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.

# ソースコードに対して適用可能な編集手順を探索するリファクタリング支援手法の提案

譜久島　亮† 　吉田　則裕† 　松下　誠† 　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
E-mail: †{r-fukusm,n-yosida,matusita,inoue}@ist.osaka-u.ac.jp

**あらまし**　リファクタリングとは，ソースコードの外部的振る舞いを保ちながら，内部構造を改善する作業である．頻繁に行われるリファクタリングとして，メンバの移動が挙げられる．多くの場合，メンバの移動を行うための編集手順は複数存在し，各編集手順を適用した結果のソースコードはそれぞれ異なる．しかし，リファクタリングの経験が少ない開発者の場合，どのような編集手順が考えられるかを十分に理解しないままソースコードを編集し，不必要な編集を行ったり，欠陥を作り込んだりする可能性が考えられる．そこで本研究では，メンバの移動を行う際に適用可能な編集手順を，自動的に導出する手法を提案し，Eclipse プラグインとして実現した．
**キーワード**　オブジェクト指向プログラミング，リファクタリング，リファクタリングパターン，Eclipse プラグイン

# Refactoring Support by Searching Applicable Steps to Change Source Code

Ryo FUKUSHIMA†, Norihiro YOSHIDA†, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka, 565-0871 Japan
E-mail: †{r-fukusm,n-yosida,matusita,inoue}@ist.osaka-u.ac.jp

**Abstract**　Refactoring is the process that can improve software's internal structure without changing its external behavior. One of refactoring patterns that are frequently applied is "Move Method". When developers move a member in a class to another class, they often have several options for applicable procedure to change source code, and each applicable procedure provides different resultant source code. However, developers who have limited refactoring experience often do not understand applicable procedure to change source code for "Move Method" refactoring. Such developers have the potential to perform unnecessary editing or write defective code. In this paper, we propose an Eclipse plug-in to derive applicable procedures to perform "Move Method" refactoring.
**Key words**　Object-Oriented Programming, Refactoring Support, Refactoring Pattern, Eclipse Plug-in

## 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保ちながら，内部構造を改善する作業のことである [1], [2]．メソッドの移動やフィールドの移動などの繰り返し行われているとされるリファクタリングは，パターンとして書籍 [2] やウェブサイト [3] にまとめられている．このようなパターンはリファクタリングパターンと呼ばれ，適用を検討すべきソースコードの特徴や適用手順などをまとめられている．例えば，Fowler のウェブサイト [3] には 93 種類のリファクタリングパターンが掲載されている．リファクタリングの経験が少ない開発者は，このようなリファクタリングパターンから熟練者が行うリファクタリングを学習することができる．

リファクタリングは開発者が手作業で行うソースコード変換 [4] であるため，各リファクタリングパターンが含む適用手順は，ソースコードの変換手順を表していると考えることができる．

しかし，リファクタリングパターンとそのリファクタリングパターンを適用するため指定する必要のあるクラスやメンバ（移動対象のメンバや移動先のクラスなど）が決まっている場合であっても，リファクタリング後のソースコードは複数考えられることがある．図 1 は，リファクタリングパターンの 1 つ

であるメソッドの移動の適用を表している．この図では，クラス A 中の A.add メソッドをクラス B に移動している．

メンバの移動では，移動するメンバが参照しているフィールド (A.p) の移動や，移動後のフィールドへの参照方法 (*getter*, *setter* の追加等) を検討する必要がある． リファクタリング後のソースコードの一例として図 1(b) や図 1(c) が考えられるが，適用した編集手順は異なり，得られるソースコードも異なる．

リファクタリングの経験が多い開発者であれば，リファクタリング後のソースコードを理解した上で，編集手順を考案することができるが，経験が少ない開発者の場合，リファクタリング後のソースコードを十分理解しないまま作業を行う可能性がある．そのため，どのような編集手順が考えられるかを十分に理解しないままソースコードを編集し，不必要な編集を行ったり，欠陥を作り込んだりする可能性が考えられる．

そこで著者らは，開発者が行いたいリファクタリングのシナリオが与えられたとき，適用可能な編集手順の候補や，およびリファクタリング後のソースコードの候補を提示するツールの実現を目指している．リファクタリングのシナリオは，開発者が行いたいリファクタリングを指定するためのリファクタリングパターン，クラスやメンバ（例えば移動対象のメンバや移動先のクラス）からなる．図 2 を例にとり著者らが実現を目指すツールを説明すると，“図 1(a) のクラス A に所属するメソッド A.add をクラス B に移動する ”というシナリオを入力すると，図 2 に示す適用可能な編集作業を表す木，および図 1(b)，図 1(c) 等のリファクタリング後のソースコードを出力するツールを実現したいと考えている．

本稿では，Java 言語におけるメンバの移動に着目し，1 つのメンバを他のクラスへ移動する際に適用可能な編集手順の候補や，リファクタリング後のソースコードの候補を提示する手法を提案する．メンバの移動に着目した理由は，メソッドの移動やフィールドの移動が頻繁に適用されるリファクタリングパターンであるからである [5].

提案手法は，対象とするメンバを他のクラスへ移動した際に，メンバ間の参照・被参照の関係が成立しなくなることが原因で発生するコンパイルエラーを検出し，そのようなコンパイルエラーを除去するための編集手順を探索的に求めることで，適用可能な編集手順の候補（図 2）や，およびリファクタリング後のソースコードの候補（図 1(b)，図 1(c)）を導出する．

ケーススタディを行うために，提案手法に基づいて private メンバの移動を支援する Eclipse プラグイン [6] を作成した．ケーススタディでは，作成した Eclipse プラグインを図 1 に示すリファクタリングに適用した．その結果，図 2 において示した適用可能な編集手順の候補，およびリファクタリング後のソースコードの候補を導出できていることを確認できた．

以下，2 章では，リファクタリング支援の問題点について述べ，3 章で提案手法について述べる．4 章では提案手法の実装概要について述べ，5 章でケーススタディを用いて手法の有効性を検証する．6 章でまとめと今後の課題について述べる．

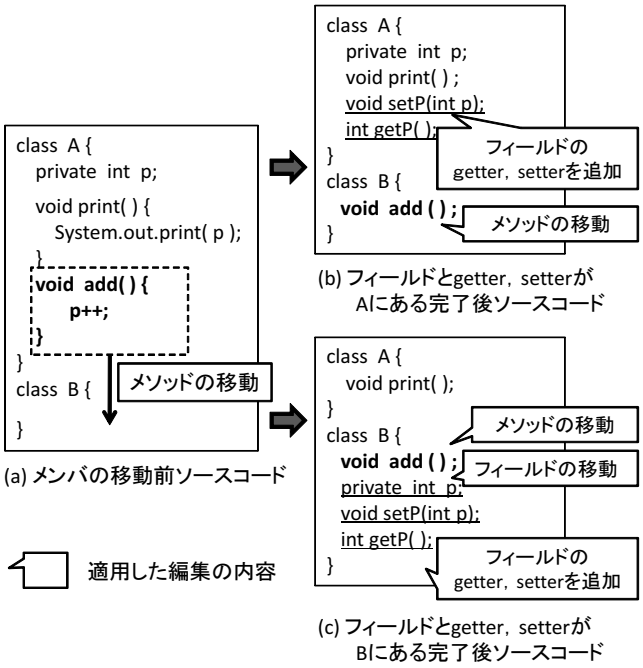


図 1 “メンバの移動”リファクタリングによる完了後のソースコード例（完了後のソースコードに含まれるメソッド本体は省略）

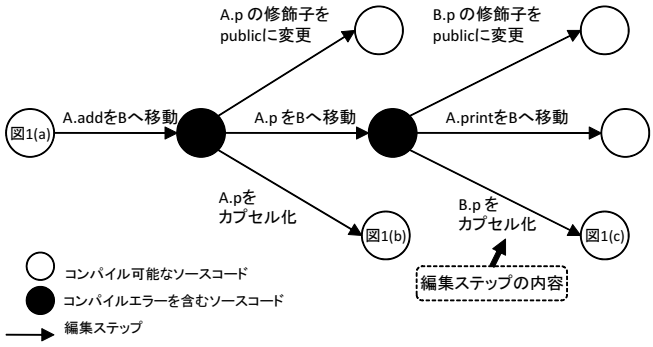


図 2 “メンバの移動”に伴う編集作業の木の例

## 2. 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的振る舞いを保ちながら内部構造を改善する作業である [1], [2]．文献 [2] では，典型的なリファクタリングがリファクタリングパターン（以降，パターン）としてまとめられている．ここでは，“メンバの移動” パターンに注目して，パターンに基づく編集作業や編集作業を支援するツールの問題点を述べる．

### 2.1 メンバの移動

“メンバの移動” リファクタリングは，あるメンバが不適切なクラスで宣言されている場合，適切なクラスへメンバを移動することで，クラスの責任を明確にするリファクタリングである．

“メンバの移動” により，メンバ間の参照・被参照の関係が成立しなくなる場合がある．これは，private メンバを参照していたメンバの移動や，他のメンバから参照されていた private メンバの移動により起こることが多い．

Java 言語における private メンバは，他のクラスから参照す

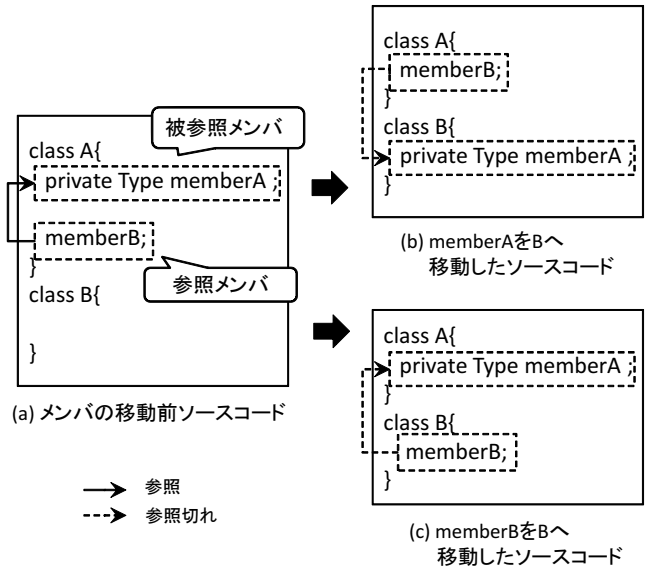


図 3　メンバの移動により参照切れになる例（$memberB$ はメソッドやフィールドが考えられる．本体は省略する．）

ることができない．ここで，$memberA$ が $memberB$ から参照されているとき，$memberA$ を “被参照メンバ”，$memberB$ を “参照メンバ” とする（図 3(a)）．図 3 のように被参照メンバが private メンバである場合，“メンバの移動” により被参照メンバや参照メンバを移動すると，参照メンバと被参照メンバが異なるクラスに所属し，参照メンバから被参照メンバへ参照ができない（以降，参照切れと呼ぶ．図 3(b),(c) で例を示す）．この場合，開発者は以下に挙げる編集のいずれかを選択し，編集をソースコードに適用して参照切れを解決する必要がある．

**編集 1**　被参照メンバの修飾子を変更する．

**編集 2**　被参照メンバが private フィールドの場合，フィールドのカプセル化を行い，参照メンバからカプセル化によって追加した *getter*, *setter* を介して被参照メンバを参照する．

**編集 3**　移動した参照メンバ（もしくは被参照メンバ）の所属するクラスへ被参照メンバ（もしくは参照メンバ）を移動する．

図 4(a) において，A.add と A.print は A.p を参照しているため，A.add と A.print は参照メンバ，A.p は被参照メンバである．図 4(a) で，A.add を B へ移動する場合，A.add の被参照メンバである A.p は private メンバであるため， A.add を B へ移動後，図 4(b) で B.add から A.p へ参照できない． 図 4(b) の参照切れに対して，A.p の修飾子の変更（編集 1）を適用したのが図 4(c)，A.p のカプセル化（編集 2）を適用したのが図 4(d)，被参照メンバである A.p の移動（編集 3）を適用したのが図 4(e) である．図 4(e) では，private な被参照メンバである B.p は A.print から参照できない．よって，A.p の移動を選択し図 4(b) に適用した開発者は，図 4(e) の参照切れに対して，さらに編集 1〜3 のいずれかを選択しソースコードに適用しなければならない．

このように，ソースコードに適用する編集手順は，ソースコードや開発者の意図によって複数存在し，各編集手順から導かれるソースコードはそれぞれ異なるものである．

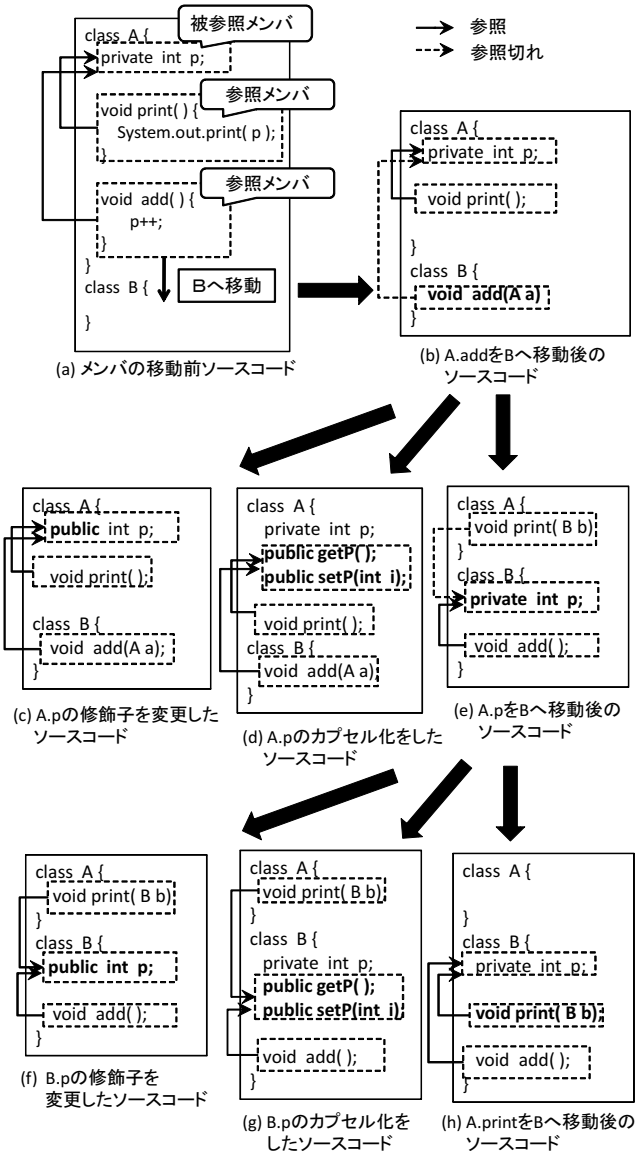


図 4　メンバの移動に伴う参照切れを編集する例（(b)〜(h) のメソッド本体は省略）

また，図 4 のようなメンバの移動に伴う編集作業は，ソースコードを頂点，編集手順を構成する各ステップ（以降，編集ステップ）を有効辺とする木構造で表現できる．木の根はリファクタリング開始時のソースコード，葉はリファクタリング完了後のソースコードを指し，道は編集手順の 1 つを表す．図 2 は図 4(a) の “メンバの移動” に伴う編集作業を表す木であり，リファクタリングを行う際，開発者は道を選択することで編集手順を決定できる．

### 2.2　リファクタリング支援ツールの問題点

リファクタリングに伴うソースコードの編集手順は，複雑な編集ステップで構成されていることが多い．そこで，ソースコードに対する編集ステップの適用を支援するツールが提供されている．統合開発環境 Eclipse では，ソースコードに適用する編集ステップの適用を支援する機能が提供されている．選択したメソッドを指定したクラスへ移動する編集や，選択したフィールドのカプセル化のために *getter*, *setter* を追加する編

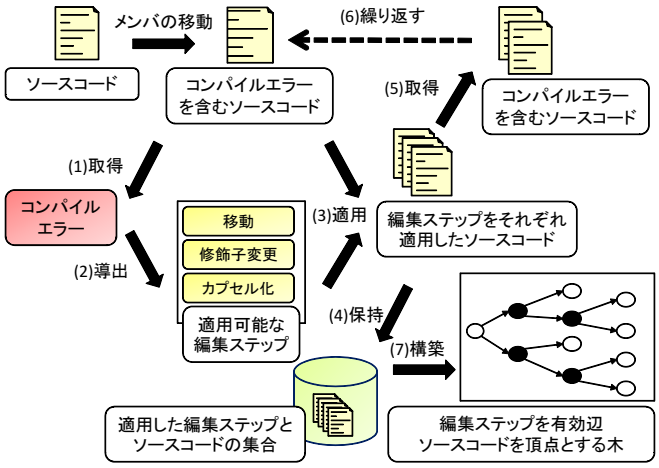


図 5　提案手法の概要図

集などを半自動的に行うことができる．

リファクタリング経験が豊富な開発者は，半自動化されている編集ステップを利用して，目的とするソースコードまでの編集手順を考案し，ソースコードを編集することができるが，リファクタリング経験の乏しい開発者にとって，これは困難である．

既存のリファクタリング支援ツールの中で，ソースコードに適用可能な編集手順が複数ある場合に図 2 のような編集作業の木を提示できるものが確認できないため，リファクタリング経験の少ない開発者は，どのような編集手順が考えられるかを十分に理解しないままソースコードを編集し，“目的のソースコードを得ることができない”，“編集の手戻りにより作業時間が大きくなる” という問題が生じる可能性がある．

## 3. 提 案 手 法

本研究では，private メンバを参照しているメンバの移動に伴う複数の編集手順を探索する手法を提案する．メンバの移動により発生した参照切れを解決する編集ステップを自動的に導出，適用することでソースコードに対する適用可能な編集手順を探索する．図 5 で示すように，以下のプロセスで編集手順を探索する．

（1） メンバの移動を適用したコンパイルエラーを含むソースコードからコンパイルエラーを取得

（2） コンパイルエラーを解決する編集ステップを導出

（3） ソースコードに編集ステップを適用

（4） 編集ステップと，その適用結果のソースコードを保持

（5） コンパイルエラーを含むソースコードを取得

（6） コンパイルエラーを含むソースコードに対し，繰り返し編集ステップを導出，適用する

（7） 編集ステップを有効辺，ソースコードを頂点とする編集作業の木を構築する

本章では，まず編集ステップについて説明し，(2) 編集ステップの導出，(3) 編集ステップの適用を具体例を用いて説明する．

### 3.1 編集ステップ

メンバを移動後，参照メンバと private な被参照メンバが異なるクラスに属する場合，参照メンバ内で参照切れが起こる．説明のため，被参照メンバである $memberA$ と参照メンバである $memberB$ が $classA$ に所属し，$classA$ から $classB$ へメンバを移動後，参照切れが起こったとする（$memberA$ を移動したソースコードと，$memberB$ を移動したソースコードが考えられる）．参照切れから以下を特定する．

- $memberA$ が所属するクラス
- $memberB$ が所属するクラス

これらを元にして，参照切れを解決する以下の編集ステップを導出する．

**メンバの移動:** $memberB$ を $classB$ へ移動したことより参照切れが発生した場合，$memberA$ が所属するクラスから $memberB$ が所属するクラスへ $memberA$ を移動する（図 6(a)）．$memberA$ を $classB$ へ移動したことより参照切れが発生した場合，$memberB$ が所属するクラスから $memberA$ が所属するクラスへ $memberB$ を移動する（図 6(b)）．移動前のメンバへの全ての参照を，移動後のメンバへの参照に変更する．

**変数のカプセル：** $memberA$ がフィールドの場合，$memberA$ のカプセル化を行い，$memberA$ への全ての参照を $getter$, $setter$ を介する間接参照に変更する．(図 6(c))

**修飾子の変更:** $memberA$ の修飾子を public に変更する．(図 6(d))

メンバの移動の編集ステップは，過去に適用したメンバの移動により異なる編集ステップが導出されるが，これは過去に移動したメンバを元のクラスに移動する編集ステップの導出を避けるためである．

### 3.2 編集ステップの導出と適用

編集ステップの導出と適用の様子を具体例で説明する．図 4(b) は，図 4(a) において A から B へ参照メンバである A.add を移動した，参照切れを含むソースコードである．コンパイルエラーから，

- 被参照メンバ（A.p）が所属するクラス：A
- 参照メンバ（B.add）が所属するクラス：B

となり図 4(b) のソースコードに対して導出される編集ステップは以下になる．

**メンバの移動:** B.add が所属している B へ A.p を移動

**変数のカプセル化:** A.p のカプセル化

**修飾子の変更:** A.p の修飾子を public に変更

導出された編集ステップをそれぞれソースコードに対して適用する．“修飾子の変更”，“変数のカプセル化” を適用した結果がそれぞれ図 4(c)，(d) であるが，参照切れが解決されたソースコードとなる．しかし，“メンバの移動” を適用した図 4(e) のソースコードは，A.print と B.p の間で参照切れが起こる．よって，図 4(e) のソースコードに対して，繰り返し編集ステップの導出・適用を行う．図 4(e) のコンパイルエラーより，

- 被参照メンバ（B.p）が所属するクラス：B
- 参照メンバ（A.print）が所属するクラス：A

となり，導出される図 4(e) のソースコードに対して編集ステップは以下になる．

**メンバの移動:** B.p が所属している B へ A.print を移動

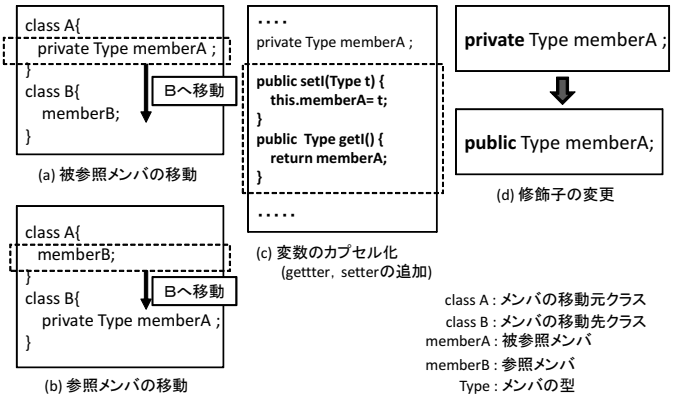


図 6　編集ステップ

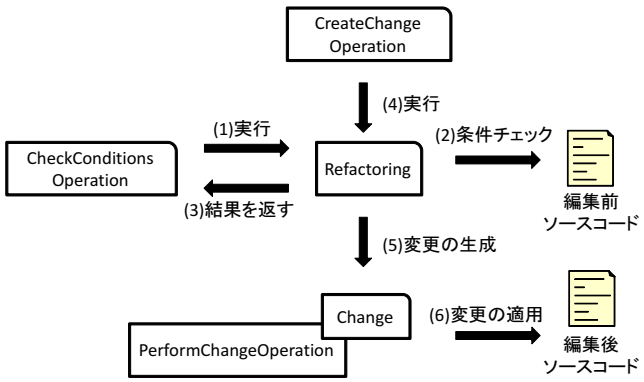


図 7　Eclipse のソースコードに対する編集適用の流れ

**変数のカプセル化:**　B.p のカプセル化
**修飾子の変更:**　B.p の修飾子を public に変更

導出された編集ステップを図 4(e) に適用した結果である図 4(f)，(g)，(h) のソースコードは，すべて参照切れが解消されたものとなる．

編集ステップと、編集ステップを適用したソースコードは保持しておき，ソースコードを頂点，編集ステップを有効辺とする編集作業の木を構築する（図 2）．開発者は，編集作業の木の道を選択することで，メンバの移動に伴う編集手順の選択ができる．

## 4. 実　　装

提案手法を Eclipse プラグインとして実装する．Eclipse で提供されているリファクタリング支援機能を利用して実装を行う．提案手法を実装するために使用した API について説明する．また，Eclipse で提供されているソースコードのコンパイルエラーの取得のための API について説明する．

### 4.1　編集ステップの導出・適用で利用する API

Eclipse のリファクタリング機能の開発のために，LTK(Language Tool Kit) と呼ばれる API 群が提供されており，提案手法の実装に利用する API は，LTK のパッケージである org.eclipse.ltk.core.refactoring で提供されている以下のものである．提案手法で定めた編集ステップの実装は，以下で説明する Refactoring クラスを実装することで実現できる．

Refactoring:　リファクタリング支援機能で提供されている編集作業のプロセスがまとめられているクラス．ソースコードに対して編集作業を行う際の条件チェックや，適用する編集内容が記述された変更オブジェクトの生成を行う．提案手法で定めた編集ステップの生成は，このクラスを継承したクラスを実装し，変更オブジェクトの生成を行うことで実現できる．

Change:　ソースコードに対して適用する編集内容が記述されたクラス．Refactoring クラスのインスタンスから生成される．編集ステップの内容を保持する変更オブジェクトである．

CheckConditionsOperation:　Refactoring クラスのインスタンスを操作して，条件チェックの実行を行うクラス．

CreateChangeOperation:　Refactoring クラスのインスタンスを操作して，変更オブジェクトの生成を実行するクラス．Change クラスのインスタンスの生成を実行する．

PerformChangeOperation:　CreateChangeOperation クラスのインスタンスにより生成された Change クラスのインスタンスをソースコードに適用する処理を実行するクラス．

図 7 で処理の流れを示す．Refactring クラスのインスタンスを用いて，CheckConditionsOperation クラスのインスタンスによって実行される条件チェックや，CreateChangeOperation クラスのインスタンスによって実行される変更の生成を行う．変更の生成によって Change インスタンスが生成された後，PerformChangeOperation クラスのインスタンスによってソースコードに対する変更の適用が実行される．

### 4.2　コンパイルエラーの取得

Eclipse では，ソースコードの構造解析や意味解析の結果を保持した抽象構文木を利用するための API が Eclipse プラグインである org.eclipse.jdt.core の org.eclipse.jdt.core.dom パッケージで提供されている．

ASTParser:　ソースコードの構造解析，意味解析の結果を保持する抽象構文木を生成するための機能を提供するクラス．

CompilationUnit:　ASTParser によって生成される抽象構文木に該当するクラス．CompilationUnit よりソースコードのコンパイルエラーが取得できる．

提案手法では，抽象構文木より取得したコンパイルエラーを元に編集ステップの導出を行う．

## 5. ケーススタディ

図 4 のメンバに移動に伴う編集作業を例に，提案手法を用いて編集手順の探索を行った．その結果，参照メンバ，被参照メンバ間の参照切れを除去した結果である複数のソースコードを得ることができた．また，参照切れが解決されたソースコードを取得する図 2 で示す各編集手順を探索できた．

図 4 の編集作業は，開発者が選択したメンバを移動し，移動対象メンバと参照・被参照の関係にあるメンバ間で参照関係が正しいソースコードを取得する状況を想定している．このような状況で，開発者の目的は，選択したメンバの移動であり，メンバの移動に伴い発生した参照切れを解決する編集ステップを選択し，参照切れを除去していく編集作業は，開発者にとって

大きな負荷となる．参照切れを除去するための編集ステップをソースコードに適用した結果，新たな参照切れが発生する場合（図 4(e)）があり，開発者は再度，参照切れを除去する編集ステップの選択に迫られ，直前の編集ステップの適用取り消しによる編集作業の手戻りが起こり，目的とするソースコードが得にくいからである．

よって，ソースコードに適用可能な編集手順を自動的に探索する本手法は，メンバの移動の結果である複数のソースコードが取得できており，編集作業の手戻りが起こらないので有用と言える．

提案手法では，メンバの移動に伴い発生した参照切れを除去した複数のソースコードを取得し，そのソースコードを得るための編集手順を探索した．リファクタリング支援のためには，メンバの移動の結果である参照切れが除去された複数のソースコードの中から，開発者が選択を行う支援をするユーザインターフェースが必要である．

## 6. まとめと今後の課題

本研究では，メンバの移動の中でも，private メンバや，private メンバを参照するメンバの移動に伴い発生する参照切れを除去する編集手順の探索を行った．参照切れを除去するための編集ステップとして，他のメンバの移動や，被参照メンバのカプセル化，被参照メンバの修飾子の変更を挙げた． また編集ステップを実際に参照切れが発生しているソースコードに適用し，参照切れを除去した複数のソースコードを得ることができた．

提案手法では，ソースコードに適用する編集ステップは，参照切れを解決するという目的のもと導出した．しかしながら，文献 [2] で定められている “メンバの移動”パターンでは，メンバの移動を行う前に “移動するメンバと関連のあるメンバについて移動を検討する必要がある”としている．“メンバの移動”パターンで定められた検討によって，移動するメンバと関連のある public なメンバの移動を行うこともあるが，本手法では，public なメンバを移動する編集ステップは導出できない．本手法で導き出される編集作業の木は，public なメンバの移動も考慮した場合の編集作業の木の部分木と言える．今後は，public なメンバの移動も考慮した編集ステップの導出も行う．

また今後は，提案手法で探索した，メンバの移動に伴う適用可能な編集手順と，編集手順を適用したソースコードを開発者に提示することも行う．

**謝辞** 本研究は一部，日本学術振興会科学研究費補助金基盤研究（A）（課題番号:21240002），文部科学省科学研究費補助金若手研究（B）（課題番号:20700024），日本学術振興会特別研究員奨励費（課題番号:20・1964）の助成を得た．

### 文　献

[1] W. F. Opdyke: “Refactoring object-oriented frameworks”, PhD thesis, University of Illinois at Urbana-Champaign (1992).
[2] M. Fowler: “Refactoring: improving the design of existing code”, Addison Wesley (1999).
[3] M. Fowler: “http://refactoring.com”.
[4] T. Mens and T. Tourwé: “A survey of software refactoring”, IEEE Trans. Sofw. Eng., **30**, 2, pp. 126–139 (2004).
[5] G. C. Murphy, M. Kersten and L. Findlater: “How are java software developers using the eclipse ide?”, IEEE Softw., **23**, 4, pp. 76–83 (2006).
[6] Eclipse: “http://eclipse.org”.