追加や削除といったプリミティブな操作が加えられたグラフとして表すことができる．これらに含まれるリファクタリング等の高レベル変更を部分グラフとして検出しまとめることにより，より抽象度の高い変更の表現とすることができる．Kehrer らはこういった操作を Semantic lifting と呼び，自動化している．

また Soetens らは，履歴のマッチングに基づく floss リファクタリングの検出手法を提案している [57]．floss リファクタリングは他の変更の最中に行われるため，版間から得られる情報は複数の修正が混ざったものとなり，その検出が困難である．この手法では，ツール ChEOPSJ [56] が記録するソースコードの編集操作履歴に基づきコード変更を表すグラフを構築し，グラフ変換ツールを用いてこのグラフがリファクタリング操作を表すグラフのパターンを含んでいるかを特定する．パターンを含んでいた場合，該当のリファクタリングが行われたとする．操作履歴に基づき検出することにより，メソッドの名前変更や移動といったリファクタリングが，既存手法よりも正確に検出できると述べている．

4.7 探索に基づく手法

リファクタリング検出においては，複合的なリファクタリングや他の変更と混ざったリファクタリングを適切に検出することも重要となる．impure リファクタリングが行われた場合，リファクタリング前後の版の間に行われた変更は，単一のリファクタリングの影響のみならず，他のリファクタリングやリファクタリングではない変更の影響が混ざったものとなる．こういった場合，変更前後の版間の差分と (単一の) リファクタリングの事前・事後条件が対応しないため，条件の検査のみではリファクタリングを正しく検出することができない．

ソフトウェア工学の問題を一種の最適化問題ととらえ，探索的手法に基づき結果を得る手法は search-based software engineering(SBSE) [21] と呼ばれており，リファクタリングの検出においてもその応用例がある．こういった手法では，プログラムを状態，リ

ファクタリングの適用を状態変化のオペレータとみなし，版間のプログラムの変化をよく表すような最適なオペレータの列を探索によって発見する．実際にソースコードなどのソフトウェア成果物に対してリファクタリングを実行することでオペレータの適用とし，新しい成果物を得ることを繰り返すことにより探索が進む．探索的手法を用いることにより，混在した変更のうち一部のみが適用された中間的なプログラム状態を間接的に扱うため，impure リファクタリングに対する検出規則を直接記述することなく検出を行えるという利点がある．一方で，用いる探索手法によっては大きな計算時間を必要とするという欠点もある．

例えば，Pérez らは UML クラス図のようなプログラムの構造的情報の差分から探索的にリファクタリング操作を発見する手法を提案している [47]．この手法では，深さ優先探索を適用しており，検出したリファクタリングの候補をプログラムに適用していくことにより，リファクタリングの列を発見している．

Hayashi らは A 探索を用いてリファクタリングを検出する手法を提案している [24] [25]．この手法では，プログラムの構造的差分の大きさをヒューリスティックな距離，適用済みのリファクタリングの重さを経路距離とし，リファクタリングの検出をこれらの和を評価関数とした経路探索問題として定式化し，A 探索を適用することによりその解経路を探索する手法を提案している．

遺伝的アルゴリズムの適用例もある．この手法では，リファクタリングの列を遺伝子座として，選択や交叉，突然変異といった遺伝子座の更新オペレータの適用を繰り返すことにより，版間の操作をもっともよく表すリファクタリング列，すなわちあらかじめ定めた適合関数の値が最適となる遺伝子座を探索する．Fadhel らは遺伝的アルゴリズムを用いてモデルリファクタリングを検出する手法を提案している [3]．また，Mahouachi らは，ソースコードに対して同様のアプローチを適用し，コードリファクタリングの列を得る手法を提案している [36]．この手法では，版間のプロダクトメトリクスの差が小さいほど最適となるような適合関数を定めて用いている．

Thangthumachit らは，抽象構文木における節の子要素および被参照の類似性に基づきリファクタリングを検出する手法を提案している [60]．この手法では，リファクタリングの検出をパッケージ，ファイル，クラス，メソッドといったレベル毎に行っており，細粒度のレベルに進む前に粗粒度のレベルで検出したリファクタリングを一方の版のプログラムに適用する．この際，適用に失敗したリファクタリングは検出結果から除外する．このように検出・適用を繰り返すことにより，複数のリファクタリングが混在した差分に対しても精度良い検出を実現している．さらに，このようにして得られたリファクタリング列を可視化するツールも提案されている [23]．

5 今後の研究課題

5.1 複数のアプローチを組み合わせた手法

今後の研究課題として，複数のアプローチを組み合わせた手法の考案・評価が考えられる．複数のアプローチの定量的比較を行った Soares らの論文では，今後の課題の 1 つとして，複数のアプローチを組み合わせたときの比較評価を挙げている [54]．

現在のところ，複数のアプローチを組み合わせた手法はあまり研究されていないことが表 2 からわかる．例えば，検出規則を用いてプログラムの構造を考慮しながら探索に基づく検出を行う，もしくはコードクローン解析に基づいてコード片の移動を検出しながら探索に基づく検出を行う手法を考案し，他の手法との比較を行う研究課題が考えられる．また，動的解析に基づく手法はあまり研究されていない．静的解析のみでは外部的振る舞いの変化を確認できない場合があると考えられるため，動的解析とコードクローン解析などの静的解析を組み合わせた手法が今後の研究課題として挙げられる．

5.2 検出手法の定量的評価

検出手法の定量的な比較評価はあまり行われていない．その理由の 1 つとして，検出すべきリファクタリングを定義することが難しいことが挙げられる．今後，定量的な比較評価を行うためのデータセットを構築し，研究コミュニティ内で共有する仕組みを作る必要がある．Soares らは，Ref-Finder [31] [50], 動的解

析に基づく手法[53]，コミットログに基づく手法[51]の3種類のリファクタリング検出手法の定量的比較を行い，その結果をウェブサイトにおいて公開している[54]．リファクタリング検出手法の改善のためには，このような比較評価および評価結果の公開を他の手法についても行うべきであると考えられる．定量的な比較評価を行うためのデータセットの構築にあたっては，impure/floss リファクタリングを対象とするか否かなど，データセットが対象とするリファクタリングの定義を明確にすることが求められる．

検出精度の定量的評価に加えて，版の系列の規模に対するスケーラビリティの定量的評価も必要であると考えられる．リファクタリングに関する大規模な実証的研究を行うためには，長年にわたり蓄積された版の系列に対して，版間の差分に関連する部分のみをインクリメンタルに解析することで．現実的な時間で検出を完了可能なスケーラビリティの高い検出手法が必要である．現状，2つの版間で実施されたリファクタリングの検出を目的としている手法が多く，各版のソースコード全体を解析するため，長年にわたり蓄積された版の系列を分析するためには大きな計算コストが必要となる．今後，版の系列の規模に対するスケーラビリティの定量的評価を行うことで，スケーラビリティの高いツールを実現していく必要がある．

6 まとめ

本稿では，リファクタリング検出手法の中でも，盛んに研究が行われている成果物の変更履歴解析に基づく手法を中心に紹介を行った．まず，本稿におけるリファクタリング検出の定義を述べ，リファクタリング検出手法をコミットログの解析に基づく手法，開発者の観察に基づく手法，ツールログの解析に基づく手法，成果物に対する変更履歴の解析に基づく手法の4つに分類した．その後，成果物に対する変更履歴の解析に基づく手法を更に6つに分類し，それぞれの分類に属する手法について紹介した．最後に，今後行われる研究の方向性について考察を行った．考察では，複数のアプローチを組み合わせた手法や手法の比較評価に関する議論を行った．

本論文が，リファクタリング検出技術発展の一助となれば幸いである．

謝辞

本研究の一部は科学研究費補助金 (課題番号：23700030，26730036) の助成を受けた．

参考文献

[1] Advani, D., Hassoun, Y. and Counsell, S.: Extracting Refactoring Trends from Open-source Software and a Possible Solution to the 'Related Refactoring' Conundrum, in *Proc. of the 21st ACM Symposium on Applied Computing* (*SAC'06*), 2006, pp. 1713–1720.

[2] Antoniol, G., Di Penta, M. and Merlo, E.: An Automatic Approach to Identify Class Evolution Discontinuities, in *Proc. of the 7th International Workshop on Principles of Software Evolution* (*IWPSE'04*), 2004, pp. 31–40.

[3] ben Fadhel, A., Kessentini, M., Langer, P. and Wimmer, M.: Search-based Detection of High-level Model Changes, in *Proc. of the 28th IEEE International Conference on Software Maintenance* (*ICSM'12*), 2012, pp. 212–221.

[4] Biegel, B. and Diehl, S.: Highly Configurable and Extensible Code Clone Detection, in *Proc. of the 17th Working Conference on Reverse Engineering* (*WCRE'10*), 2010, pp. 237–241.

[5] Biegel, B., Soetens, Q. D., Hornig, W., Diehl, S. and Demeyer, S.: Comparison of Similarity Metrics for Refactoring Detection, in *Proc. of the 8th Working Conference on Mining Software Repositories* (*MSR'11*), 2011, pp. 53–62.

[6] Boshernitsan, M., Graham, S. L. and Hearst, M. A.: Aligning Development Tools with the Way Programmers Think About Code Changes, in *Proc. of the SIGCHI Conference on Human Factors in Computing Systems* (*CHI'07*), 2007, pp. 567–576.

[7] Broder, A. Z.: On the Resemblance and Containment of Documents, in *Proc. of the Compression and Complexity of Sequences* (*SEQUENCES'97*), 1997, pp. 21–29.

[8] Chidamber, S. R. and Kemerer, C. F.: A Metrics Suite for Object Oriented Design, *IEEE Transactions on Software Engineering*, Vol. 20, No. 6(1994), pp. 476–493.

[9] Choi, E., Yoshida, N. and Inoue, K.: An Investigation into the Characteristics of Merged Code Clones during Software Evolution, *IEICE Transactions on Information and Systems*, Vol. E97-D, No. 5(2014), pp. 1244–1253.

[10] Demeyer, S., Ducasse, S. and Nierstrasz, O.: Finding Refactorings via Change Metrics, in *Proc. of the 15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications* (*OOPSLA'00*), 2000, pp. 166–177.

[11] Dig, D., Comertoglu, C., Marinov, D. and Johnson, R.: Automated Detection of Refactorings in Evolving Components, in *Proc. of the 20th European Conference on Object-Oriented Programming* (*ECOOP'06*), 2006, pp. 404–428.

[12] Dig, D., Manzoor, K., Johnson, R. and Nguyen, T. N.: Refactoring-Aware Configuration Management for Object-Oriented Programs, in *Proc. of the 29th International Conference on Software Engineering* (*ICSE'07*), 2007, pp. 427–436.

[13] Du Bois, B., Van Gorp, P., Amsel, A., Van Eetvelde, N., Stenten, H., Demeyer, S. and Mens, T.: A Discussion of Refactoring in Research and Practice, Technical report, University of Antwerp, 2004.

[14] Fluri, B., Wursch, M., PInzger, M. and Gall, H. C.: Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction, *IEEE Transactions on Software Engineering*, Vol. 33, No. 11(2007), pp. 725–743.

[15] Fowler, M.: Refactoring, http://refactoring.com/.

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

[17] 藤原賢二, 吉田則裕, 飯田元: ソフトウェアリポジトリを対象とした細粒度リファクタリング検出, ソフトウェア工学の基礎 XX (FOSE'13), 2013, pp. 101–106.

[18] Ge, X., Sarkar, S. and Murphy-Hill, E.: Towards Refactoring-Aware Code Review, in *Proc. of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering* (*CHASE'14*), 2014, pp. 99–102.

[19] Godfrey, M. W. and Zou, L.: Using Origin Analysis to Detect Merging and Splitting of Source Code Entities, *IEEE Transactions on Software Engineering*, Vol. 31, No. 2(2005), pp. 166–181.

[20] Görg, C. and Weißgerber, P.: Detecting and Visualizing Refactorings from Software Archives, in *Proc. of the 13th International Workshop on Program Comprehension* (*IWPC'05*), 2005, pp. 205–214.

[21] Harman, M.: Software Engineering Meets Evolutionary Computation, *IEEE Computer*, Vol. 44, No. 10(2011), pp. 31–39.

[22] Hata, H., Mizuno, O. and Kikuno, T.: Historage: Fine-grained Version Control System for Java, in *Proc. of the 12th International Workshop on Principles on Software Evolution and 7th ERCIM Workshop on Software Evolution* (*IWPSE-EVOL'11*), 2011, pp. 96–100.

[23] Hayashi, S., Thangthumachit, S. and Saeki, M.: REdiffs: Refactoring-Aware Difference Viewer for Java, in *Proc. of the 20th Working Conference on Reverse Engineering* (*WCRE'13*), 2013, pp. 487–488.

[24] Hayashi, S., Tsuda, Y. and Saeki, M.: Detecting Occurrences of Refactoring with Heuristic Search, in *Proc. of the 15th Asia-Pacific Software Engineering Conference* (*APSEC'08*), 2008, pp. 453–460.

[25] Hayashi, S., Tsuda, Y. and Saeki, M.: Search-Based Refactoring Detection from Source Code Revisions, *IEICE Transactions on Information and Systems*, Vol. E93-D, No. 4(2010), pp. 754–762.

[26] Herzig, K. and Zeller, A.: The Impact of Tangled Code Changes, in *Proc. of the 10th Working Conference on Mining Software Repositories* (*MSR'13*), 2013, pp. 121–130.

[27] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7(2002), pp. 654–670.

[28] Kehrer, T., Kelter, U. and Taentzer, G.: A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning, in *Proc. of the 26th IEEE/ACM International Conference on Automated Software Engineering* (*ASE'11*), 2011, pp. 163–172.

[29] Kerievsky, J.: *Refactoring to Patterns*, Addison-Wesley, 2005.

[30] Kim, M., Cai, D. and Kim, S.: An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution, in *Proc. of the 33rd International Conference on Software Engineering* (*ICSE'11*), 2011, pp. 151–160.

[31] Kim, M., Gee, M., Loh, A. and Rachatasumrit, N.: Ref-Finder: A Refactoring Reconstruction Tool Based on Logic Query Templates, in *Proc. of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*FSE'10*), 2010, pp. 371–372.

[32] Kim, M. and Notkin, D.: Discovering and Representing Systematic Code Changes, in *Proc. of the 31st International Conference on Software Engineering* (*ICSE'09*), 2009, pp. 309–319.

[33] Kim, M., Notkin, D. and Grossman, D.: Automatic Inference of Structural Changes for Matching across Program Versions, in *Proc. of the 29th International Conference on Software Engineering* (*ICSE'07*), 2007, pp. 333–343.

[34] Kim, S., Pan, K. and E. James Whitehead, J.: When Functions Change Their Names: Automatic Detection of Origin Relationships, in *Proc. of the 12th Working Conference on Reverse Engineering* (*WCRE'05*), 2005, pp. 143–152.

[35] Lorenz, M. and Kidd, J.: *Object-Oriented Software Metrics: A Practical Approach*, Prentice Hall, 1994.

[36] Mahouachi, R., Kessentini, M. and Cinnéide, M. Ó.: Search-Based Refactoring Detection Using Software Metrics Variation, in *Proc. of the 5th International Symposium on Search-Based Software Engineering* (*SSBSE'13*), 2013, pp. 126–140.

[37] Mens, T. and Tourwé, T.: A Survey of Software Refactoring, *IEEE Transactions on Software Engineering*, Vol. 30, No. 2(2004), pp. 126–139.

[38] Mens, T. and Van Deursen, A.: Refactoring: Emerging trends and open problems, in *Proc. of the 1st International Workshop on REFactoring: Achievements, Challenges, Effects* (*REFACE'03*), 2003.

[39] Murphy, G. C., Kersten, M. and Findlater, L.: How Are Java Software Developers Using the Eclipse IDE?, *IEEE Software*, Vol. 23, No. 4(2006), pp. 76–83.

[40] Murphy-Hill, E. and Black, A. P.: Breaking the Barriers to Successful Refactoring: Observations and Tools for Extract Method, in *Proc. of the 30th International Conference on Software Engineering* (*ICSE'08*), 2008, pp. 421–430.

[41] Murphy-Hill, E., Black, A. P., Dig, D. and Parnin, C.: Gathering Refactoring Data: A Comparison of Four Methods, in *Proc. of the 2nd ACM Workshop on Refactoring Tools* (*WRT'08*), 2008.

[42] 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(2012), pp. 5–18.

[43] Murphy-Hill, E. R. and Black, A. P.: Why Don't People Use Refactoring Tools?, in *Proc. of the 1st ACM Workshop on Refactoring Tools* (*WRT'07*), 2007, pp. 60–61.

[44] 大庭晋: ソフトウェアのバージョン間で実施されたリファクタリング検出手法の改良, 修士論文, 信州大学大学院工学系研究科, 2013.

[45] Opdyke, W. F.: *Refactoring Object-oriented Frameworks*, PhD Thesis, University of Illinois, 1992.

[46] Parnin, C. and Görg, C.: Lightweight Visualizations for Inspecting Code Smells, in *Proc. of the 2006 ACM Symposium on Software Visualization* (*SoftVis'06*), 2006, pp. 171–172.

[47] Pérez, J. and Crespo, Y.: Exploring a Method to Detect Behaviour-Preserving Evolution Using Graph Transformation, in *Proc. of the 3rd International ERCIM Workshop on Software Evolution* (*EVOL'07*), 2007, pp. 114–122.

[48] Pizka, M.: Straightening Spaghetti-Code with Refactoring?, in *Proc. of the 2004 International Conference on Software Engineering Research and Practice* (*SERP'04*), 2004, pp. 846–852.

[49] Prete, K., Rachatasumrit, N. and Kim, M.: A Catalogue of Template Refactoring Rules, Technical Report UTAUSTINECE-TR-041610, The University of Texas at Austin, 2010.

[50] Prete, K., Rachatasumrit, N., Sudan, N. and Kim, M.: Template-based Reconstruction of Complex Refactorings, in *Proc. of the 26th International Conference on Software Maintenance* (*ICSM'10*), 2010.

[51] Ratzinger, J., Sigmund, T. and Gall, H. C.: On the Relation of Refactorings and Software Defect Prediction, in *Proc. of the 5th Working Conference on Mining Software Repositories* (*MSR'08*), 2008, pp. 35–38.

[52] Robbes, R. and Lanza, M.: SpyWare: A Change-aware Development Toolset, in *Proc. of the 30th International Conference on Software Engineering* (*ICSE'08*), 2008, pp. 847–850.

[53] Soares, G., Catao, B., Varjao, C., Aguiar, S., Gheyi, R. and Massoni, T.: Analyzing Refactorings on Software Repositories, in *Proc. of the 2011 25th Brazilian Symposium on Software Engineering* (*SBSE'11*), 2011, pp. 164–173.

[54] Soares, G., Gheyi, R., Murphy-Hill, E. and Johnson, B.: Comparing Approaches to Analyze Refactoring Activity on Software Repositories, *Journal of Systems and Software*, Vol. 86, No. 4(2013), pp. 1006–1022.

[55] Soares, G., Gheyi, R., Serey, D. and Massoni, T.: Making Program Refactoring Safer, *IEEE Software*, Vol. 27, No. 4(2010), pp. 52–57.

[56] Soetens, Q. and Demeyer, S.: ChEOPSJ: Change-Based Test Optimization, in *Proc. of the 16th European Conference on Software Maintenance and Reengineering* (*CSMR'12*), 2012, pp. 535–538.

[57] Soetens, Q. D., Pérez, J. and Demeyer, S.: An Initial Investigation into Change-Based Reconstruction of Floss-Refactorings, in *Proc. of the 29th IEEE International Conference on Software Maintenance* (*ICSM'13*), 2013, pp. 384–387.

[58] Stroggylos, K. and Spinellis, D.: Refactoring–Does It Improve Software Quality?, in *Proc. of the 5th International Workshop on Software Quality* (*WoSQ'07*), 2007.

[59] Taneja, K., Dig, D. and Xie, T.: Automated Detection of API Refactorings in Libraries, in *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering* (*ASE'07*), 2007, pp. 377–380.

[60] Thangthumachit, S., Hayashi, S. and Saeki, M.: Understanding Source Code Differences by Separating Refactoring Effects, in *Proc. of the 18th Asia-Pacific Software Engineering Conference* (*APSEC'11*), 2011, pp. 339–347.

[61] Weißgerber, P. and Diehl, S.: Identifying Refactorings from Source-Code Changes, in *Proc. of the 21st International Conference on Automated Software Engineering* (*ASE'06*), 2006, pp. 231–240.

[62] Xing, Z. and Stroulia, E.: UMLDiff: An Algorithm for Object-oriented Design Differencing, in *Proc. of the 20th International Conference on Automated Software Engineering* (*ASE'05*), 2005, pp. 54–65.

[63] Xing, Z. and Stroulia, E.: Refactoring Detection Based on UMLDiff Change-Facts Queries, in *Proc. of the 13th Working Conference on Reverse Engineering* (*WCRE'06*), 2006, pp. 263–274.

[64] Xing, Z. and Stroulia, E.: Differencing logical UML models, *Automated Software Engineering*, Vol. 14, No. 2(2007), pp. 215–259.

崔 恩 瀞

2012 年大阪大学大学院情報科学研究科博士前期課程修了．現在同研究科博士後期課程に在籍．コードクローン管理やリファクタリング支援手法に関する研究に従事．ACM，情報処理学会各会員．

藤 原 賢 二

2010 年大阪府立工業高等専門学校総合工学システム専攻修了．2012 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．現在同大学情報科学研究科博士後期課程に在籍．修士 (工学)．リファクタリングの適用履歴分析，プログラミング教育支援に興味を持つ．情報処理学会，電子情報通信学会，IEEE 各会員．

吉 田 則 裕

2004 年九州工業大学情報工学部知能情報工学科卒業．2009 年大阪大学大学院情報科学研究科博士後期課程修了．同年日本学術振興会特別研究員 (PD)．2010 年奈良先端科学技術大学院大学情報科学研究科助教．2014 年より名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．博士 (情報科学)．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

林 晋 平

2004 年北海道大学工学部情報工学科卒．2006 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．2008 年同専攻博士後期課程修了．2009 年より同専攻助教，現在に至る．博士 (工学)．ソフトウェア進化やソフトウェア開発環境などの研究に従事．IEEE Computer Society，ACM，情報処理学会各会員．