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.

複数のプロジェクトを対象としたクローンの系譜にもとづくソースコード再利用分析手法の提案

森脇　匠哉† 　堀田　圭佑† 　井垣　　宏† 　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
E-mail: †{m-takuya,k-hotta,igaki,inoue,kusumoto}@ist.osaka-u.ac.jp

あらまし　一般に，ソースコード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ライセンス違反やバグの伝搬を引き起こすこともあり，非常に困難なタスクである．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非常に重要である．しかしながら，様々なソースコード再利用分析が行われている一方で，組織内の再利用動向を対象とした再利用分析は，複数プロジェクトにまたがって分析を行うという課題もあり，あまり行われていないのが現状である．そこで本研究では，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動向を分析する手法を提案する．提案手法では，クローンの系譜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クローンの系譜の導出し，その系譜に基づいて再利用の特定を行う．実際に，OSS を対象に手法の適用を実施し，開発者ごとに再利用動向が異なることを示した．また，プロジェクト間での再利用や再利用される回数の多いソースコードを特定し，その特徴を示した．

キーワード　ソースコード再利用, 版管理システム, コードクローン, クローンの系譜, Code Authorship,

Source Code Reuse Analysis in Multiple Projects based on the Clone Genealogy

Takuya MORIWAKI†, Keisuke HOTTA†, Hiroshi IGAKI†, Katsuro INOUE†, and Shinji KUSUMOTO†

† Graduate School of Information Science, Osaka University
Yamadaoka 1-5, Suita, Osaka 565-0871 Japan
E-mail: †{m-takuya,k-hotta,igaki,inoue,kusumoto}@ist.osaka-u.ac.jp

Abstract　In the software industry and OSS projects, it is said that source code reuse could improve productivity and reliability of software development, and reduce development time. On the other hand, source code reuse requires professional skills to developers. Ad-hoc reuse might introduce some maintenance problems. The source code reuse analysis for software development organizations is worthy to be conducted. However, there exists few analyses about reuse behaviors in organization. In a software development organization, developers usually belong to multiple projects. Therefore, in this paper, we propose a source code reuse analysis utilizing a clone genealogy detection tool for each developer over two or more projects with using multiple software repositories. We conducted a case study to investigate source code reuse behaviors among 5 software repositories.

Key words　Source Code Reuse, Version Control System, Code Clone, Clone Genealogy, Code Authorship

1. ま え が き

ソフトウェアの実装において，生産性や信頼性の改善を目的として，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 [1]．そのため，開発者には，再利用しやすいソースコードを書くことと品質の高いソースコードを再利用することが求められている．一方で，再利用可能な部品の開発や既存の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]．そして，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とされて

いる．また，特定の組織における再利用の分析を行うことで，ライセンス違反の特定やよく再利用されるソースコードの外部ライブラリ化といった提案も可能となるとされている [3].

そこで，ソースコードの再利用についての理解を深めることを目的として，Open Source Software(以下，OSS と示す) や企業内のプロジェクトを対象としたソースコードの再利用実績の分析が行われるようになってきた [4,5].

ソースコードの再利用動向を特定する手法の一つとしてコードクローン検出技術を用いたものが提案されている [6,7]．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 [8]．また，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．コピーアンドペーストによる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．

本研究では版管理システム (Version Control System) とコードクローン検出技術を用いて，複数プロジェクトにまたがる，開発者ごとのソースコードの再利用傾向についての調査を行う．

提案手法では，最初に複数プロジェクトのリポジトリに対して，開発日時の情報が古い順にディレクトリ構造を取得していくことで 1 つのリポジトリへのマージを行う．

以降，2 節では本研究の背景を説明する．3 節では提案手法について説明する．4 節では本研究で行った実装について説明し，5 節ではケーススタディについて述べる．6 節ではケーススタディに対する考察を述べる．7 節ではむすびとして，まとめと今後の課題について述べる．

2. 準　　備

ソフトウェアの再利用および既存の再利用分析手法について説明する．

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．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，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において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．しかし，不用意なソフトウェアの再利用はライセンス違反やバグの伝搬といった問題を発生させる危険もある [3]．結果として再利用手法の改善や問題の特定といった目的で，ソフトウェアの再利用動向の分析が重要とされている．

ソースコードを作成している個人や組織についての再利用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 [4]．この研究において，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た．また，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．

鷲崎ら [5] は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ている．そして，比較によって得られた知見を基に，再利用性の高いソースコードの特徴を具体的に示している．

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概算している．つまり，メトリクスによってソフトウェアコンポーネントが再利用に適しているかどうかを知ることが可能となる．

このように様々なソースコード再利用分析が行われている一方で，組織内の再利用動向を対象とした再利用分析はあまり考慮されていない．通常，ソフトウェア開発を行う組織では複数のソフトウェア開発プロジェクトが行われている．そのため，特定プロジェクト内だけでなく，プロジェクトにまたがった再利用を考慮する必要がある．さらに，組織内での再利用分析においては，再利用手法の改善や問題の特定のために，誰が開発したソースコードをいつ，誰が再利用したかを特定することが重要である．すなわち，再利用分析において関連する開発者を特定することで，開発者自身に改善や問題の解決を直接促すことが可能となる．

そこで本研究では，コードクローン技術を利用し，複数プロジェクトにまたがったソースコード再利用を開発者ごとに特定する手法を提案する．

2.2 ソースコード再利用分析のためのコードクローン技術

Harder らはコードクローン技術を用いた再利用動向の特定手法を提案している [9]．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．Harder らはあるプロジェクトの数千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の開発者情報からコードクローンの Authorship を特定する研究を行っている．

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．ここで，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再利用元開発者，再利用元開発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再利用者と呼ぶ．再利用元開発者と再利用者の関係を図 1 に示す．

本研究では，堀田らのクローンの系譜検出手法を参考にしている [10–12]．クローンの系譜とは，コードクローンの変更の履歴のことである．クローンの系譜から，コードクローンがいつ発生したか，またいつ新たなコード片追加されたか，といった情報を得ることが出来る．堀田らは，CRD(Clone Region Descriptors) [13] というコードクローンを構成するコード片の

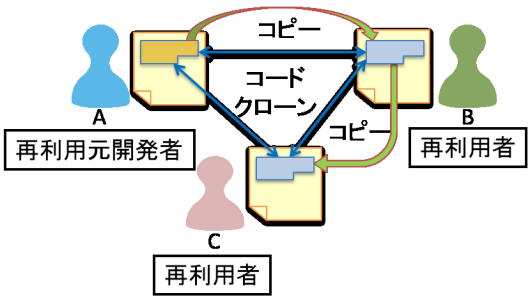

図 1　再利用元開発者と再利用者

位置情報を用いたコードクローンの追跡手法に改良を加え，より正確な追跡を実現した手法を提案している．コードクローンの追跡とは，あるリビジョンのソースコード中に存在するコードクローンについて，次リビジョンのソースコードのどこに存在するかを特定し，その対応付けを行う技術である，

クローンの系譜からあるリビジョンにおけるコードクローンの発生を特定し，それが以前のリビジョンに存在するクローンセットに追加されたものである時，ソースコードの再利用が行われたとみなすことができる．

以降，堀田らの手法について説明を行う．

a）　STEP1：ブロックの特定

各リビジョンにおけるソースファイルに対し字句解析及び構文解析を行うことでブロックの特定を行う．また，それぞれのブロックについての CRD の算出も行う．

b）　STEP2：コードクローンの検出

各リビジョンにおいて，ブロック単位でのコードクローンの検出を行う．検出の単位をクローンとすることで，行やトークンなどの細かい粒度で検出を行う手法に比べて，高速なコードクローンの検出を行うことが可能となる．ブロックの比較は，正規化したブロックの文字列から算出したハッシュ値を比較することで行う．

c）　STEP3：コードクローンの追跡

CRD を用いてコード片の対応付けを行うことで，コードクローンの追跡を行う．CRD の類似度の算出にはレーベンシュタイン距離 [14] を用いる．レーベンシュタイン距離とは，一方の文字列をもう一方の文字列に変換する際に，どれだけの編集回数が必要であるかを数値化したものである．この値が小さいほど，文字列間の類似度が高いことを示す．

次節では，これらのコードクローン技術を応用し，開発者ごとの再利用動向を分析する手法について詳述する．

3.　提 案 手 法

本研究では，複数プロジェクトから検出されたクローンセットの系譜に基づいて，クローンセットが新たに発生した，あるいは系譜をもつ既存のクローンセットにそのクローンセットとクローンの関係にあるコード片が追加された際に，再利用が行われたものと判断する．

3.1　概　　要

我々が提案する再利用動向検出の手順について概要を以下に示す．

入力

複数プロジェクトのリポジトリをシステムへの入力として与える．

STEP1：リポジトリのマージ

複数のリポジトリを一つのリポジトリにマージし，合成リポジトリを作成する．

STEP2：クローンの系譜の導出

マージしたリポジトリにおけるクローンの系譜を導出する．

STEP3：再利用元開発者と再利用者の特定

クローンの系譜に基づいて，ソースコードの再利用を特定する．

出力

再利用についての情報がデータベースに出力される．

3.2　STEP1：リポジトリのマージ

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，複数のリポジトリを一つのリポジトリにマージする．

3.2.1　入　　力

複数プロジェクトのリポジトリを入力として与える．

3.2.2　マージ手法

例として，リポジトリ A，リポジトリ B をマージし，合成リポジトリを作成する様子を図 2 に示す．リポジトリ A はリビジョン A1 及び A2 を持ち，リポジトリ B はリビジョン B1 及び B2 を持つ．全リビジョンの内，最もコミット日時の古いリビジョンである A1 が合成リポジトリの初期リビジョンとなる．以降，合成リポジトリのリビジョンのことを合成リビジョンと呼ぶ．そして，時系列順にチェックアウトしていくことで合成リポジトリの作成を行う．つまり，合成リビジョン 2 はリビジョン A1 及びリビジョン B1 のディレクトリ構造を持つ．また，合成リビジョン 3 はリビジョン A2 及びリビジョン B1 のディレクトリ構造を持つ．そして，合成リビジョン 4 はリビジョン A2 及びリビジョン B2 のディレクトリ構造を持つ．

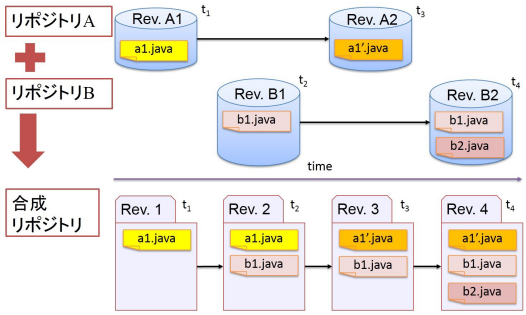

図 2　リポジトリのマージ手法

3.2.3　出　　力

複数プロジェクトのリポジトリをマージした合成リポジトリが出力として得られる．

3.3　STEP2：クローンの系譜の導出

全リビジョンにおけるクローンの系譜を導出する．

3.3.1 入　　力

入力として，節 3.2 で得たリポジトリを与える．

3.3.2 クローンの系譜導出手法

節**??**で説明した堀田らの手法を用いてクローンの系譜を求める．

3.3.3 出　　力

合成リポジトリから検出されたクローンの系譜の情報が出力される．

3.4 STEP3：再利用元開発者と再利用者の特定

再利用元開発者及び再利用者の導出において Code Authorship を用いる．Code Authorship とは，ソースコードの各行を誰が書いたかを示す情報である．本研究ではコード片の行ごとの Code Authorship を求め，過半数を占める開発者をそのコード片の Authorship と定義する．

STEP2 までで導出されたクローンの系譜について，コード片ごとの Authorship を求めることで，系譜ごとに再利用元開発者及び再利用者を特定する．

3.4.1 導 出 方 法

導出方法を以下に示す．

まず，各系譜について，コードクローン発生したリビジョンを特定する．発生リビジョンにおけるコードクローンを構成するコード片の集合の内，最も実装日時が古いコード片の Authorship を再利用元開発者として特定する．そして，それ以外のコード片の Authorship より特定された開発者を再利用者とする．さらにクローンの系譜を時系列順に辿って行き，あるリビジョンにおいてクローンセットに新たなコード片が追加された際，そのコード片の Authorship を用いて再利用者を特定する．

3.4.2 出　　力

全クローンの系譜における再利用元開発者及び再利用者の情報がデータベースに出力される．

3.5 並 列 処 理

クローンの系譜の導出において，リビジョン単位で処理を分割し，最後にまとめてマージすることで処理の高速化を実現した．また，再利用元開発者と再利用者の特定において，クローンの系譜ごとに処理を分割することで高速化を実現した．

4. ケーススタディ

開発したシステムを用いて，リポジトリの再利用元開発者と再利用者に着目した再利用分析を行った．本節では，5 つの Java プロジェクトに対して提案手法を適用して行っ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 概　　要

実験対象として，OSS(Open Source Software) の Java プロジェクト 5 つを入力として用意した．提案手法に従いプロジェクトをマージ後，クローンセットの系譜を導出し，再利用元開発者と再利用者を導出した．そして，導出された再利用元開発者と再利用者の情報を基に分析を行った．

4.2 リポジトリの説明

ケーススタディで対象としたリポジトリについての情報を表 1 に示す．なお，表 1 の情報は 2015 年 1 月 31 日時点のものである．

表 1　対象リポジトリ

リポジトリ名	概要	リビジョン数	開発者数	開発開始日時
benten[a]	翻訳支援ツール	3258	7	2009/4/17
mergedoc[b]	Eclipse 日本語化ツール	908	4	2007/11/8
itext[c]	PDF ドキュメント作成ツール	6738	28	2000/11/29
mailmarket[d]	メーラー	1158	7	2012/2/19
davmai[e]	メーラー	2332	1	2006/12/13

[a] 脚注: http://sourceforge.jp/projects/benten/
[b] 脚注: http://sourceforge.jp/projects/mergedoc/
[c] 脚注: http://sourceforge.jp/projects/sfnet_itext/
[d] 脚注: http://sourceforge.net/projects/mailmarket/
[e] 脚注: http://sourceforge.jp/projects/sfnet_davmail/

対象としたリポジトリについて，共通の開発者の存在する java プロジェクトを Sourceforge(注1) から選定した．benten と mergedoc には共通の開発者 D が存在し，itext と mailmarket について，共通の開発者 F 及び H が存在する．davmail については，共通の開発者は存在していないが機能の似ているソフトウェアについて分析を行うために追加した．

4.3 実 験 環 境

本ケーススタディでは，AWS(Amazon Web Services)(注2) を利用し複数の仮想マシン (Virtual Machine) を用意し実験を行った．仮想マシンの OS は CentOS release 6.5(Final) である．AWS で作成した仮想マシンはインスタンスによってスペックが異なる．各インスタンスのスペックについて表 2 に示す．

表 2　仮想マシンのスペック

インスタンス名	vCPU	vCPU 数	メモリ (GiB)
t2.micro	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz	1	1
c3.xlarge	Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz	4	7.5
m3.2xlarge	Intel(R) Xeon(R) CPU E5-2670 v2 @ 2.50GHz	8	30

複数の仮想マシンの並列実行を行うスクリプトを起動するために，t2.micro の仮想マシンを一台用意した．そして，プログラムの実行を行う worker として，c3.xlarge の仮想マシンを 30 台用意した．また，PostgreSQL のデータベースサーバには m3.2xlarge の仮想マシンを利用した．

なお，STEP2 の並列化を行う際，後半のリビジョンほどファイル数が増加し処理に時間が掛かることを考慮した．つまり，前半のリビジョンを担当する仮想マシンほど担当リビジョン数を多く設定した．

4.4 分 析 項 目

本研究では，複数プロジェクトを対象とした再利用元開発者と再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 4 つの分析を行った．

（1） 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も多いか

(注1)：http://sourceforge.jp
(注2)：http://aws.amazon.com/jp/

（2） 再利用される回数の多い開発者はどのような特徴を持つか

（3） 再利用される回数の多いソースコードに特徴があるか

（4） 再利用回数の多い開発者にどのような特徴があるか

（5） プロジェクト間で行われた再利用にどのような特徴があるか

分析項目 1 では，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．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．分析項目 2 では，再利用される回数の多い開発者について特徴があるかを知ることを目的としている．分析項目 3 では，再利用されやすいソースコードについて特徴があるか知ることを目的としている．分析項目 4 では，再利用する回数の多い開発者について特徴があるか知ることを目的としている．分析項目 5 では．プロジェクト間で行われた再利用について，再利用されたソースコードや関わっている開発者に特徴があるのかを知ることを目的としている．

4.4.1 分 析 結 果

実験結果ついて示す．実験ではクローンの系譜が 5396 個検出された．また，クローンの系譜の内，10 個のクローンの系譜においてプロジェクト間での再利用が行われていた．

以下に，分析結果と考察を示す．

分析内容 1

図 3 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．

分析内容 2

再利用される回数の多いソースコードを記述する開発者について，図 reffig:devR から開発者 A と開発者 E が挙げられる．開発者 A，E は共に itext プロジェクトに所属している．両開発者共にロールは Admin である．今回の調査対象のうち，複数のプロジェクトには参加していない．開発者 A について，Sourceforge 内で 11 個のプロジェクトに所属している．開発者 E について，Sourceforbe 内で 8 個のプロジェクトに所属している．

分析内容 3

再利用された回数の多いソースコードについて 4 に示す．X 軸は再利用された回数を示し，Y 軸は再利用された回数ごとのクローンの系譜の数を示す．今回，5 回以上の再利用が行われたものは全て自分自身のソースコードの再利用であった．そして，メソッド単位やクラス単位での再利用が多かった．

分析内容 4

再利用をよく行う開発者について，図 reffig:devR から開発者 A と開発者 E が挙げられる．これは，分析内容 2 で挙がった開発者と同じである．

分析内容 5

本実験では，10 個プロジェクト間での再利用が検出された．そのうち 1 つの再利用については，getter 及び setter のみのコードクローンであったことから偶然の一致であると考えられる．そして，残りの 9 個すべての再利用について，両プロジェクトに所属する開発者が再利用元開発者かつ再利用者であった．

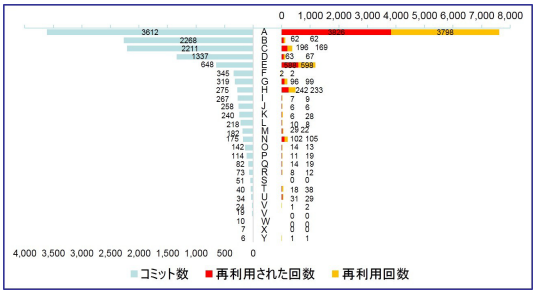

図 3 再開発者ごとのコードクローン作成数と再利用数

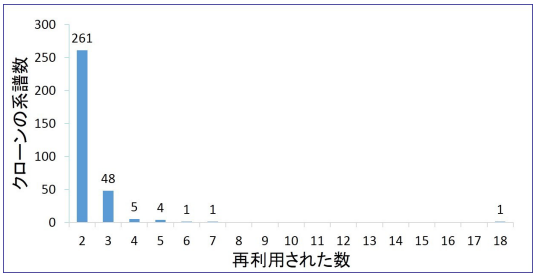

図 4 クローンの系譜の被再利用回数の分布

4.4.2 パフォーマンス

本研究では AWS により複数の仮想マシンを用意することで，STEP2 及び STEP3 での処理を並列に実行しパフォーマンスの向上を図った．各 STEP の処理時間について，表 3 に示す．なお，延べ処理時間はすべての仮想マシンでの処理時間を合計したものである．

表 3 処 理 時 間

処理	VM 数	処理時間	延べ処理時間
STEP1	1	1 分 55 秒	1 分 55 秒
STEP2	30	12 時間 44 分 42 秒	93 時間 57 分 12 秒
STEP3	30	3 分 20 秒	81 分 44 秒

5. 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 から，単純にコミット数が多ければ再利用される数，再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．

分析内容 2 及び 4 から，再利用される回数の多い開発者は主に自分のソースコードを再利用していることが分かった．つまり，自分の書いたソースコードの理解が再利用に大きく関与すると考えられる．また，自分以外の書いたソースコードを把握し理解することが，組織における再利用の促進に繋がるのではないかと考えられる．

分析内容 3 について，よく再利用されるソースコードは大きい規模での再利用によって生まれることが多いことが分かった．また，再利用が 7 回行われていたソースコードについて，ある

リビジョンにおいて特定のメソッドが 7 箇所に再利用されていた．つまり，プロジェクト内に 8 つ同じメソッドが存在する．この事例のように，特定のプロジェクトにおいて何箇所にも存在するメソッドを特定することは，ソースコードのライブラリ化の提案により組織での再利用の改善に繋がると考えられる．

そして，分析内容 5 について，プロジェクト間での再利用が行われた際，複数プロジェクトに関わっている開発者が再利用元開発者，または再利用者に含まれていた．そして，再利用は両プロジェクトで一般的に用いられるであろう文字列のトリムを行う処理に関するファイルについて行われていた．この結果より，複数プロジェクトに関わっている開発者が，プロジェクト間での再利用を行い効率よく開発を行った可能性があると考えられる．

6. むすび

本研究では版管理システムとクローンの系譜検出ツールを用いて，複数のプロジェクトにおけるクローンの系譜に基づいて再利用元開発者と再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．実際に分析手法を実装し，5 つの java プロジェクトを用いてケーススタディを実施した．ケーススタディにより，再利用元開発者と再利用者についての分析を行うことできた．分析の結果，コミット数の多い開発者は再利用によく関わっているわけではなく，開発者ごとの再利用傾向には特徴があることを示した．また，再利用される回数の多いコード片について調査することで，再利用されやすいコード片はメソッドやクラス単位といった規模の大きく処理の分かりやすいものが多いことを示した．

今後の課題を以下に挙げる．

タイプ 3 のコードクローンへの対応

本研究ではタイプ 1 とタイプ 2 のコードクローンに対して検証を行った．そのため，タイプ 3 のコードクローンへの適用方法を検討する必要がある．ただし，タイプ 3 のコードクローンは再利用されるごとに処理内容が変わる場合もあるため，発生したコードクローンと元のコード片が別物になっている可能性もあるので考慮が必要である．

大規模なリポジトリでの適用

本研究では，合計リビジョンに対して実験を行った． より有用なデータを得るためにはさらに規模の大きいリポジトリに対して結果分析を行う必要があると考えられる．

分析プロジェクト数の増加

本研究では，5 つのプロジェクトをマージし，分析を行った．プロジェクト間での再利用についての分析をより詳細に行うためには，さらに多くのプロジェクトを対象に実験を行う必要があると考えられる．

再利用支援ソフトウェアの考案

再利用動向の特定により，ライセンス違反の特定や，ライブラリ化の支援のような再利用支援が行えると考えられる．

謝辞 本研究は，日本学術振興会科研費基盤研究 (S)（課題番号 25220003）の助成を得た．

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] Naoki Fukuyasu, Sachio Saiki, Hiroshi Igaki, and Yuki Manabe. Experimental report of the exercise environment for software development pbl. In *Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, pp. 482 –487, 8 2012.

[4] 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.

[5] 鷲崎弘宜, 森田翔, 長井恭兵, 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.

[6] 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.

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

[8] 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.

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

[10] Yoshiki Higo, Keisuke Hotta, and Shinji Kusumoto. Enhancement of crd-based clone tracking. In *Proceedings of the 13th International Workshop on Principles of Software Evolution (IWPSE2013)*, pp. 28–37, 8 2013.

[11] 堀田圭佑, 肥後芳樹, 楠本真二. Crd の類似度に基づくコードクローン追跡手法. 電子情報通信学会技術研究報告, 第 113 巻, pp. 127–132, 7 2013.

[12] 堀田圭佑, 肥後芳樹, 楠本真二. Crd を用いたコードクローンの生存期間と修正回数に関する調査. ソフトウェアエンジニアリングシンポジウム 2013, pp. F03:1–F03:8, 9 2013.

[13] Ekwa Duala-Ekoko and Martin P. Robillard. Clone region descriptors: Representing and tracking duplication in source code. *ACM Trans. Softw. Eng. Methodol.*, Vol. 20, No. 1, pp. 3:1–3:31, July 2010.

[14] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, Vol. 10, p. 707, February 1966.