Vol.2009-SE-166 No.5
2009/11/5

プログラム変更支援のための再利用コンテキスト収集ツール

伏井洋平[†1]　大森隆行[†2]　丸山勝久[†2]

開発者の持つ知識は，ソフトウェア開発において有用な資産である．その中でも，本研究ではコード片の再利用に関する知識に着目する．コード片の再利用では，一般的に開発者はコード片の修正を行う．本論文では，コード片を再利用する際のソースコードの状況や修正を再利用コンテキストと呼び，これを半自動的に収集する仕組みとその実装を提案する．再利用コンテキストを効率的に収集することで，コード片に関する知識を開発者間で共有することが可能となり，従来とは異なるプログラム変更支援が期待できる．

A Tool Collecting Reuse Contexts to Support Program Modification

Yohei Fushii,[†1] Takayuki Omori[†2] and Katsuhisa Maruyama[†2]

Developers' knowledge is one of assets that help their software development. Our work focuses on knowledge about reuse of code snippets in software construction. A developer often brings code snippets from elsewhere and then in general modifies them so that they meet his/her requirement. In this paper, a situation deriving such reuse and derived from it is called a reuse context. It includes the reused snippets and their modifications. The paper illustrates a mechanism for semi-automatically collecting reuse contexts during software construction processes, and proposes a tool implementing the mechanism. A collection of reuse contexts promotes the sharing of various knowledge of many developers, and might foster new kind of support for program modification.

†1 立命館大学 大学院 理工学研究科 Graduate School of Science and Engineering, Ritsumeikan University
†2 立命館大学 情報理工学部 Dept. of Computer Science, Ritsumeikan University

1. はじめに

開発者が持つ知識は，ソフトウェア開発において有用な資産であると考えられる．開発者の知識としては，具体的には，ソースコードのもつ性質に関する知識や，ソースコード間の関係に関する知識などが考えられる．例えば，ある開発者の持つ特定のコード同士が強く関係しているという知識が利用できれば，別の開発者がそれらのコード同士の関係に注意しながらプログラム変更を行うことが可能になり，よりバグを生みにくいソフトウェア開発を行うことができる．こういった開発者の知識の中でも，本研究では数行のコードをコピーアンドペーストすることで行われるコード片の再利用によって得られる知識に着目する．

一般的に，コード片の再利用は次のような手順で行われていると考えられる．まず開発者は，これから作成するコードを，既存のコードの再利用によって記述すべきかを判断する．再利用すべきと判断すれば，開発者はプロジェクトのリポジトリや Web ページなどから自分の要求にあったコードを検索する．コードを発見できれば，発見したコードの必要な部分を選択してコピーし，自分が作成しているソースコードにペーストする．通常そのままでは目的通りにプログラムが動作しないことが多いため，プログラムが動作するようにソースコードに対して変更を加える．

このようにコード片の再利用において，開発者は再利用に関係するソースコードを調査して理解したり，修正したりすることになる．この調査や修正の過程で開発者が得る知識は，ソースコードの特定の部分・側面のみに関する知識であると考えられる．つまり，再利用時に開発者が得る知識を取り出すことで，ソースコードの特定の部分のみに限った知識を扱うことができる．開発者の知識を対象のソースコードの部分・側面によって分類することで，これまでにないプログラム変更支援が可能になる．しかし，現在のところ，コード片再利用時の開発者の知識を利用すること容易ではない．バージョン管理システムの持つ履歴情報や，開発者の残したドキュメントからその一部を手に入れることはできるが，情報が不十分であったり，情報を記録するのに手間がかかったりするなどの問題がある．

そこで，本研究では，コード片再利用時の開発者の知識を得るために，再利用時のコンテキストを半自動的に収集し，集めた情報から開発者の知識を取得する仕組みの提案と，それを実現するためのツールの開発を行う．ソフトウェア開発に関するコンテキストを扱った研究には，プログラミングのタスクや開発者の関心事に関するコンテキストを対象にしたものが存在する[1)2)]．しかし，これらの研究では再利用に特化しておらず，再利用コンテキストを取得するための新しい仕組みが必要となる．

また，本研究では，再利用コンテキスト取得の仕組みに加えて，取得した再利用コンテキストを用いた，プログラム変更支援ツールを提案する．このプログラム変更支援ツールは再利用コンテキストから得られる再利用用毎の開発者の知識を利用することで，プログラミング中の開発者に，その場面で利用できる可能性のあるコード片や，そのコードを再利用するために必要な変更例を提示することで，プログラムの変更を支援する．

以降，本論文では，まず，2 章でコード片の再利用について説明する．次に 3 章で，再利用コンテキストを収集する方法と，それを実現するためのツールの実装述べる．また，4 章では，再利用コンテキストから取り出した開発者の知識を活用したプログラム変更を支援する手法について説明する．最後の 5 章で研究の今後とまとめを述べる．

2. コード片の再利用

コード片とは，ソースコード中の特定のひとかたまりの処理を実現する数行から数十行のコードである．本研究におけるコード片の再利用とは，コード片を他のソースコードに流用することを指す．コード片は再利用を前提としていないため，他のコードに流用した際に直接ソースコードに変更を加える必要があることがほとんどである．

コード片の再利用は，多くの場合エディタの持つコピーアンドペースト機能を利用して行われる．コード片の再利用の例を図 1 に示す．この例では，Java におけるファイルから文字を 1 文字ずつ読み込むためのコード片を再利用している．読み込むファイルや読み込んだ文字に対して行う処理が目的にそぐわなかったため，変更を加えている．

本研究では，上記のようなコード片の再利用を以下の 4 種類の操作の組み合わせであると定義する．

(1) 再利用元コードの選択
　コード片を選択してコピーする
(2) 再利用先コードの選択
　コード片をペーストする
(3) コードの変更
　再利用したコードが動作するように変更を加える
(4) 再利用の終了
　以下のいずれかの操作を行うこと
- ソースコードのセーブ

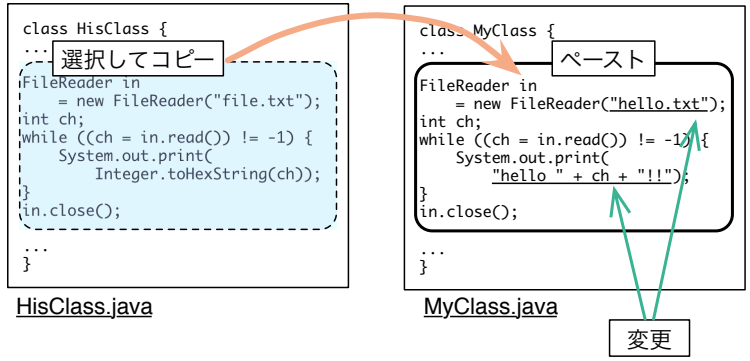

図 1 コード片の再利用

- エディタのクローズ
- エディタの非アクティブ化
- ツールによる再利用終了の指示

もっとも単純な再利用であれば，上記の順番通り，再利用元を選択したあと再利用先を選択し，コードに変更を加えたあと再利用を終了することになる．この様子を図 2 に示す．再利用に関わるソースコードについては，以降，再利用されたコード片自体を 再利用コード片，再利用されたコード片がもともと記述されていたソースコードを 再利用元コード，再利用されたコード片が埋め込まれたソースコードを 再利用先コード と呼ぶ．

再利用を構成する 4 つの操作のうち，再利用の終了に対応する操作が複数になっているのは，本来のコード片の再利用では，開発者が再利用が終了したときに必ず特定の操作を行うとは限らないと考えられるからである．そのため，本研究では，利用の終了と見なしうる操作をすべて取り扱うことにしている． ただし，ツールによる再利用終了の指示は本来の再利用に必要な操作ではなく，明示的に再利用を終了するツールの機能を指す．このツールについては後述する．

コード片を再利用するにあたって，開発者は以下のような活動を行っていると考えられる．

- 再利用が必要かの判断
　コードを作成するのに再利用すべきかの判断．

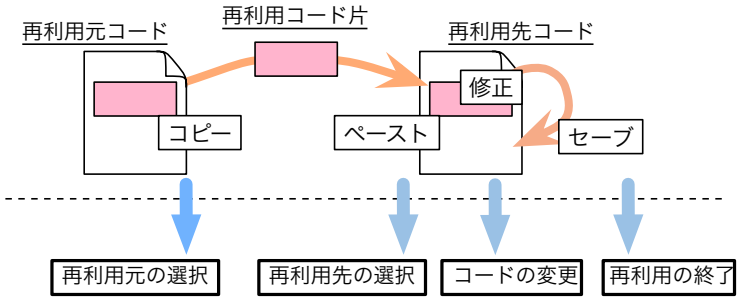

図 2 再利用を構成する 4 つの操作

- 再利用するコードの選出
  ソースコードリポジトリや Web ページなどを検索することによって行う，再利用するコードの選出．
- 再利用する箇所の判断
  どの部分のコードをコード片として取り出し再利用するかの判断．
- ソースコードの変更
  再利用時にプログラムが動作するよう行うソースコードの変更．

再利用を行う開発者は上記の活動の中で，再利用に関わったソースコードを調査して理解したり変更方法を試行錯誤することで，さまざまな知識を得ることになる．本研究では以下を再利用ごとの開発者の知識として取り扱う．

- 再利用コード片の機能
- 再利用コード片の利用方法
- 再利用コード片と再利用元コード，再利用先コードの関係

3. 再利用コンテキストの収集手法

本章では，2 章で述べたコード片を行った開発者の得た知識を，再利用ごとに取得する手法について説明する．本手法では，開発者がコード片の再利用を行ったことを自動的に検知し，後述する再利用コンテキストを記録する．記録された再利用コンテキストには開発者知識の一部が含まれる．これらを分析することで開発者の知識を取得し，プログラム変更支援に役立てる．

3.1 再利用コンテキスト

本手法では，開発者が再利用を行っている際のコンテキストを再利用コンテキストと呼ぶ．本研究では以下の情報を再利用コンテキストとして記録する．

- 再利用の目的
  開発者がコード片を再利用した目的を指す．再利用の目的としては，信頼性の高いコードを利用するため，高速なコードが必要だったため，典型的な処理であったため等が考えられる．再利用の目的は間接的に再利用されたコード片の性質を示すと考えられる．

- 再利用に関わったソースコード
  コード片の再利用が行われたときに，それに関わったソースコードの場所と内容を指す．再利用に関わったソースコードとは，図 2 に示した，再利用コード片，再利用元コード，再利用先コードの 3 つを指す．

- 再利用に伴うソースコードの変更
  コード片の再利用が行われたときに，行われたソースコードに対する変更を指す．

再利用コンテキストを単に記録するだけでは，十分な開発者の知識を得ることはできない．しかしながら，記録した情報を分析することで，開発者の知識の一部を取り出すことができる．例えば，2 章で述べた開発者の知識の一つである，”再利用コード片と再利用元コード，再利用先コードの関係”は再利用に関わったソースコードを解析することで得ることができる．

3.2 再利用コンテキストの取得

本手法では，再利用が行われたことを検知して，再利用コンテキストを半自動的に取得する．本手法では 2 章で図 2 を用いて説明した，再利用を構成する 4 つの操作を検知し操作の種類に応じて開発環境から必要な情報を取得する．以下では，再利用コンテキストを構成する，再利用の目的，再利用に関わったソースコード，再利用時に行われたソースコードのそれぞれについて取得方法を説明する．

まず，再利用に関わったソースコードを取得する方法について述べる．再利用に関わったソースコードは，上記の再利用を構成する 4 つの操作で対象となっているソースコードと

して取得できる．再利用元の選択をおこなったときに開発者が選択してコピーしたコード片は再利用コード片として取得する．また，そのコード片のコピー元となったソースコードを再利用元コードとして取得する．同様に再利用先の選択時にペースト操作を行った対象ソースコードを再利用先コードとして取得する．

次に，再利用に伴うソースコードの変更を取得する方法について述べる．再利用に伴うソースコード変更は，再利用先の選択から再利用の終了の間に再利用先コードに対して行われた変更のことを指す．本手法では，再利用先コードの選択の直後と再利用の終了が行われる時のそれぞれのタイミングで再利用先コードの内容を記録しておき，それらを比較することで再利用時に行われたソースコードの変更を取得する．

最後に再利用の目的を取得する方法について述べる．再利用コンテキストのうち上記 2 つは自動的に取得することができる．しかし，この再利用の目的に限っては自動的に取得することはできない．なぜなら，再利用の目的は開発者が再利用時に何を考えていたかに依存するからである．本手法では，再利用終了時に開発者に再利用の目的を手動で入力することが可能な機能を用意する．基本的には開発者が自然言語を用いて入力するほか，分析して利用しやすいように再利用の目的を表すタグや，コード片のレーティングも入力可能にする．

3.3 再利用コンテキストの記録

上記の手法で取得された再利用コンテキストは再利用ごとに別々に取り扱えるように記録する必要がある．本手法では，各操作で取得された情報を記録するときに，その操作が行われた日時や，操作を行った開発者名，操作に関わったファイルの場所を記録しておく．これらの情報を活用し，日時が連続しており，開発者名やファイルの場所が同じものを一つの再利用に対応した再利用コンテキストとして取り出す．

実際のコード片の再利用では，3.2 節で述べた再利用を構成する 4 つの操作が順番に行われるとは限らない．例えば，複数の部分からコード片を利用するために，コピーアンドペーストを繰り返した後に，ソースコードを修正するといったことがあり得る．そのため，本手法のように，時間的に空間的に同じ再利用のコンテキストであると思われる情報をあとから分析して取り出す手法が妥当である．

3.4 再利用コンテキスト収集ツールの実装

本研究では，3.1 節で説明した再利用コンテキスト収集手法を実現するためのツールとして **ReuseCollector** を実装している．このツールは統合開発環境の一つである Eclipse のプラグインとして実装されており，Eclipse 上で行われた再利用のコンテキストを半自動的に記録することを可能にする．本ツールは，開発者がエディタ上で行った操作を監視し，特定の操作が行われた際にその内容に合わせた処理を行う．監視する操作は 3.2 章で定義した，再利用を構成する 4 種類の操作である．

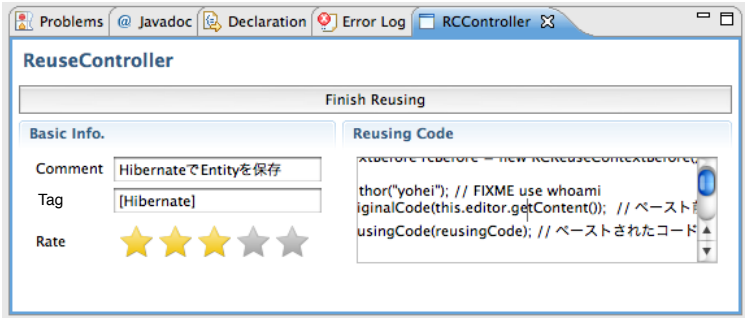

図 3 操作パネル

3.4.1 ツールの構成

本ツールは，以下の 4 つの部分から構成される．

- RCEditor
  コピー操作およびペースト操作を検知し，RCRecorder に通知するエディタ．Eclipse 標準の Java エディタと同等の機能を持つ．
- RCSensor
  エディタのセーブ操作，クローズ操作，非アクティブ化を検知して RCRecorder に通知するモジュール．
- RCRecorder
  RCEditor や RCSensor から受けた通知に従い，Eclipse から必要な情報を収集し，データベースに記録する機能を持つモジュール．本ツールの中心部分．
- RCController
  明示的に再利用の終了を指示したり，再利用の目的を記録するためのインターフェース．再利用の目的として，自然言語による説明・タグ・レーティングを入力できる．図 3 に示す．

開発者は RCEditor を用いてソースコードを編集することで自動的に再利用コンテキストを記録することができる．また，RCController を利用して明示的に再利用を終了させたり，再利用の目的を指定することができる．本ツールの構成を図 4 に示す．

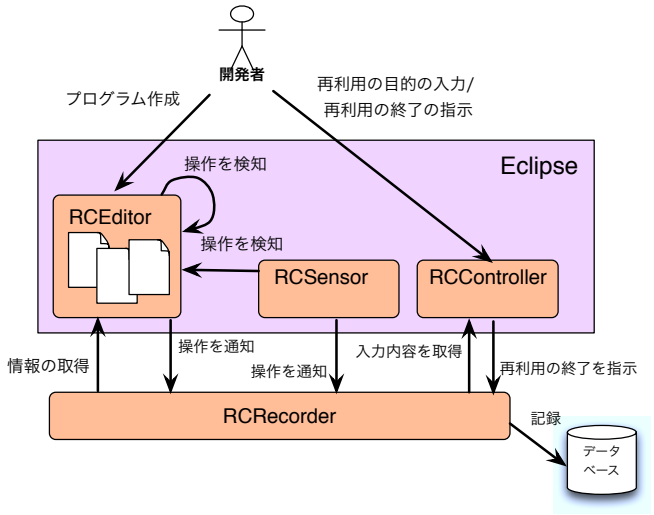

図 4 システム構成図

3.4.2 ツールの動作

本ツールの動作を図 2 で示した再利用を用いて説明する．

本ツールの動作は，開発者が再利用元の選択を行うところから始まる．開発者が再利用元コード上でコピー操作を行うと RCEditor がそれを検知する．RCEditor は再利用元が指定されたことを RCRecorder に通知する．RCRecorder は再利用元コードの情報をエディタから取得しメモリ上に記録しておく．

次に開発者が再利用先コード上でペースト操作を行うとコピー操作の場合と同じように RCEditor がそれを検知する．さらにコピー操作の場合と同じように RCEditor は再利用先が指定されたことを RCRecorder に通知する．RCRecorder はエディタから取得した再利用先コードの情報と先ほどメモリ上に記録しておいた再利用元コードの情報を合わせてデータベースに記録する．

その後，開発者は再利用先コードが動作するようにコードに変更を加える．この操作については本ツールは関与しない．

最後に開発者は再利用の終了を行う．ここでは開発者がセーブを行ったこととする．開発者がセーブを行うと RCEditor がそれを検知する．RCEditor は同様に再利用が終了したことを RCRecorder に通知する．RCRecorder は再利用終了時の再利用先コードの情報と操作パネルに入力されている再利用の目的を RCControler とそのエディタから取得する．

各操作ごとに記録される情報は以下になる．

- 再利用元の選択
  - 再利用元コードの内容
  - 再利用元コードのファイル名
  - 開発者名
  - 記録日時
- 再利用先の選択
  - 再利用先コードの内容
  - 再利用先コードのファイル名
  - 再利用コード片
  - 再利用コード片がペーストされた位置
  - 開発者名
  - 記録日時
- 再利用の終了
  - 再利用先コードの内容
  - 再利用先コードのファイル名
  - 再利用を終了させた実際の操作
  - 操作パネルに入力されている再利用の目的
  - 開発者名
  - 記録日時

3.4.3 開発者操作の検知

本ツールの実装において困難であったのが Eclipse 上での開発者の操作の検知である．本節では，操作の検知を行う RCSensor と RCEditor について詳しく説明する．

まず，RCSensor について説明する．RCSensor はエディタのクローズと非アクティブ化を検知する機能を持つ．RCSensor は拡張ポイントの一つである `org.eclipse.ui.startup` を拡張しており，Eclipse が起動する際に動作する．RCSensor はエディタの生成やアクティブ状態を管理する `IPartService` の持つ機能をフックする．機能をフックするために

は IPartListener インターフェースを実装し，それを IPartService に登録する．エディタのクローズと非アクティブ化をフックするためには，それぞれ partClosed メソッドと partDeactivated メソッドの二つを実装する．

次に，RCEditor について説明する．RCEditor はエディタのセーブ操作，コピーおよびペースト操作を検知する機能を持つ．RCEditor は拡張ポイント org.eclipse.ui.editors を用いた，いわゆるエディタプラグインで CompilationUnitEditor を継承している．基本的に Eclipse に標準搭載されている Java 用のエディタと同様に動作する．セーブ操作とコピーおよびペースト操作の検知の方法は異なっているため，それぞれ説明する．

セーブ操作の検知はシンプルで，エディタのセーブ時に呼び出される doSave メソッドをオーバーライドするのみである．doSave が呼び出されると RCEditor は再利用が終了したことを RCRecorder に伝える．

コピーおよびペースト操作を検知するには，工夫が必要となる．Eclipse 標準の Java 用エディタのコピーおよびペースト操作を実現する機能はエディタ内では実装されておらず，Java エディタの内部に用意されている ClipboardOperationAction クラスに委譲されている．Java エディタのコピーおよびペースト操作を検知するにはこの ClipboardOperationAction クラスのメソッドの呼び出しにフックする必要がある．ClipboardOperationAction クラスのメソッド呼び出しにフックする方法としてまず考えられるのは，ClipboardOperationAction クラスのサブクラスを作成し，フックしたいメソッドをオーバーライドし，RCEditor で利用する方法である．しかし，ClipboardOperationAction は JDT 内部で利用されることを想定しているためか static クラスとして実装されている．static クラスのサブクラスは作成することができない．そのため，本ツールでは，既存のクラスにフックすることをあきらめ，ClipboardOperationAction クラスを複製し，それを RCEditor で利用することとした．複製したクラスでは，run メソッド内のコピーおよびペースト機能の実行の直前に，必要な命令を追加している．この手法は，Eclipse の Java エディタの持つ ClipboardOperationAction クラスと本プラグインの持つ ClipboardOperationAction クラスの間に相違が起こった際に問題が発生する可能性が高く，改善の必要がある．改善の方法としてリフレクションを用いた実装方法などが考えられる．作成した新しいクラスは，CompilationUnitEditor の createActions メソッドをオーバーライドして登録する．

開発者の操作の検知の実装にあたっては，Weckerle らの CPC[3] や，大森らの OperationRecorder[4] の実装を参考にした．

4. プログラム変更支援

本章では，収集した再利用コンテキストを分析することで取得した開発者の知識を活用したプログラム変更支援について説明する．

再利用先が選択された時と再利用が終了した時に取得したコードを比較すれば，再利用がどのように行われたかを取得することができる．開発者が再利用を行うときに，同様の再利用コードに対して，過去にどのような再利用が行われたかを知ることができれば，それを参考にすることで効率よく再利用を行うことができると考えられる．

本手法では，開発者がエディタのペースト操作を行ったときに，自動的にどのように再利用を行えば良いかを提案する．開発者がペーストしたコードをキーにして，あらかじめ蓄積しておいた再利用コンテキストを検索する．そして得られた再利用コンテキストから再利用に伴うソースコードの変更を取り出し, わかりやすく加工して開発者に提示する．開発者は提示されたを再利用コード片の利用方法をもとに再利用を行うことが可能になる．

再利用コンテキストを検索した結果，多くのソースコードの変更情報が得られる可能性がある．そこで本手法では，再利用コンテキストに含まれる，タグ・レーティング・自然言語で記入された目的を利用する．再利用の目的情報によって検索結果をソートしたりフィルタリングしたりすることで，開発者が必要としている再利用に関する再利用方法を取得することが可能となる．

5. おわりに

本論文では，コード片再利用時の開発者の知識を得るために，再利用時のコンテキストを半自動的に収集し，集めた情報から開発者の知識を取得する仕組みの提案と，それを実現するためのツールの開発について述べた． また，取得した再利用コンテキストを用いた，プログラム変更支援ツールの提案を行った．本提案により，再利用コンテキストから導かれた，コード片の再利用時の開発者の知識を利用可能にすることで，ソースコードの特定の部分のみに限った知識を利用することが可能になり，これまでにないプログラム変更支援が可能と考えられる．

今後の課題としては，まず，再利用コンテキスト収集ツールの有効性の検証がある．これについては，オープンソースリポジトリを解析することで，仮想的な再利用コンテキストを収集し，それらに基づいた検証を行うことをを予定している．また，プログラム変更支援手法を洗練し提案・実現することや，再利用コンテキストとして取得すべき要素を再考し，

Vol.2009-SE-166 No.5
2009/11/5

より適切な知識を取得できるように工夫することも課題として挙げられる．

参 考 文 献

1) M.P. Robillard and P.Manggala. Reusing Program Investigation Knowledge for Code Understanding. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 202–211, 2008.
2) Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–11, New York, NY, USA, 2006. ACM.
3) Valentin Weckerle and FreieUniversit at Berlin. CPC: An Eclipse Framework for Automated Clone Life Cycle Tracking and Update Anomaly Detection. 2008.
4) 大森隆行 and 丸山勝久. 開発者による編集操作に基づくソースコード変更抽出. **情報処理学会論文誌**, 49(7):2349–2359, 2008.