特別研究報告

題目

開発履歴を用いたコードクローン作成者と利用者の分析手法とその適用

指導教員
井上 克郎 教授

報告者
森脇 匠哉

平成 25 年 2 月 12 日

大阪大学 基礎工学部 情報科学科

平成 24 年度 特別研究報告

開発履歴を用いたコードクローン作成者と利用者の分析手法とその適用

森脇 匠哉

内容梗概

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といったソフトウェアの再利用が行われている．一般に，ソフトウェアの再利用は生産性，信頼性やコスト等の改善に繋がると言われており，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，特に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，具体的に誰がどのような再利用を行っているかという再利用分析を定量的に行った事例は非常に限られているのが現状である．特定の組織における再利用分析を行う場合，開発者が複数のプロジェクトに関与している場合が多い．また，既存の研究において，開発者によって再利用の状況は異なっており，再利用のメリットをよく認識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言われている．そこで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を行うことで，開発者ごとのソースコードの再利用傾向の分析を行う．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．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．そのため，再利用分析において，コードクローン検出手法は非常に良く用いられている．

版管理システムとは，ファイルの変更履歴を管理するために用いられるシステムである．変更履歴はリビジョンという単位でリポジトリと呼ばれるデータベースに保持される．本研究では，版管理システムとコードクローン検出ツールを組み合わせることで，開発において行われたコード片間の再利用状況の遷移を辿ることができる．

提案手法では，最初に複数プロジェクトのリポジトリを 1 つのリポジトリにマージする．次に，隣接するリビジョン間でクローン検出を行い，クローンセットにコードクローンが

追加された，または削除されたというクローンセット遷移情報を導出する．そして，2 リビジョン間のクローンセット遷移情報のマージを行い，複数リビジョンにおけるクローンセット遷移情報をクローンセット履歴として保持する．クローンセット履歴を分析することで，そのクローンセットの起源を辿ることが可能となる．ここで，あるクローンセットが発生した際に，その元コード片を記述した開発者をコードクローン作成者と定義する．また，そのコード片を再利用した開発者をコードクローン利用者と定義する．そして，各クローンセットの再利用履歴と，そのコードクローン作成者と利用者を導出することで，再利用した利用者数の多いコードクローンや積極的に再利用を行っている開発者といった開発者ごとの再利用に関する振る舞いを計測することが可能となる．

本手法を用いて，実際に OSS から eclipse.platform.text，eclipse.pde の 2 つの Java プロジェクトを用意し，コードクローン作成者と利用者について分析を行った．分析したプロジェクトは合わせて約 1500 リビジョン，開発者は約 20 名の規模である．

分析の結果，コードクローンの利用数が極端に多い開発者や，コードクローンを作成も利用もしていない開発者がいるといったように，開発者ごとの再利用傾向には違いがあり，さらなる分析を進める価値があることを示した．

主な用語

再利用
版管理システム
コードクローン
Code Authorship
Origin Analysis

目 次

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．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．

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やプロジェクトの性質，プロジェクトのステージによって異なることを示した [3]．また，この研究によって，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いうことが確認された．

再利用傾向の分析に関する他の研究としては，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，鷲崎らの研究では特定のコード片が再利用されたかどうかのみに着目しており，開発者単位での再利用傾向は示されていない．

ソースコードの再利用検出は，コードクローンを用いて行うことが可能である [5]．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 [6]．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．そのため，再利用分析において，コードクローン検出手法は非常に良く用いられている．そして，多くのコードクローン検出ツールが提案されている [7, 8, 9]．

Mihai Balint らは，コードクローンを用いて開発者単位での再利用傾向の分析を行っている [10]．Balint らの研究では，開発者のコードクローン保守作業に着目し可視化を行っている．具体的には，コードクローンの各行について開発者情報を付加することで，開発者ごとのコピー・編集活動を分析しており，各コードクローンについて誰にいつ編集されたかを可視化する仕組みを構築した．しかし，開発者ごとの再利用傾向を可視化しているが，定量的な評価は行っていない．また，3 つのプロジェクトの各 1 バージョンについてを対象としており，版管理システムを用いた過去の開発履歴の分析は行っていない．

そこで，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ツールを用い，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．版管理システムとは，ファイルの変更履歴を管理するために用いられるシステムである．版管理システムにおいて，変更履歴はリビジョンという単位でリポジトリと呼ばれるデータベースに保持される．

本研究では，版管理システムとコードクローン検出ツールを用いて，コードクローン作成者と利用者を導出し，開発において行われたコード片間の再利用状況を分析する手法を提案する．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．次に，隣接するリビジョン間でクローン検出を行い，クローンセットにコードクローンが追加された，または削除されたというクローンセット遷移情報を導出する．そして，2 リビジョン間のクローンセット遷移情報をコードクローンのパス情報を基にマージすることで，複数リビジョンにおけるクローンセット遷移情報をクローンセット履歴として保持する．クローンセット履歴を分析することで，そのクローンセットの起源を辿ることが可能となる．そこで，あるクローンセットが発生した際に，その元コード片を記述した開発者をコードクローン作成者と定義する．また，そのコード片を再利用した開発者をコードクローン利用者と定義する．各クローンセットの再利用履歴と，そのコードクローン作成者と利用者を導出することで，再利用したユニークユーザ数の多いコードクローンや積極的に再利用を行っている開発者といった開発者ごとの再利用に関する振る舞いを計測することが可能となる．

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 節では評価実験について述べる．6 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　背景

本研究の背景として，ソフトウェアの再利用，版管理システム，コードクローンと再利用分析に関する既存研究について説明する．

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．

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 [3]．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．

再利用傾向の分析については，鷲崎らが研究を行っている [4]．鷲崎ら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示している．例えば，外部結合グローバル変数の使用により多くの他ファイルに依存している関数は再利用性が低いことが示されている．このように鷲崎らの研究では特定のコード片が再利用されたかどうかの実績を分析しており，開発者単位での再利用傾向は示されていない．

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．

このように再利用分析に関する研究は数多く行われているが，組織における開発者個人に着目した定量的な分析は殆ど行われていない．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

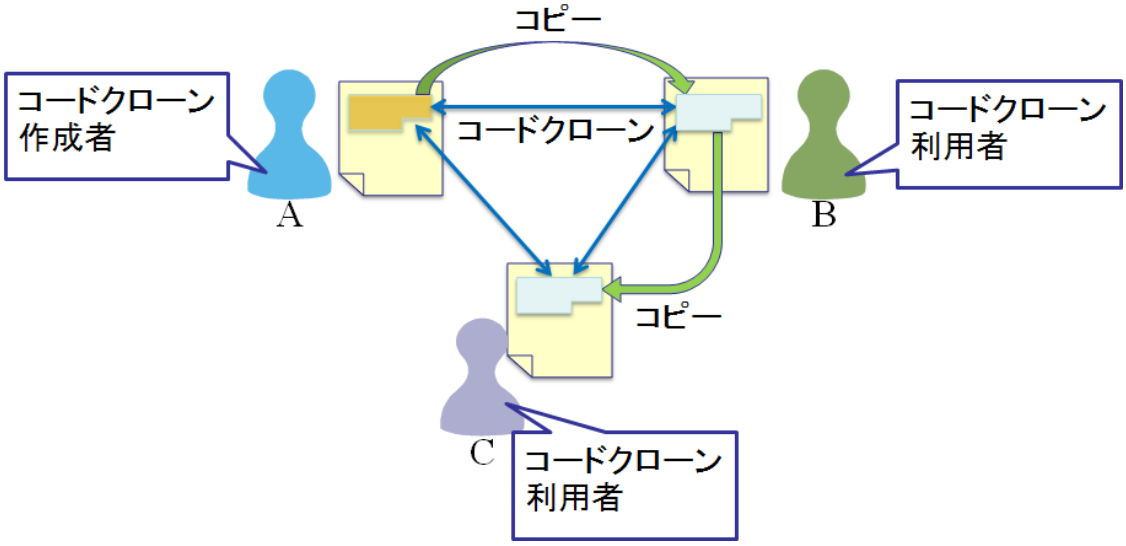


図 1: コードクローン利用者と作成者

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[1]．作成者と利用者の関係を図 1 に示す．図 1 が示すように，開発者個人に着目した再利用分析を行うためには再利用元のコード片がいつ，誰によって開発されたのかを調査する必要がある．そのため，版管理システムを対象とした分析が行われることが多い．以降の節では，版管理システムと版管理システムを用いたコード片ごとの開発者分析を行う既存手法について詳述する．

ここで，既存のソフトウェアの再利用検出手法について説明する．

2.1.1 Code Authorship

後述する版管理システムを用いることで過去のソースコードの取得や，Code Authorship の取得を行うことが可能となる．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．Code Authorship により，再利用元と再利用先のコード片があったときに，再利用元を誰が作成したか知ることができる．プログラムのソースコードが Git[2]や Subversion[3] などの版管理システムで管理されている場合，コマンドによって Code Authorship を導出することができる．Git では blame コマンドで Code Authorship の取得やソースコードのコミットが行われた日時情報取得が可能である．図 2 に Code Authorshjp についての説明と

[1] コードクローンの詳細については 2.3 節で詳述する
[2] http://git-scm.com/
[3] http://subversion.tigris.org/

blame コマンドでの出力例を示す．図 2 (a) では，DeveloperA がコミットしたファイルにバグが見付かり，DeveloperB がソースコードの修正を行った場合についての Code Authorship を示す．また，図 2 (b) にはソースコード修正後に blame コマンドを実行した場合の出力例を示す．例示したように，blame コマンドでの出力にはコミットの ID，Code Authorship，コミット日時，行番号，コード内容が含まれている．そして，修正行の Code Authorship が DeveloperB となっていることが確認できる．

2.1.2 Origin Analysis

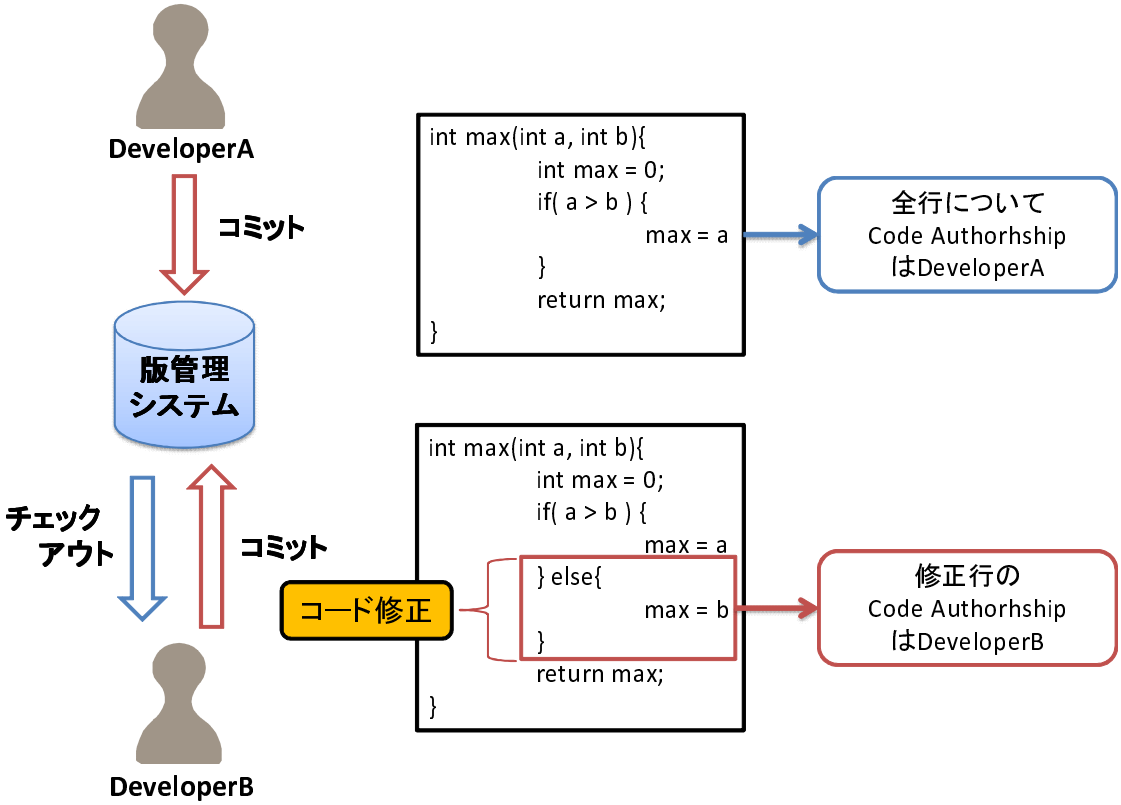
Michael Godfrey らは，あるバージョンで新しいソフトウェアエンティティが生成された際，そのソフトウェアエンティティが新たに出来たものであるか，あるいは以前のバージョンのものを再利用して作られたものかを判断する手法 (Origin Analysis) について研究を行っている [11]．Origin Analysis は，Entity Analysis と Relationship Analysis の二つの段階に分けられる．

Entity Analysis

ソフトウェアエンティティごとに一種の指紋を生成する．基本的なメトリクスであるライン数，コメント数，循環的複雑度 (Cyclomatic complexity)，グローバル変数アクセス数などに加え，2 つのエンティティの比較を容易にするためのメトリクスを計算する．これらのメトリクスを用いることで，あるエンティティについての比較を行う場合に，関係のありそうな候補集合から選ぶことが可能となる．

Relationship Analysis

“あるエンティティに名前変更があった場合でも，他のエンティティとの関係は変わらない見込みが高い．”という考えに基づいて Relationship Analysis が行われる．例えば，あるエンティティF に対して分析を行う場合，最初に F を呼び出す関数集合と F に呼び出される関数集合を求める．そして，以前のバージョン中の候補集合についても同様の関数集合を求め，その集合の共通部分に基づいて調査することでエンティティ同士の関係を知ることができる．



(a) コード修正の様子

```
dac483b5 (DeveloperA 2013-02-10 13:08:07 +0900 1) int max(int a, int b){
dac483b5 (DeveloperA 2013-02-10 13:08:07 +0900 2)     int max = 0;
dac483b5 (DeveloperA 2013-02-10 13:08:07 +0900 3)     if( a > b ) {
dac483b5 (DeveloperA 2013-02-10 13:08:07 +0900 4)         max = a
79542f34 (DeveloperB 2013-02-10 14:09:41 +0900 5)     } else{
79542f34 (DeveloperB 2013-02-10 14:09:41 +0900 6)         max = b
79542f34 (DeveloperB 2013-02-10 14:09:41 +0900 7)     }
dac483b5 (DeveloperA 2013-02-10 13:08:07 +0900 8)     return max;
dac483b5 (DeveloperA 2013-02-10 13:08:07 +0900 9) }
```

(b) blame コマンド実行

図 2: blame コマンド出力例

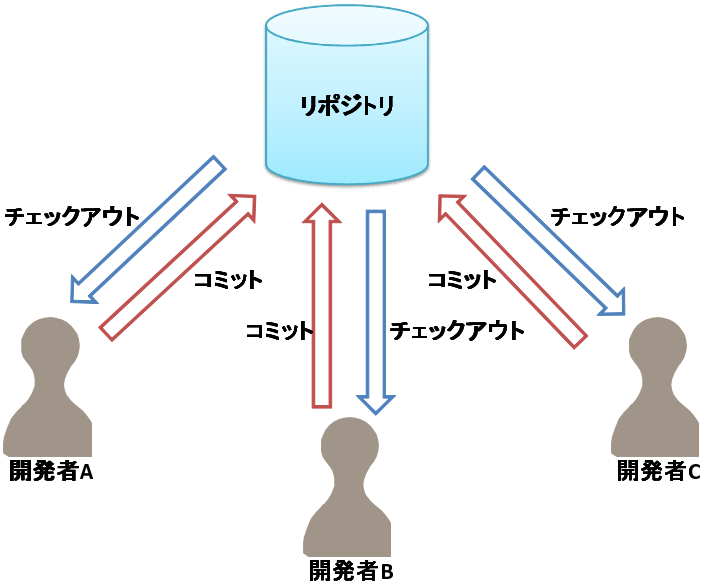


図 3: 版管理システム

2.2 版管理システム

版管理システムとは Git や Subversion など，ファイルの変更履歴を管理するために用いられるシステムである．ファイルにバージョン付けを行うことでファイルの追加，削除，修正といった変更履歴を管理し，過去の変更内容の確認や履歴をさかのぼってのファイル修正などを行うことができる．

図 3 に版管理システムのイメージ図を示す．各バージョンのファイルの履歴データは，リポジトリ (Repository) に蓄積される．リポジトリでは，蓄積されたファイルをリビジョン (Revision) を単位として管理する．リビジョンごとに，ファイル，作成日時，ログメッセージなどの属性データが保管される．リポジトリからあるリビジョンのファイルを取得することをチェックアウト (Checkout) という．逆に，ファイルをリポジトリに格納し，新たなリビジョンを作成することをコミット (Commit) という．開発者は，リポジトリからファイルをチェックアウトし，変更内容をコミットすることで開発を行う．

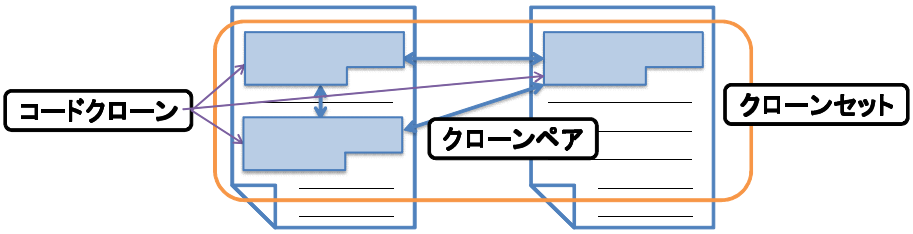


図 4: コードクローン

2.3 コードクローン

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 [6]．互いにコードクローン関係にあるコード片をクローンペア，コードクローン関係にある全てのコード片集合をクローンセットという．コードクローン，クローンペア，クローンセットの関係を図 4 に示す．あるコード片に欠陥が含まれている場合，そのコード片の属するすべてのコードクローンについても同様の欠陥が含まれている可能性が高く，それらを修正するのは大きなコストとなる．以上の理由から，一般にソースコード中のコードクローンは少ないほうがよいとされている．しかし，最近の研究で悪影響を及ぼさないコードクローンがあることがわかった [12]．例えば，言語の制約やプロジェクトをまたがる開発によって発生したコードクローンである．

2.3.1 発生原因

コードクローンがソースコード中に発生する原因を以下に示す [13, 14]．

既存コードのコピーアンドペーストによる再利用

ソースコードを一から書くよりも，同様のまたは類似した処理を行う既存コードを流用し，部分的な変更を加える方が信頼性が高い．そのため，コピーアンドペーストによる既存コードの再利用が多く存在する．

定型処理

定義上簡単で頻繁に用いられる処理はコードクローンになる傾向がある．例えば，給与税の計算や，キューの挿入処理，データ構造アクセスなどである．

プログラミング言語における適切な機能の欠如

抽象データ型や，ローカル変数を用いることができない場合，類似したアルゴリズムをもつ処理を繰り返し書かなくてはならない場合がある．

パフォーマンスの改善

リアルタイムシステムなどの時間制約のあるシステムにおいて，インライン展開などの機能が提供されていない場合，特定のコード片を意図的に繰り返し記述することでパフォーマンスの改善を図ることがある．

コード生成ツールの生成コード

コード生成ツールはあらかじめ定められたコードをベースにして自動的にコード生成を行う．そのため，目的とする処理が類似している場合，識別子などを除いて類似したコードが生成される．

複数のプラットフォームに対応したコード

複数の OS や CPU に対応したソフトウェアは，各プラットフォーム用のコード部分に重複した処理が存在する可能性が高い．

偶然

偶然，開発者が同一のコードを記述することがある．

2.3.2 コードクローンの定義

コードクローンには様々な検出手法が提案されており，その手法ごとに異なったコードクローンの定義をもつ．

Bellon は，コードクローンを以下に示す三つの種類に分類している [15]．

タイプ 1

空白やタブの有無，括弧の位置などのコーディングスタイルを除くと，完全に一致するコードクローンを指す．

タイプ 2

変数名や関数名などのユーザ定義名，または変数の型などの一部予約語のみが異なるコードクローンを指す．

タイプ 3

タイプ 2 における変更に加えて，文の挿入や削除，変更が行われたコードクローンを指す．

本研究では，タイプ 1 及びタイプ 2 のコードクローンを対象とする．

2.3.3 コードクローン作成者と利用者

コードクローンが発生した際，その元コードを記述した開発者をコードクローン作成者と定義する．また，元コード片を再利用し，コードクローンを発生させた開発者をコードクローン利用者と定義する．

2.3.4 Authorship

コードクローンが異なった定義を持つように，その Authorship についても様々な定義が存在する．

コードクローンの Authorship を調査する研究として，Harder らの研究がある [16]．Harbar らは，トークンベースでコードクローンの Code Authorship を求め，最も多くのトークンの Code Authorship である開発者を，そのコードクローンの Clone Authorship と定義した．ここで，Clone Authorship はコードクローン毎に求められるため，クローンペアで異なる Clone Authorship となる場合も考えられる．そして，クローンペアの Clone Authorship が同一か異なるかでクローンセットの分類を行い，それぞれコードクローンへの一貫した修正がなされているかどうかを検証した．検証により，複数の Clone Authorship をもつクローンセットは一貫した修正がなされにくいことを示している．

以下に，本研究で用いる Authorship 関連の用語の定義を示す．

Code Clone Authorship

本研究では，版管理システムを用いてコードクローンの各行の Code Authorship を求めることで，その過半数を占める開発者を Code Clone Authorship と定義する．

Clone Set Authorship

あるクローンセットについてそのクローンセットのコードクローン作成者を Clone Set Authorship と定義する．

2.3.5 検出ツール

コードクローン検出手法には，ソースコードの字句解析に基づく手法 [14, 17, 18] や，特徴メトリクスに基づく手法 [19, 20] などがある．ソースコードの字句解析に基づく手法では，ソースコード中の同一文字列を検索することによりコードクローン検出を行う．特徴メトリクスに基づく手法では，クラスや関数，ファイルなどのプログラム中のある種の単位ごとに特徴メトリクスを定義・算出し，それらのメトリクス値が類似したものをクローンとして抽出することにより検出を行う．一般には字句解析に基づく手法のほうがコストが増えるが，より細粒度なクローンを抽出できる．

本実験では，字句単位の検出を行う CCFinder をコードクローン検出ツールとして用いる [14]．CCFinder は高いスケーラビリティを有しており，大規模なソフトウェアに対しても実用的な時間でコードクローン検出を行うとができる．また，実際にさまざまな大規模ソフトウェアへ適用され，その有用性が確認されている [21]．CCFinder では字句解析でソースコードをトークン列に変換し，変換処理で変数名や関数名等を同一のトークンに変換する．その後，検出処理で閾値以上の長さの共通トークン列を探索し，全てのコードクローンの対のリストを出力する．従って，CCFinder はタイプ 1，タイプ 2 のコードクローンを検出することが可能である．

2.4 クローンセット履歴分析

クローンセット履歴とは，ある開発期間において，どのリビジョンでクローンセットが発生したか，またはクローンセットにコードクローンが追加されたかといったクローンセットの遷移情報のことである．あるリビジョンで新たなコードクローンが発生した時，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コードクローンであればソースコードの再利用が行われたとみなせる．

2.4.1 コードクローン変更管理

複数リビジョン間でのコードクローン変更管理に関して様々な研究が行われている．

Miyung Kim らは，コードクローンの履歴 (Code Clone Genealogies) を分析する手法を用いた研究を行っている [12]．Kim らは，クローンセットの発生から消滅までの生存期間について着目し，その違いによってクローンセットにどのような特徴があるかを調査している．なお，コードクローン検出には CCFinder を用いている．

川口らは，クローンセットの分岐を分析するため，コードクローンの履歴分析手法 (Clone History Analysis) を提案している [22]．この提案手法では，まず過去のリビジョンのクローンセットと現在のバージョンのクローンセットの間でのコードの変化について調べる．そして，変化のあったコードについて過去のリビジョンと現在のリビジョンでの対応関係を求めることでコードクローンの遷移を検出している．なお，コードクローン検出には CCFinder を用いている．

本研究では，山中らの提案する 2 リビジョン間でのコードクローンの変更状況分析手法を参考にしている [23]．以降，山中らの手法について説明を行う．表 1 に説明に用いる記号を示す．

STEP1：**コードクローンの親子関係**

R_t と R_{t+1} 間のコードクローンの親子関係を求めるには，各リビジョンで発生したコードクローンについての対応付けを行う必要がある．ここで， C_t をリビジョン R_t で検出された全コードクローンの集合として説明を行う．あるコードクローン $A \in C_t$ について，コードクローンの開始行と終了行に基づいて，対応するコードクローン $B \in C_{t+1}$ が存在するか求める．コードクローン B が存在し，コードクローン A とクローンペアである場合，コードクローン A をコードクローン B の親クローン，コードクローン B をコードクローン A の子クローンと定義する．また，そのような場合コードクローン A とコードクローン B の間には親子関係が存在すると定義する．

本手法では，GNU diff[4]を用いてソースコードの差分を取得することで，リビジョン間のコードクローンの対応付けを行っている．

STEP2：**コードクローンの分類**

R_t と R_{t+1} に存在するコードクローンについて，コードクローンの親子関係に基づいて分類を行う．本研究では，“Stable Clone”，“Modified Clone”，“Added Clone”，“Deleted Clone”の 4 つに分類分けを行う．

Stable Clone

R_t ， R_{t+1} のリビジョン間で親子関係にあり，コード片に差分のないコードクローンを Stable Clone と定義する．

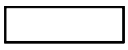
Modified Clone

R_t ， R_{t+1} のリビジョン間で親子関係にあり，コード片に差分のあるコードクローンを Modified Clone と定義する．

Added Clone

R_{t+1} に存在する，親クローンを持たないコードクローンを Added Clone と定義する．

表 1: 記号一覧

コードクローン	コード片	クローンの種類	クローンセット
			

[4] http://www.gnu.org/software/diffutils/

Deleted Clone

R_t に存在する，子クローンを持たないコードクローンを Deleted Clone と定義する．

STEP3：クローンセットの分類

R_t と R_{t+1} に存在するクローンセットについて，クローンセットに属するコードクローンの種類に基づいて分類を行う．本研究では，“Stable Clone Set”，“Changed Clone Set”，“New Clone Set”，“Deleted Clone Set”の 4 つに分類分けを行う．図 5 にクローンセットの分類を示す.

Stable Clone Set

R_t ， R_{t+1} の両バージョンに存在するクローンセットで，クローンセット内の全コードクローンが Stable Clone であるものを Stable Clone Set と定義する．図 5(a) に Stable Clone Set を例示する．

Changed Clone Set

R_t ， R_{t+1} の両バージョンに存在するクローンセットで，クローンセット内に Modified Clone，Added Clone，Deleted Clone であるコードクローンが一つでも含まれているものを Changed Clone Set と定義する．Modified Clone を含む Changed Clone Set を図 5(b) に，Added Clone を含む Changed Clone Set を図 5(c) に，Deleted Clone を含む Changed Clone Set を図 5(d) に例示する．なお，Modified Clone と Added Clone クローンを含むクローンセットのように，図 5(b)〜(d) を組み合わせたような Changed Clone Set も存在する．

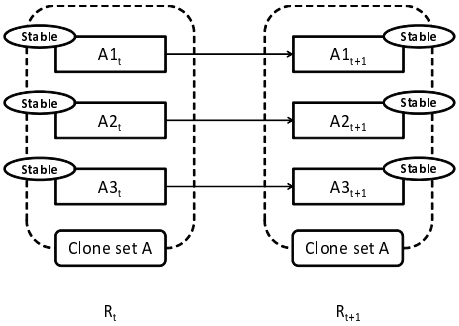
New Clone Set

R_{t+1} にのみ存在するクローンセットを New Clone Set と定義する．図 5(e) に New Clone Set を例示する．

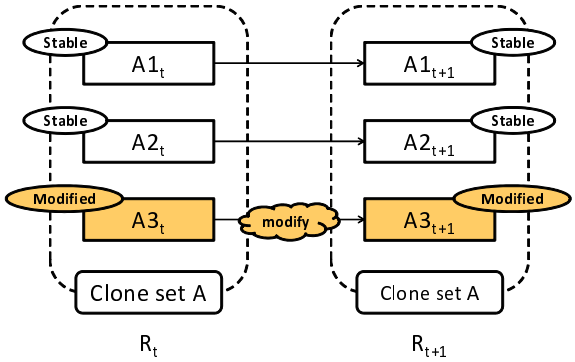
Deleted Clone Set

R_t にのみ存在するクローンセットを Deleted Clone Set と定義する．図 5(f) に Deleted Clone Set を例示する．

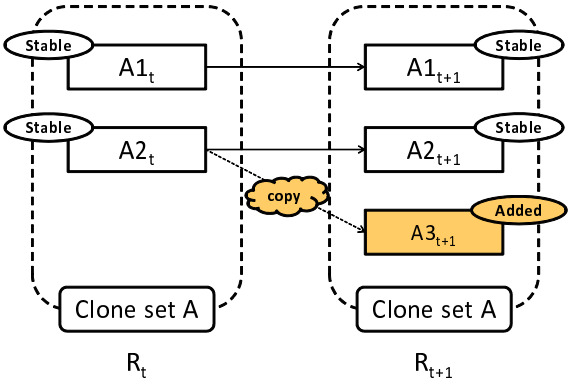
次節では，これらのコードクローン技術を応用し，開発者毎の再利用分析手法について詳述する．



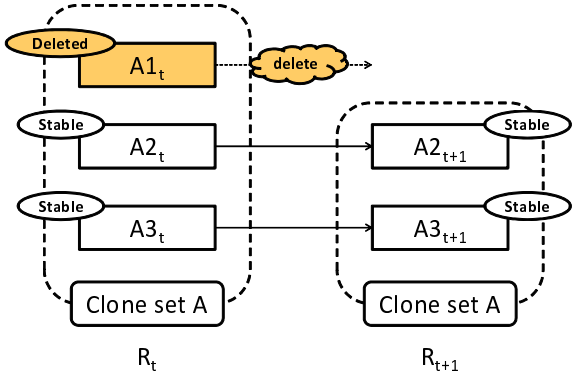
(a) Stable Clone Set



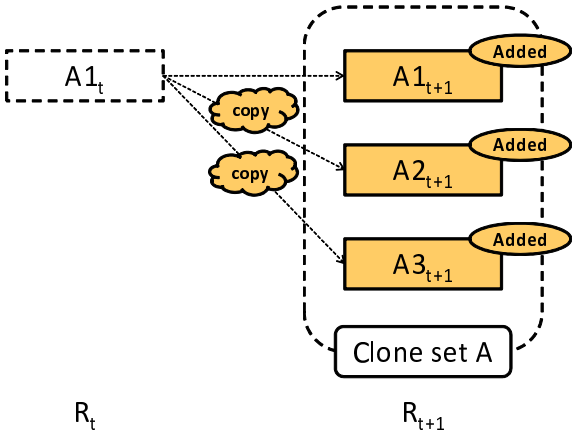
(b) Changed Clone Set [Modified]



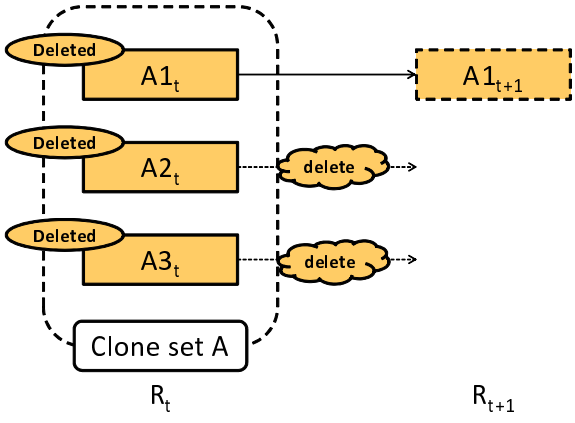
(c) Changed Clone Set [Added]



(d) Changed Clone Set [Deleted]



(e) Added Clone Set



(f) Deleted Clone Set

図 5: クローンセットの分類

3　提案手法

本研究では，リビジョン間のコードクローン検出を行い，その結果からクローンセット履歴を求めることで，クローンセットが発生した，またはクローンセットに新たなコードクローンが追加された場合の元コード片の情報を得る．

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する．

STEP2：**隣接するリビジョン間でのクローン遷移情報の導出**

隣接するリビジョンの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**

全コードクローンの遷移情報を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の特定を行う．

出力

クローンセットごとに，コードクローン作成者とそのコードクローンの発生リビジョン，パスなどの情報及びコードクローン利用者とそのコードクローンの情報が出力される．

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．

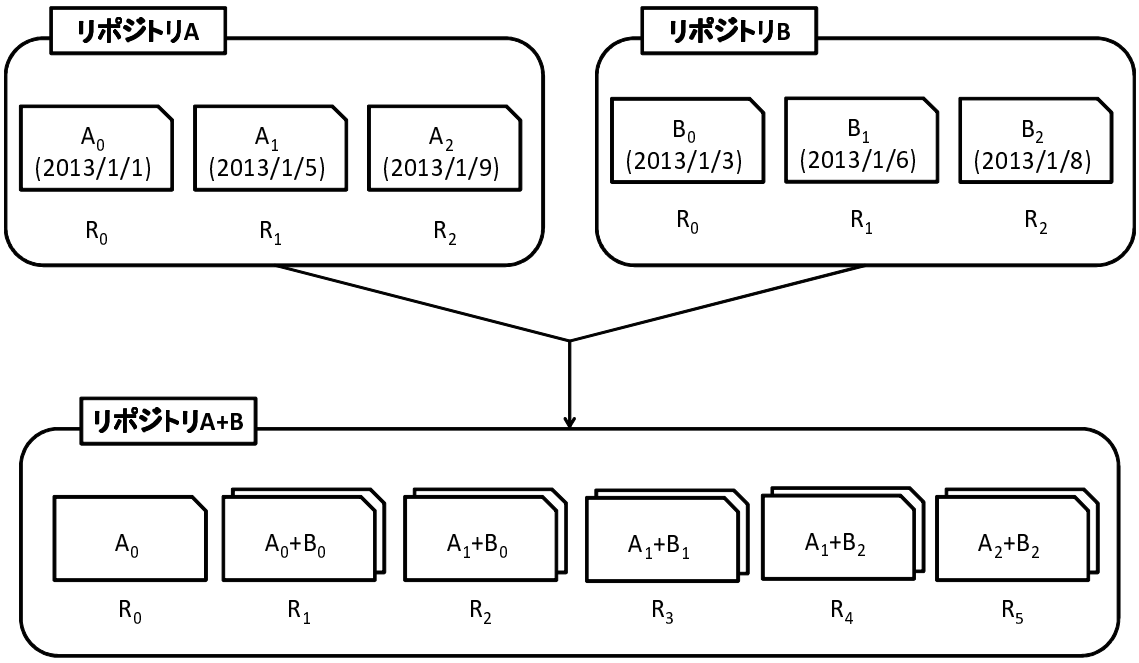


図 6: リポジトリのマージ手法

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，リポジトリ $A+B$ とする様子を図 6 に示す．2 つのリポジトリ A，B の内，最もコミット日時の古いものをリポジトリ $A+B$ の初期リビジョンとする．以降，時系列順にチェックアウトしていくことでリポジトリのマージを行うことができる．

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一つのリポジトリが出力として得られる．

3.3 STEP2：隣接するリビジョン間でのクローン遷移情報の導出

2 リビジョン間でのクローン遷移情報を導出する．

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 クローン遷移情報の抽出手法

最初に，リポジトリの各リビジョンを時系列順にチェックアウトする．そして，隣接したリビジョン間のコードクローンを検出し，節 2.4.1 で説明した山中らの手法を用いてクローンセットの遷移情報を求める．

3.3.3 出力

隣接したリビジョン間のクローンセット遷移情報が出力される．

3.4 STEP3：クローンセット履歴の抽出

2 リビジョン間でのクローン遷移を全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定義する．

3.4.1 入力

入力として，節 3.2 で得たリポジトリを与える．

3.4.2 クローンセット履歴の抽出手法

R_t と R_{t+1} ， R_{t+1} と R_{t+2} でクローンセットの分類が行われている場合，クローンセットごとに遷移情報を結合することで R_t 〜 R_{t+2} までのクローンセット履歴を得ることができる．また，マージの際には，クローンセットに属するコードクローンのうち Added Clone についての情報を保持する．さらに，その Added Clone を含むクローンセットが New Clone Set か Changed Clone Set のどちらであるかを保持することにより，コード片が追加されたことでコードクローンが発生したのか，または前リビジョンのクローンセットにコードクローンが追加されたのかを知ることができる．

以上の処理を繰り返せば，全開発履歴におけるクローンセット履歴の取得が可能である．

3.4.3 出力

クローンセット履歴をまとめたデータ構造が得られる．クローンセット履歴では，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．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 7 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．

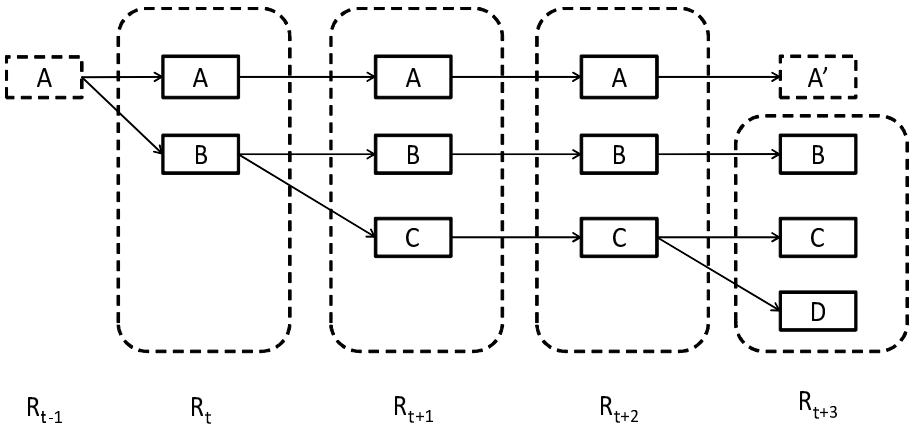


図 7: クローンセット変更の様子

3.5 STEP4 : コードクローン作成者と利用者の特定

3.5.1 入力

入力として，節 3.4 で得たクローンセット履歴を与える．

3.5.2 コードクローン作成者と利用者

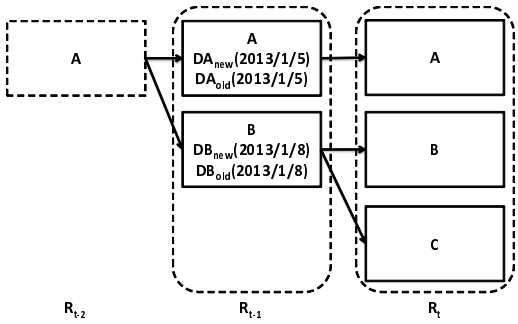
あるリビジョンで新たに発生したコードクローンについて，そのコードクローンの Code Authorship からコードクローン利用者を導出する．そして，クローンセット履歴の分析を行うことで，あるリビジョンで発生したコードクローンがどのリビジョンのコード片が元になっているかを調査する．元コード片の各行の Code Authorship を求めることでコードクローン作成者を導出する．

本研究では，コードクローン作成者と利用者について分析することで再利用したユニークユーザ数の多いコード片や積極的に再利用を行っている開発者といった開発者ごとの再利用に関する振る舞いを計測するシステムの開発を行っている．

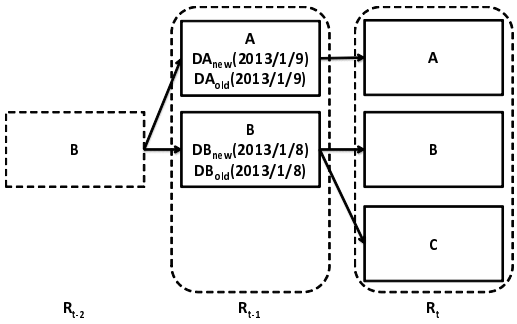
3.5.3 導出方法

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

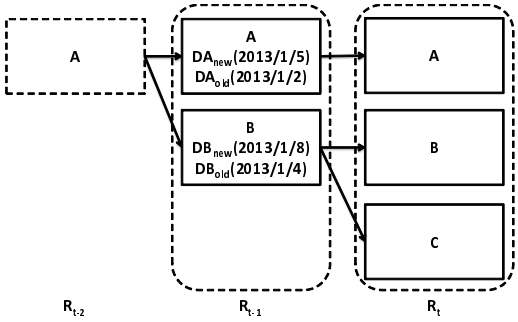
あるリビジョンでコードクローン C_t が発生したとする．そのコードクローン C の Code Authorship である開発者がコードクローン利用者である．最初に，クローンセット内に親クローンを持つコードクローンがなくなるまでリビジョンをさかのぼる．次に，クローンセッ



(a)CASE1



(b) CASE2



(c) CASE3

図 8: コードクローン作成者導出

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 8 (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 8 (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 8 (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．

3.5.4 出力

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．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　実装

本研究では，提案手法 3.3 と 3.4 についての実装を Java で行った．実装行数は約 3000 行である．以下に開発環境を示す．

- CPU：Intel Xeon E5420 2.50GHz
- メモリ (RAM)：16GB
- OS：Windows 7 Enterprise,　Service Pack 1(64 ビット)
- JDK：1.7.0
- Eclipse：4.2.0

本プログラムは，Git でソースコードの管理が行われている Java プロジェクトに対しての適用を想定している．システムはユーザが用意したリポジトリからソースコードのチェックアウトや Code Authorship を取得することで分析を行う．分析によって得られるコードクローン作成者と利用者の情報は csv ファイルで出力される．

以下，実装についての説明を行う．

4.1　Main クラス

Main クラスではリポジトリのパスの設定や出力ファイルディレクトリの設定などを行う．その後，ログの取得，チェックアウトやコードクローンの検出，分類などを行う．

4.2　データ構造

データ構造に関するクラスのクラス図を図 9 に示す．各クラスについて説明を行う．

Project クラス

2 リビジョン間のクローン情報を保持するためのクラス

Clone クラス

コードクローンの情報を保持するためのクラス

CloneSet クラス

クローンセットの情報を保持するためのクラス

CloneAuthor クラス

コードクローンの著者情報を保持するためのクラス

Line クラス

コードクローンの行ごとの CodeAuthorship とコミット日時を保持するためのクラス

CloneSetHistory クラス

クローンセット履歴を保持するためのクラス

CloneHistory クラス

クローンセット履歴において，コードクローンが追加されたことによって新たにクローンセットができたという情報や，クローンセットにコードクローンが追加されたという情報を保持するクラス

4.3 Git コマンド

本実験では，Git コマンドを利用して開発履歴の取得を行った．Git コマンドは GitSummary クラスに実装を行った．以下，実装した Git コマンドの機能を示す．なお，コマンドの実装には JGit[5] を用いた．

- git log
  リポジトリのログ情報を取得する．
- git checkout
  リポジトリから特定リビジョンのディレクトリ構造をチェックアウトする．
- git blame
  ソースコードの各行の Code Authorship とコミット日時を取得する．

4.4 コードクローン検出

コードクローン検出に関するクラスを以下に示す．

CCFinder クラス

CCFinder の実行を行いコードクローンを検出し，Clone クラスに保持する．なお，検出するコードクローンの最低トークン長を 30 に設定した．

CloneDiffDetecter クラス

GNU diff を用いてファイル間の差分を検出し，2 リビジョン間のコードクローンの対応関係を保持する．

[5] http://www.eclipse.org/jgit/

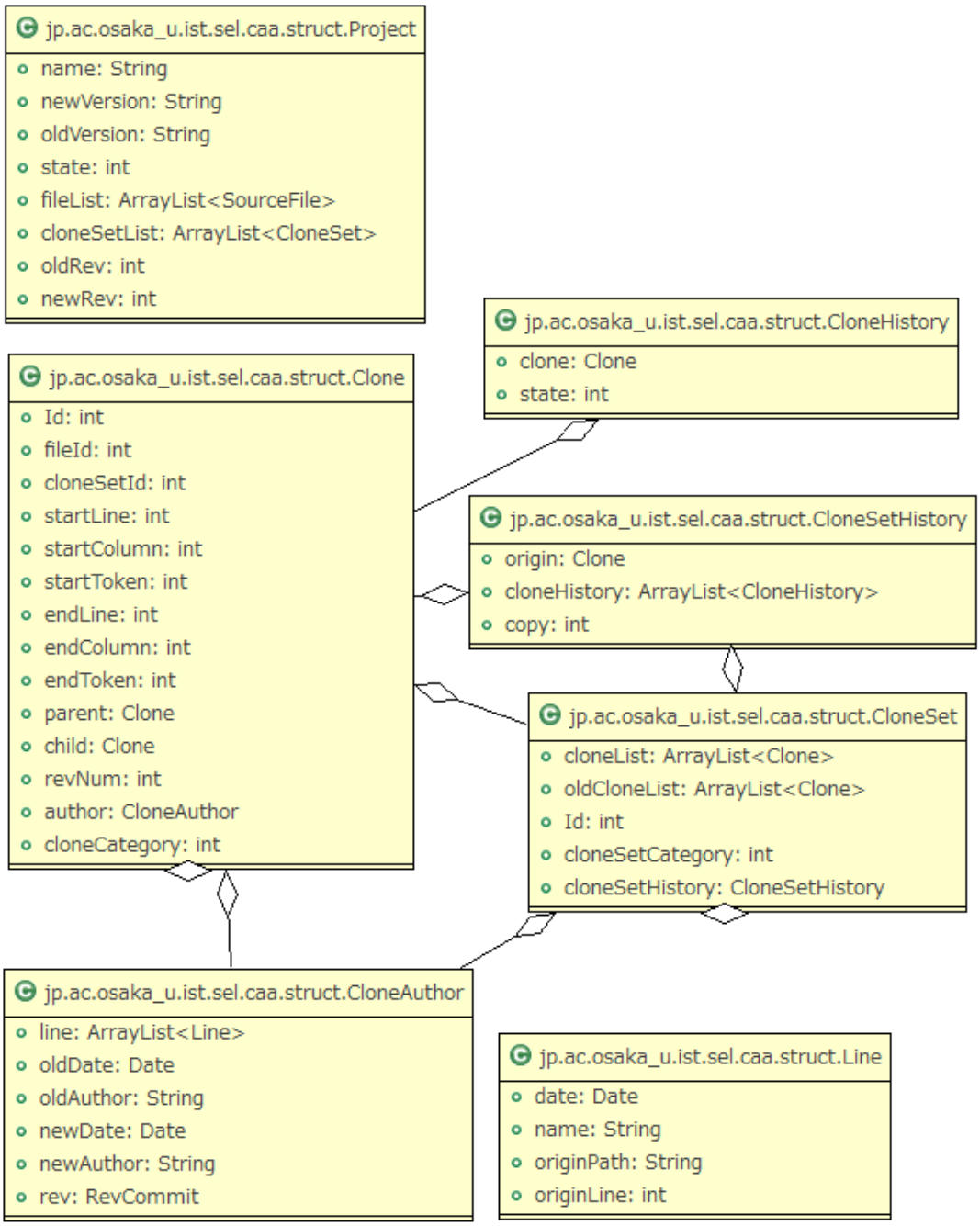

jp.ac.osaka_u.ist.sel.caa.struct.Project
name: String
newVersion: String
oldVersion: String
state: int
fileList: ArrayList<SourceFile>
cloneSetList: ArrayList<CloneSet>
oldRev: int
newRev: int
jp.ac.osaka_u.ist.sel.caa.struct.CloneHistory
clone: Clone
state: int
jp.ac.osaka_u.ist.sel.caa.struct.Clone
Id: int
fileId: int
cloneSetId: int
startLine: int
startColumn: int
startToken: int
endLine: int
endColumn: int
endToken: int
parent: Clone
child: Clone
revNum: int
author: CloneAuthor
cloneCategory: int
jp.ac.osaka_u.ist.sel.caa.struct.CloneSetHistory
origin: Clone
cloneHistory: ArrayList<CloneHistory>
copy: int
jp.ac.osaka_u.ist.sel.caa.struct.CloneSet
cloneList: ArrayList<Clone>
oldCloneList: ArrayList<Clone>
Id: int
cloneSetCategory: int
cloneSetHistory: CloneSetHistory
jp.ac.osaka_u.ist.sel.caa.struct.CloneAuthor
line: ArrayList<Line>
oldDate: Date
oldAuthor: String
newDate: Date
newAuthor: String
rev: RevCommit
jp.ac.osaka_u.ist.sel.caa.struct.Line
date: Date
name: String
originPath: String
originLine: int

図 9: データクラス

CloneCategorizer クラス

コードクローンやクローンセットの分類分けを行う.

4.5 クローン遷移情報のマージ

クローン遷移情報のマージは，Main クラスの mergeClone メソッドで実装している．コードクローンのリビジョンやパスが一致するかでマージを行っている.

4.6 ファイル出力

ファイル出力に関するクラスを以下に示す.

CloneWriter クラス

リビジョン毎のコードクローンをすべて出力する outputAuthor メソッドと，クローンセット履歴のファイル出力を行う showCloneHistory メソッドを実装している.

5　評価実験

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て行った再利用分析に関する評価実験について詳述する．

5.1　実験内容

実験対象として，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．

提案手法に従いプロジェクトをマージ後，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について
2. プロジェクト間コードクローンの作成者と利用者について，両プロジェクトで開発をしている開発者とそうでない開発者で違いがあるかについて
3. 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
4. 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
5. 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の分析

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，プロジェクト間での再利用を行う開発者にどのような特徴があるかを知ることを目的としている．分析内容 3 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 4 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5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．

5.2 実験結果

実験結果と考察について示す．実験ではクローンセットが969個検出された．また，複数の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．本実験では，この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．

以下に，分析結果と考察を示す．

分析内容 1

図10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

本実験では，5つのプロジェクト間コードクローンが検出された．そして，両プロジェクトにまたがって開発をしている開発者の内の一名がコードクローンの作成，利用に関わっていた．

分析内容 3

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 4

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[6] 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

[6] http://dash.eclipse.org/dash/commits/web-app/active-committers.cgi

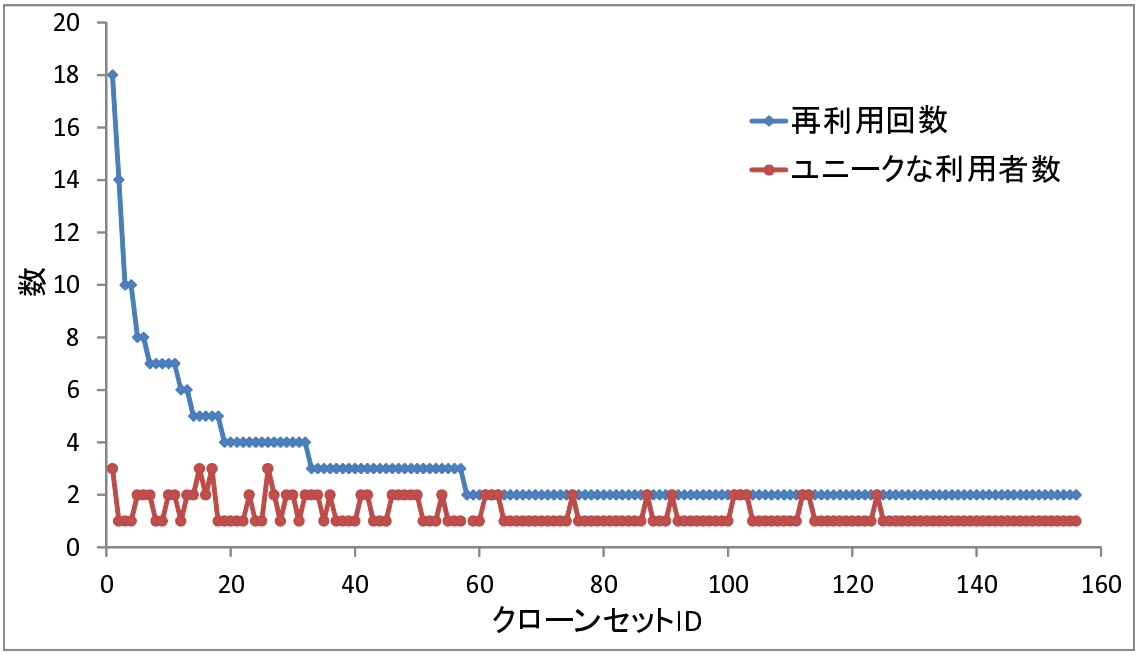


図 10: 再利用回数の多いクローンセットとユニーク利用者数の多いクローンセットの関係

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 5

図 11 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．

5.3 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．また，分析内容 2 について，プロジェクト間コードクローンが生成された際は，複数プロジェクトに関わっている開発者がコードクローン作成者，または利用者に含まれていた．この結果より，複数プロジェクトに関わっている開発者はプロジェクト間での再利用を行い効率よく開発を行っている可能性があると考えられる．分析内容 3 について，再利用回数とユニーク

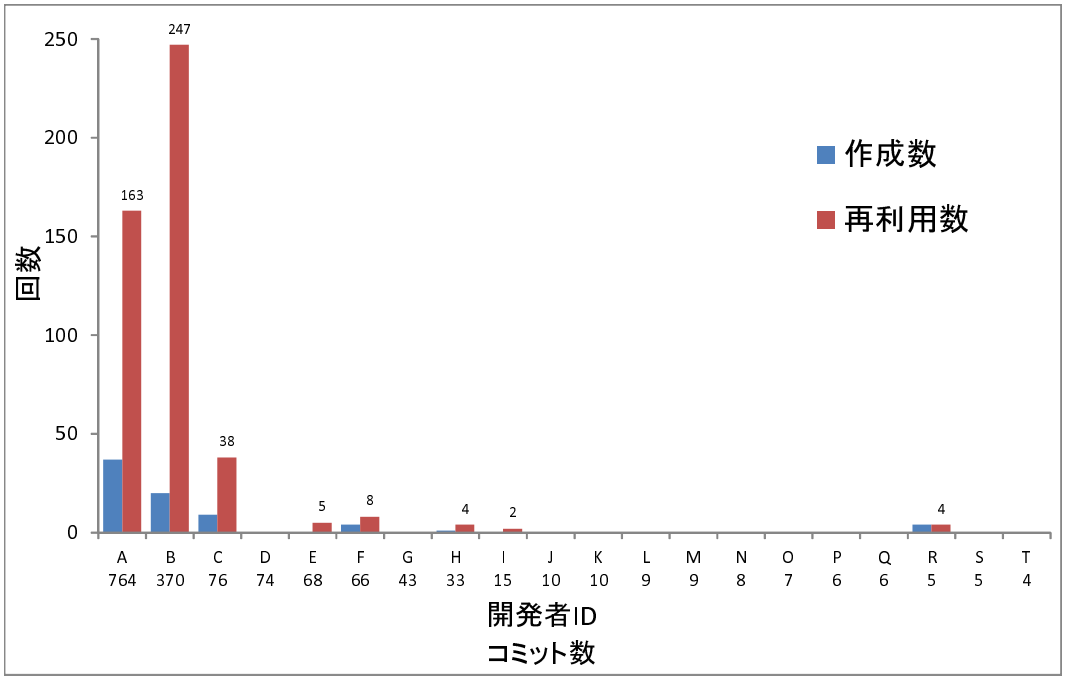


図 11: 開発者ごとのコードクローン作成数と再利用数

利用者数がともに多かったコードクローンの例を図 13 に，ユニーク利用者数の多いコードクローンの例を図 12 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 14 に示す．図 14 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 12 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 14 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．結果として，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く，汎用性が高いコードであると考えられる．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多少複雑になっているものが多い．また，分析内容 4 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から，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 5 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．

```
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 12: 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 13: 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 14: 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

6　むすび

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．実際に分析手法を実装し，2 つの Eclipse プロジェクトを用いて評価実験を行った．本実験により，コードクローンの作成者と利用者についての分析を行うことできた．分析の結果，コードクローンの利用数が極端に多い開発者や，コードクローンを作成も利用もしていない開発者がいるといったように，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，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再利用されやすいコード片についての比較を行うことで，再利用されやすいコード片の特徴を示した．

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，合計約 1500 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，2 つのプロジェクトをマージし，分析を行った．プロジェクト間クローン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マージし結果分析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

多くの開発者の分析情報を集めることにより，協調フィルタリングに基づいた再利用候補推薦ソフトウェアを開発することが可能となると考えられる．

謝辞

本研究において，常に適切な御指導およびご助言頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻ソフトウェア工学講座 井上克郎 教授に深く感謝いたします.

本研究において，随時適切な御指導およびご助言頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻ソフトウェア工学講座 松下誠 准教授に深く感謝いたします.

本研究の遂行ならびに本論文の執筆にあたり，貴重なお時間を割いて頂き，終始適切で熱心でかつ丁寧な御指導およびご助言頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻ソフトウェア設計学講座 井垣宏 特任准教授に深く感謝いたします.

本研究において，逐次適切な御指導およびご助言頂きました奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学講座 吉田則裕 助教に深く感謝いたします.

本研究において，適時適切な御指導およびご助言頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻ソフトウェア工学講座 石尾隆 助教に深く感謝いたします.

本研究を通して，様々な御指導およびご助言頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 山中裕樹 氏に深く感謝いたします.

最後に，苦楽を共にした大阪大学大学院情報科学研究科コンピュータサイエンス専攻ソフトウェア工学講座 井上研究室の皆様に深く感謝いたします.

参考文献

[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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] Siman. `http://www.harukizaemon.com/simian/`.

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

[11] Michael Godfrey and Qiang Tu. Tracking structural evolution using origin analysis. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pp. 117–119, USA, 2002. ACM.

[12] Miryung Kim, Vibha Sazawal, and David Notkin. An empirical study of code clone genealogies. In *13th ACM SIGSOFT international symposium on Foundations of software engineering*, 2005.

[13] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone Detection Using Abstract Syntax Trees. In *Proceedings of ICSM 1998*, pp. 368–377, 1998.

[14] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670, 2002.

[15] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Trans. Softw. Eng.*, Vol. 33, No. 9, pp. 577–591, 2007.

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

[17] Brenda S. Baker. A program for identifying duplicated code. In *Proceedings of Symposium on the Interface*, pp. 49–57, March 1992.

[18] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. *IEEE Trans. Softw. Eng.*, Vol. 32, No. 3, pp. 176–192, 2006.

[19] J. Howard Johnson. Identifying redundancy in source code using fingerprints. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1*, CASCON '93, pp. 171–183. IBM Press, 1993.

[20] Merlo Ettore, Antoniol Giuliano, Penta Massimiliano, Di, Rollo Vincenzo, Fabio, and Ecole Polytechnique De Montreal. Linear Complexity Object-Oriented Similarity for Clone Detection and Software Evolution Analyses. In *Proceedings of the 20th IEEE*

International Conference on Software Maintenance, pp. 412–416. Society Press, IEEE Computer Society Press, 2004.

[21] 門田暁人, 佐藤慎一, 神谷年洋, 松本健一. コードクローンに基づくレガシーソフトウェアの品質の分析 (システム評価・管理技術). 情報処理学会論文誌, Vol. 44, No. 8, pp. 2178–2188, aug 2003.

[22] 川口真司, 松下誠, 井上克郎. 版管理システムを用いたコードクローン履歴分析. 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, Vol. 105, No. 228, pp. 43–48, jul 2005.

[23] 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.