一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2013-7 (2013-5)

ソフトウェアリポジトリにおける コードクローン作成者・利用者関係分析手法とその適用

森脇　匠哉† 井垣　宏† 山中　裕樹†
吉田　則裕†† 井上　克郎† 楠本　真二†

† 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 1-5
†† 奈良先端科学技術大学院大学情報科学研究科　〒 630-0192 奈良県生駒市高山町 8916-5
E-mail: †{m-takuya,igaki,y-yuuki,inoue,kusumoto}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp

あらまし　一般に，ソースコードやライブラリの再利用はソフトウェア開発における生産性や信頼性，コスト等の改善に繋がると言われている．一方でソースコードの再利用は，対象となるソースコードを十分に理解し，利用する必要があるため，非常に困難なタスクであると考えられている．そのため，再利用しやすいソースコードの特徴や開発者がどのようなときに再利用を行うかといった既存の再利用動向の分析は，再利用支援において非常に重要である．しかしながら，複数のプロジェクトを対象とした再利用分析において，具体的に誰がどのような再利用を行っているかを定量的に分析した事例は非常に限られているのが現状である．そこで本研究では，コードクローン検出ツールと版管理システムを用い，複数プロジェクトを対象としたコードクローン利用者と作成者の分析手法を提案し，実際に OSS を対象とした分析を行った．その結果，開発者ごとに再利用動向が異なり，分析を進める価値があることを示した．
キーワード　ソフトウェア再利用，コードクローン，版管理システム，Code Authorship

A Case study on How Clone Author and User Interact in Software Repository

Takuya MORIWAKI†, Hiroshi IGAKI†, Yuki YAMANAKA†,
Norihiro YOSHIDA††, Katsuro INOUE†, and Shinji KUSUMOTO†

† Graduate School of Information Science, Osaka University
Yamadaoka 1-5, Suita, Osaka 565-0871 Japan
†† Graduate School of Information Science, Nara Institute of Science and Tecnology
Takayama-cho 8916-5, Ikoma-shi, Nara 630-0192 Japan
E-mail: †{m-takuya,igaki,y-yuuki,inoue,kusumoto}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp

Abstract　In the software industry and OSS projects, it is said that code reuse could improve productivity and reliability of software development, and reduce development time. In many software development organizations, reuse of existing libraries and source code is recommended. Developers in such organizations are required to implement reusable source code and reuse existing reliable source code. On the other hand, source code reuse is especially difficult task because the reuse requires developers to understand the source code fully. In order to encourage such reuse, it is significant to analyze developers' reuse behaviors in the existing projects. However, there exists few analyses about reuse behaviors for each developer. In a software development organization, developers usually belong to multiple projects. An existing research indicated that the situation of the reuse differs by developers, and a developer who recognizes merits of reuse often reuses. Therefore, in this paper, we propose a source code reuse analysis for each developer over two or more projects with using multiple software repositories. As a result, we show that source code reuse analysis in consideration of the difference between each developer is worthy to be conducted.
Key words　Software Reuse, Code Clone, Version Control System, Code Authorship

1. はじめに

ソフトウェアの実装において，保守性や信頼性，生産性といった品質の確保は非常に重要である．そのため，多くのソフトウェア開発現場において，生産性や信頼性の向上を目的とした既存ライブラリやソースコードの再利用が行われている [1]．

組織内での再利用率を高めるためには，開発者による再利用しやすいソースコードの実装と，品質の高いソースコードの再利用を支援する枠組みが重要である．一方で，再利用可能なソースコード部品の開発や既存ソースコードの再利用は困難であることもよく知られている [2]．そのため，再利用支援の一環として，開発者の再利用動向を分析する調査が複数実施されている [3]．鷲崎らはソースファイルやディレクトリ単位のメトリクスと再利用実績を比較し，再利用性の高いソースコードの特徴を示す研究を行っている [4]．しかしながら，開発者個人に着目し，誰が開発したソースコードをどの開発者がどのように利用したかをソフトウェアリポジトリを対象として分析した事例はほとんど存在しない．実際に Manuel Sojer らは数百人の OSS の開発者にアンケートを実施し，既存ソースコードの再利用傾向が開発者ごとに異なる可能性があることを示している [5]．

そこで我々はソフトウェアリポジトリを対象とした開発者個人レベルでの再利用関係を定量的に分析する手法を提案する．ソフトウェアリポジトリを対象とした再利用分析では，コードクローン検出ツール [6] [7] [8] が利用されることが多い [3]．ここでコードクローンとは，ソースコード中で互いに類似または一致した部分を持つコード片のことである [9]．さらに，互いに類似するコードクローンの集合のことをクローンセットと呼ぶ．

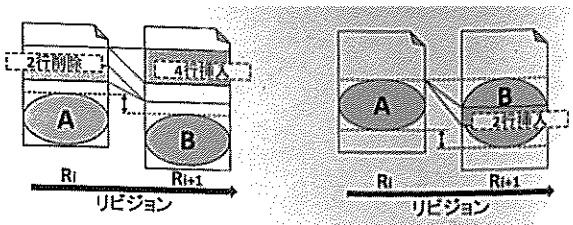
ソースコードの再利用を行う場合，再利用元と先のソースコードはコードクローンとなることが多い．そこで，本研究では版管理システム (Version Control System) の保持するソフトウェアリポジトリを対象としてコードクローン検出ツールを用い，複数プロジェクトにまたがった開発者ごとのソースコード再利用傾向についての調査を行う．再利用傾向の分析においては，再利用されたソースコードのオリジナルが誰によって開発されたのかをリポジトリ分析によって明らかにすることで，再利用されたソースコードの開発者（コードクローン作成者）と再利用を行った開発者（コードクローン利用者）の関係をより正確に分析することが可能となると考えている．

分析に際して設定した Research Question は以下に示すとおりである．

2. 背　　景

2.1 ソフトウェアの再利用

ソフトウェアの再利用とは，ソフトウェア開発の分析，設計，実装の各工程において，既存のソフトウェアで起きた問題とその問題に対する解法などの知識を，開発中のソフトウェアに対して適用することである [10]．既存のソフトウェアの知識を活用することで，新たなソフトウェア開発が容易となる．一般に，ソフトウェアの再利用は開発時間の短縮や開発コストの削減，



(a) ケース 1　　(b) ケース 2

図 1　2 リビジョン間のコードクローンの対応関係

さらにはグループ開発での生産性，信頼性の向上改善など，ソフトウェアの品質向上に繋がると言われている [1]．そのため，多くのソフトウェア開発で既存のライブラリやソースコードの再利用が行われている．

再利用支援を目的とした再利用傾向の分析は数多く行われている．Sojer らは数百人の OSS の開発者にアンケートを実施し，既存ソースコードの再利用傾向が開発者毎やプロジェクトの性質，プロジェクトのフェーズによって異なることを示した [5]．この研究では再利用のメリットをよく認識しており，多くのプロジェクトに関与している開発者ほど再利用を積極的に行う傾向にあるということがアンケートによって確認されている．結果として，開発者単位での再利用傾向について定量的で客観的な分析が重要であることが示されている．

定量的な再利用傾向の分析としては，鷲崎らがソースファイルやディレクトリ単位の再利用メトリクス候補を 102 種提案し，実際の再利用実績と比較し，再利用性の高いソースコードの特徴を抽出する研究を行っている [4]．また Prakriti Trivedi らは，ソフトウェアの再利用性についてのメトリクス計測手法を提案している [1]．この研究では，ソフトウェアコンポーネントについて結合度や凝集度についてのメトリクスを求めることで，そのソフトウェアコンポーネントを再利用した場合にソフトウェアの信頼性にどの程度影響を与えるかを算出している．Balint らはあるプロジェクトの特定のリビジョンを対象としてコードクローンを検出し，検出されたコードクローンに開発者情報を付与することで，コードクローンが誰にいつ編集されたかを可視化し，分析を行っている [11]．

このように再利用傾向についての分析は数多く行われているが，特定のコミュニティにおける複数のプロジェクトを対象として，版管理システムが保持するソフトウェアリポジトリを対象として再利用傾向がどのように変遷しているかを分析した研究は著者らの知る限り存在しない．

我々は次節で詳述する 2 バージョン間でのコードクローンの遷移情報分析手法 [12] を利用し，複数プロジェクト間の複数リビジョンを保持するリポジトリを対象として，再利用傾向を分析する手法を提案する．

2.2 コードクローン遷移分析

コードクローンの遷移情報を分析するためには，連続する 2 リビジョン R_i，R_{i+1} に含まれるコードクローンの対応関係を求める必要がある．ここでは，リビジョン R_i に存在する全て

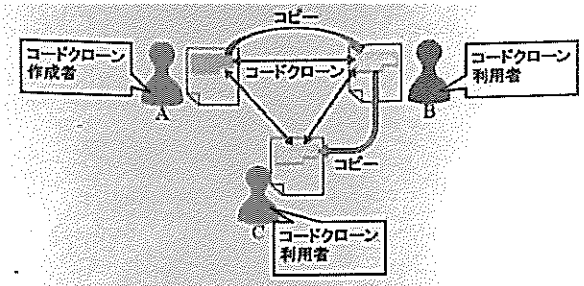


図 2　コードクローン利用者と作成者

のコードクローンの集合を C_i，リビジョン R_{i+1} に存在する全てのコードクローンの集合を C_{i+1} と表す．

1 は 2 リビジョン間のコードクローンの対応関係の例を示している．図 1(a) では，コードクローン $A \in C_i$ の前で 4 行の挿入と 2 行の削除が行われているため，A に対応するコード片 B の開始行番号と終了行番号は，それぞれ A の開始行番号と終了行番号に 2 行追加した値となる．B が C_{i+1} に含まれる場合，コードクローン A に対応するコードクローンは B となる．また，図 1(b) では，コードクローン A の前で編集操作は行われていないため，対応するコード片 B の開始行番号は A の開始行番号と同じになる．一方，A に 2 行の挿入が行われているために，B の終了行番号は A の終了行番号に 2 行追加した値となる．B が C_{i+1} に含まれる場合，コードクローン A に対応するコードクローンは B となる．このように，あるコードクローン $A \in C_i$ に対応するコード片をその開始行番号と終了行番号の対応に基づいて求め，そのコード片 B がコードクローンとなっている場合，コードクローン A とコードクローン B の間に履歴関係があると定義する．2 リビジョン間のコードクローンの履歴関係を分析することで，R_{i+1} に存在するクローンが新たに追加されたクローン (Added Clone) なのか，変更されたクローン (Modified Clone) なのかといった分類を行うことが可能となる．さらに，コードクローンの分類を利用することで，そのコードクローンを含むクローンセットに対しても，新たに発生したのか (New Clone Set)，変更があったのか (Changed Clone Set) といった分類が可能となる．

本稿では，このような 2 バージョン間のコードクローン遷移情報にもとづき，3 バージョン以上にまたがるコードクローン遷移情報を導出し，開発者ごとの再利用傾向を分析する手法を提案する．

2.3　再利用分析におけるコードクローン作成者とコードクローン利用者

通常，再利用はある開発者が実装したソースコードを自分自身もしくは他の開発者がコピーすることで行われる．本研究では，再利用元のコード片を一番最初に実装した開発者のことをコードクローン作成者，作成者が実装したソースコードを直接・間接的にコピーして利用した開発者のことをコードクローン利用者と呼ぶ．作成者と利用者の関係を図 2 に示す．我々はソフトウェアリポジトリとバージョン間のコードクローン遷移情報を利用することで，対象となるソフトウェアシステムのライフサイクルを通じた再利用傾向分析の実現を目指す．

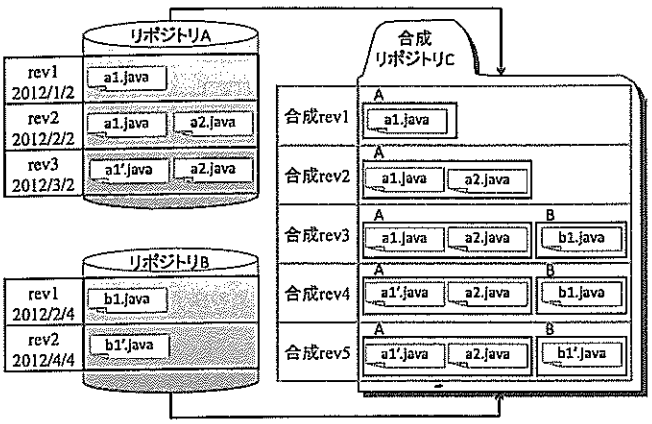


図 3　複数リポジトリからのチェックアウト

3.　提案手法

3.1　概　　要

システムの手順を以下に示す．

入力

複数プロジェクトのリポジトリ

STEP1：複数リポジトリからのチェックアウト

複数のリポジトリからリビジョンの時系列順にチェックアウトを行い，合成リポジトリを生成する．

STEP2：隣接リビジョン間でのクローン遷移情報の導出

合成リポジトリの隣接するリビジョン間それぞれについてコードクローンを検出し，2 リビジョン間のクローン遷移情報を導出する．

STEP3：クローンセット履歴の抽出

2 リビジョン間のクローン遷移情報を組み合わせてクローンセット履歴として抽出する．

STEP4：コードクローン作成者と利用者の特定

クローンセット履歴に基づいてクローンセットごとのコードクローン作成者及び利用者の特定を行う．

出力

クローンセットごとのコードクローン作成者，発生リビジョン，コードクローン利用者及びコード片に関する各種情報

3.2　STEP1：複数リポジトリからのチェックアウト

本研究では，プロジェクト間コードクローンを検知するために，複数のリポジトリからファイルのチェックアウトを行う．

3.2.1　チェックアウト手法

例として，リポジトリ A，リポジトリ B からファイルをチェックアウトする様子を図 3 に示す．最初に，2 つのリポジトリ A，B の内，最もコミット日時の古いリポジトリ A の *rev1* のディレクトリ内容をチェックアウトする．この時，各リポジトリに同名のファイルが存在する場合の競合を防ぐため，ディレクトリ A を作成しその中にチェックアウトを行う，以降，時系列順にファイルをチェックアウトしていくことで，ある時点での各リポジトリのファイル内容を保持するディレクトリが得られる．このディレクトリに対して分析を行うことで複数リポジトリ間のコードクローンの分析が可能である．

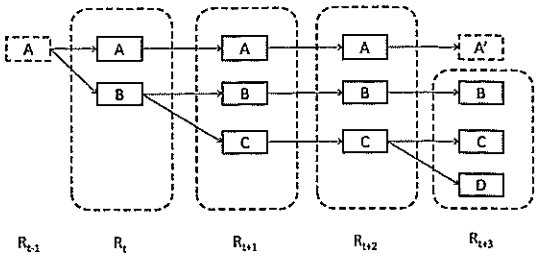


図 4　クローンセット変更の様子

このStepによって，複数プロジェクトのある時点における各リポジトリのファイル内容を保持するディレクトリが得られる．ここでは各ディレクトリのことを複数プロジェクトの合成リビジョン，合成リビジョンの集合のことを合成リポジトリと呼ぶ．図3の例では，リポジトリ A，B の合成リポジトリ C を作成したことになる．

3.3　STEP2：隣接リビジョン間でのクローン遷移情報の導出

Step1で得られた合成リポジトリを対象として2リビジョン間でのクローン遷移情報を導出する．

隣接する合成リビジョンを対象として，2.2節で説明した山中らの手法を用い，クローンセットの遷移情報を求める．図3の例では，合成リビジョン1と合成リビジョン2間でのクローンセットの遷移情報，合成リビジョン2と合成リビジョン3間でのクローンセットの遷移情報,,,，合成リビジョン4と合成リビジョン5間でのクローンセットの遷移情報を求める．

3.4　STEP3：クローンセット履歴の抽出

ここでは2リビジョン間でのクローン遷移を全合成リビジョン間でのコードクローンの遷移情報に拡張したものをクローンセット履歴と呼ぶ．

隣接する合成リビジョン R_t，R_{t+1}，R_{t+2} があったときに，R_t と R_{t+1}，R_{t+1} と R_{t+2} 間でそれぞれクローンセット遷移情報が算出されていた場合，クローンセットごとに遷移情報を結合することで $R_t \sim R_{t+2}$ までのクローンセット履歴を得ることができる．なお，ここでは隣接するリビジョン間で変化が無いクローンセット及び新たに発生したクローンセットを対象として結合処理を行った．全隣接リビジョン間において遷移情報の結合を行うことで，全合成リビジョンを通したクローンセット履歴の取得が可能となる．

クローンセット履歴は，クローンセットごとに追加されたコードクローンの情報 (リビジョン番号，パス，各行の Code Authorship) を持っている．ここで Code Authorship [13] とはバージョン管理システムに保存されている行単位の開発者情報を指すものとする．なお，パスはファイルパスとコードクローンの開始行，終了行のことを指す．例えば，図4に示すようにクローンセットが遷移していった場合，クローンセット履歴として，R_t におけるコードクローンAとコードクローンBの情報，R_{t+1} におけるコードクローンCの情報，R_{t+3} におけるコードクローンDの情報が保持される．

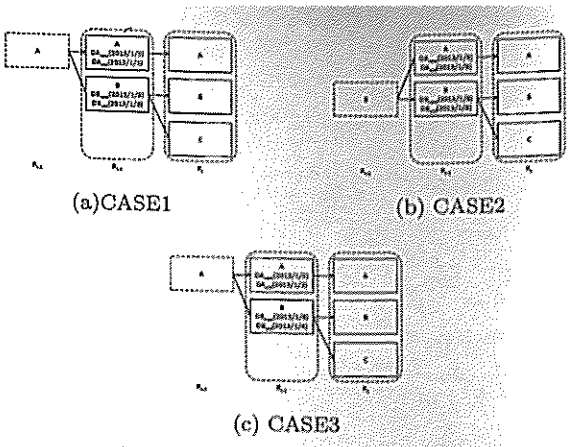


図 5　コードクローン作成者導出

3.5　STEP4：コードクローン作成者と利用者の特定

3.5.1　コードクローン作成者と利用者の分析

ある合成リビジョンで新たに発生したコードクローンについて，そのコードクローンの Code Authorship からコードクローン利用者を導出する．そして，クローンセット履歴の分析を行うことで，ある合成リビジョンで発生したコードクローンが元のリポジトリのどのリビジョンのコード片が元になっているかを特定する．元コード片の各行の Code Authorship を求めることでコードクローン作成者を導出する．

3.5.2　導出方法

導出方法を以下に示す．以下，R_t を分析対象の合成リビジョン，C_t を R_t に含まれる全コードクローンの集合とする．

ある合成リビジョンでコードクローン C_t が発生したとする．そのコードクローンCの Code Authorship である開発者がコードクローン利用者である．最初に，クローンセット内に親クローンを持つコードクローンがなくなるまで合成リビジョンをさかのぼる．次に，クローンセット内のコードクローンの各行について，版管理システムを基に Code Authorship とコミット日時を求める．

ここで，コードクローン A とコードクローン B が属するクローンセットにコードクローン C が追加されたと仮定して説明を行う．コードクローン A 内で最も古いコミット日時の行のコミット日時を DA_{old} と定義し，最も新しいコミット日時を DA_{new} とする．同様に，コードクローン B 内で最も古いコミット日時の行のコミット日時を DB_{old}，最も新しいコミット日時を DB_{new} とする．

CASE1

図5 (a) に示すように，DA_{new} が DB_{old} より古い場合，コードクローン C の作成者はコードクローン A の Code Authorship である開発者である．

CASE2

図5 (b) に示すように，DB_{new} が DA_{old} より古い場合，コードクローン C の作成者はコードクローン B の Code Authorship である開発者である．

CASE3

図5 (c) に示すように，CASE1，CASE2 に当てはまらない場合，DA_{old} と DB_{old} を比較し，より古いコードクローンの

Code Authorship である開発者がコードクローン作成者である．この例では，コードクローン A の Code Authorship である開発者がコードクローンの作成者である．

結果として，クローンセットごとにクローンセット履歴をまとめたデータ構造が得られる．さらにコードクローン作成者情報 (作成したコードクローンの情報) と利用者の関係が得られる．

4. ケーススタディ

開発したシステムを用いて，リポジトリのコードクローン作成者と利用者に着目した再利用分析を行った．本節では，2 つの Java プロジェクトに対して提案手法を適用した再利用分析に関するケーススタディについて詳述する．

4.1 準　　備

対象として，OSS の Java プロジェクト eclipse.platform.text と eclipse.pde を入力として用意した．eclipse.platform.text(2001/5/2 ～ 2003/12/22) は 1369 リビジョンで開発者 19 名の規模であり．eclipse.pde(2002/4/11 ～ 2003/12/18) は 144 リビジョンで開発者 3 名の規模である．また，両プロジェクトにまたがって開発を行っている開発者が 2 名いた．

提案手法に従い，クローンセット履歴分析を行い，コードクローン利用者と作成者を導出した．そして，導出されたコードクローン作成者と利用者の情報を基に分析を行った．

本研究では，複数プロジェクトを対象としたコードクローン作成者と利用者の分析を行うことで，どの程度開発者ごとに再利用傾向が異なるのかを明らかにすることを目的し，以下の 5 つの分析を行った．

（1） 再利用回数の多いクローンセットとコードクローン利用者数が多いクローンセットが一致するか

（2） コードクローン利用者数の多いコードクローンと，再利用回数は多いがコードクローン利用者数の少ないコードクローンに違いがあるか

（3） コードクローンの作成数と利用数の多い開発者にどのような特徴があるか

（4） コミット数の多い開発者はコードクローンの作成数と利用数が多いかどうか

分析内容 1 では，再利用回数の多いコード片では，ユニークな利用者数も比例して多いのかを分析する．この分析では，既存研究で言われるような再利用傾向の差異が開発者間に実際に存在するかを明らかにすることを目的としている．分析内容 2 では，自分で自分の書いたコードを再利用する場合と，他人のコードを再利用する場合での対象となるコード片の違いを知ることを目的としている，分析内容 3 では，数多くの再利用に関わっている開発者について共通の特徴があるかを知ることを目的としている．分析内容 4 では，分析 1 とは異なったコミットという観点において，開発者間における再利用傾向の差異を知ることを目的としている．

4.2 結　　果

本ケーススタディではクローンセットが 969 個検出された．また，複数の合成リビジョンにまたがって再利用が行われていたクローンセットが 156 個抽出された．その内，5 つのクローンセットはプロジェクト間での再利用が行われていた．本ケーススタディでは，合成リビジョンにまたがって再利用が行われていた 156 個のクローンセットを対象として分析を行った．

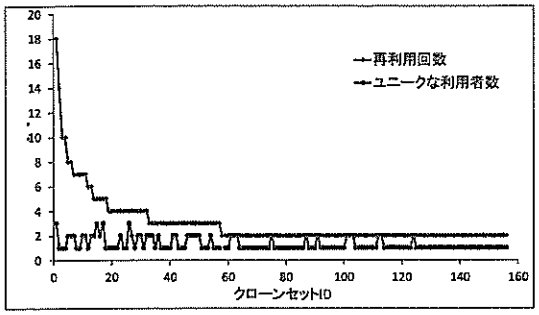


図 6　再利用回数の多いクローンセットとユニーク利用者数の多いクローンセットの関係

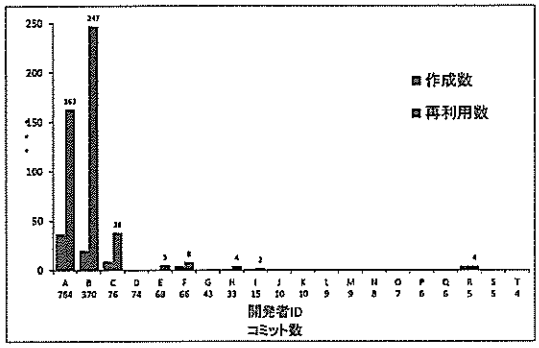


図 7　開発者ごとのコードクローン作成数と再利用数

以下に，分析結果を示す．

分析内容 1

図 6 に再利用回数の多いクローンセットとユニーク利用者数の多いクローンセットについて示す．X 軸には再利用の多い順にクローンセットを並べている．Y 軸はそのクローンセットの再利用回数とユニーク利用者数を示す．ここでユニーク利用者数とはクローンセットの利用者である開発者が何名であるかを示す．

分析内容 2

ユニークユーザの多いコードクローンとユーザ数に関係なく再利用回数の多いコードクローンについて分析を行った．ユニークユーザの多いコードクローンについて，検出されたコードクローンを調査し，ユーザ・インターフェース (以下，UI と示す) に関する操作を行うソースコードが再利用されていることが分かった．また，再利用回数の多いコードクローンについて，検出されたコードクローンを調査し，OS ごとに対応した処理を行うソースコードが再利用されていることが分かった．

分析内容 3

コードクローン作成数と利用数の多かった開発者について，205 個の Eclipse プロジェクト(注1) のうち，どの程度のプロジェクトに関わっているかを調査した．作成数が 37 回で一番多く，再

(注1) : http://dash.eclipse.org/dash/commits/web-app/active-committers.cgi

利用数が 163 回で二番目に多い開発者 A は，48 個のプロジェクトに関わっていた．作成数が 20 回で二番目に多く，再利用数が 247 回で一番多い開発者 B は，19 個のプロジェクトに関わっていた．作成数が 9 回で三番目に多く，再利用数が 38 回で三番目に多い開発者 C は，7 個のプロジェクトに関わっていた．

分析内容 4

図 7 に開発者ごとのコードクローン作成数と再利用数を示す．X 軸は開発者とコミット数を示し，コミット数が多い順に並べている．Y 軸はコードクローン作成数及び再利用数を示す．

4.3 考　　察

分析結果より得られた考察を以下に示す．分析内容 1 について，再利用回数の多いクローンセットとコードクローン利用者数が多いクローンセットが一致しなかった．この結果より，再利用回数が多ければユニークユーザ数が多いというわけではないことが分かる．分析内容 2 について，再利用回数とユニーク利用者数がともに多かったコードクローンの例を図 9 に，ユニーク利用者数の多いコードクローンの例を図 8 に，ユニーク利用者数が少なく，再利用回数が多いコードクローンの例を図 10 に示す．図 10 に示すコード片は再利用が 16 回でユニーク利用者数は 3 名である．また，図 8 に示すコード片は再利用が 4 回でユニーク利用者数は 3 名である．そして，図 10 に示すコード片は再利用が 6 回でユニーク利用者数は 1 名である．今回の実験のみでは必ずしも判断できないが，本事例に関してはユニーク利用者数が多いコードのほうが単純で理解しやすいものであると判断できた．一方，ユニーク利用者数が少ないコード片については，汎用性は高いが，ロジックが利用者数が多いものと比較して複雑になっているものが多くあった．分析内容 3 について，コードクローンの作成数が多い開発者は比較的多くのプロジェクトに関わっていることが分かった．コードクローン作成者の記述した元コード片が他のプロジェクトから再利用してきたものであることも考えられる．そして，分析内容 4 について．単純にコミット数が多ければコードクローン作成数，利用数が多いわけではないことから，開発者ごとの再利用傾向には特徴があることが分かる．

本研究では版管理システムとコードクローン検出システムを用いて，複数のプロジェクトにおけるコードクローンの遷移情報からコードクローンの作成者と利用者を導出することにより，開発者ごとの再利用傾向を分析する手法を提案した．

今後，本分析手法を用いた再利用支援の枠組みを構築することを計画している．具体的には，再利用を行おうと考えているユーザにコードクローン作成者の情報を提示する仕組みやユニークユーザ数の多いコード片から再利用性の高いコード片の特徴抽出といった仕組みを構築し，より再利用しやすい開発環境の構築を目指して行きたい．

謝辞　本研究は，日本学術振興会科学研究費補助金若手研究 (B)(課題番号：24700030) の助成を得た．

```
setAction(ITextEditorActionConstants.GOTO_LINE, action);
markAsContentDependentAction(ITextEditorActionConstants.UNDO, true);
markAsContentDependentAction(ITextEditorActionConstants.REDO, true);
markAsContentDependentAction(ITextEditorActionConstants.FIND, true);
markAsContentDependentAction(ITextEditorActionConstants.FIND_NEXT, true);
markAsContentDependentAction(ITextEditorActionConstants.FIND_PREVIOUS, true);
markAsContentDependentAction(ITextEditorActionConstants.FIND_INCREMENTAL, true);
```

図 8　ユニーク利用者数の多いコードクローンの例

```
gestureMap.put("E", "org.eclipse.ui.navigate.forwardHistory");
gestureMap.put("N", "org.eclipse.ui.file.save");
gestureMap.put("NW", "org.eclipse.ui.file.saveAll");
gestureMap.put("S", "org.eclipse.ui.file.close");
gestureMap.put("SW", "org.eclipse.ui.file.closeAll");
gestureMap.put("W", "org.eclipse.ui.navigate.backwardHistory");
gestureMap.put("EN", "org.eclipse.ui.edit.copy");
```

図 9　ユニーク利用者数，再利用数ともに多いコードクローンの例

```
if (store.contains(LINE_NUMBER_COLOR)) {
  if (store.isDefault(LINE_NUMBER_COLOR))
    rgb= PreferenceConverter.getDefaultColor(store, LINE_NUMBER_COLOR);
  else
    rgb= PreferenceConverter.getColor(store, LINE_NUMBER_COLOR);
  }
```

図 10　ユニーク利用者数が少なく，再利用回数が多いコードクローンの例

文　　献

[1] Trivedi Prakriti and Kumar Rajeev. Software Metrics to Estimate Software Quality using Software Component Reusability. *IJCSI International Journal of Computer Science Issues*, Vol. 9, pp. 144–149, 2012.

[2] Will Tracz. Confessions of a used-program salesman: Lessons learned. In *Proceedings of the 1995 Symposium on Software Reusability*, SSR '95, pp. 11–13, New York, NY, USA, 1995. ACM.

[3] Lars Heinemann, Florian Deissenboeck, Mario Gleirscher, Benjamin Hummel, and Maximilian Irlbeck. On the Extent and Nature of Software Reuse in Open Source Java Projects. In *Proceedings of the 12th International Conference on Top Productivity Through Software Reuse*, ICSR'11, pp. 207–222, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] 鷲崎弘宜, 森田翔, 長井悠兵, 布谷貞夫, 佐藤雅宏, 杉村俊輔, 関洋平. 組込みソフトウェアの派生開発におけるソースコードメトリクスによる再利用性測定. ソフトウェア品質シンポジウム 2012, 2012.

[5] Manuel Sojer and Joachim Henkel. Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. *Journal of the Association for Information Systems*, Vol. 11, No. 12, 2010.

[6] Lingxiao Jiang, Ghassan Misherghi, and Zhendong Su. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In *Proceedings of International Conference on Software Engineering*, pp. 96–105, 2007.

[7] CCFinderX. `http://www.ccfinder.net/`.

[8] Simian. `http://www.harukizaemon.com/simian/`.

[9] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[10] Johannes Sametinger. *Software Engineering with Reusable Components*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[11] Mihai Balint, Tudor Girba, and Radu Marinescu. How developers copy. In *Proceedings of International Conference on Program Comprehension 2006*, pp. 56–65, 2006.

[12] 山中裕樹, 崔恩瀞, 吉田則裕, 井上克郎, 佐野建樹. コードクローン変更管理システムの開発と実プロジェクトへの適用. ソフトウェアエンジニアリングシンポジウム 2012 論文集, 第 2012 巻, pp. 1–8, aug 2012.

[13] Jan Harder. Code Clone Authorship — A First Look. *STT*, Vol. 32, No. 2, pp. 25–26, 2012.