修士学位論文

題目

プロジェクト間のクローンの系譜に基づく開発者ごと再利用動向の分析

指導教員
井上 克郎 教授

報告者
森脇 匠哉

平成 27 年 2 月 6 日

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

平成 26 年度 修士学位論文

プロジェクト間のクローンの系譜に基づく開発者ごと再利用動向の分析

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た上で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動向の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．そこで本研究では，クローンの系譜検出ツール及び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分析を行うことで，開発者ごとのソースコードの再利用動向の分析を行う．

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．

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，またクローンセットにいつコードクローンが追加されたか，といった情報を得ることが出来る．

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，複数のプロジェクトの情報をひとつのプロジェクトにマージし，そのプロジェクトからクローンの系譜を導出する．得られたクローンの系譜の分析を行うことで，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 5 つの Java プロジェクトを用意し，開発者ごとの再利用動向の分析を行った．分析したプロジェクトは合わせて 14394 リビジョン，開発者は約 44 名の規模である．

分析の結果，再利用回数及び再利用される回数の極端に多い開発者や，コミット数は多いがほとんど再利用を実施していない開発者がいるといったように，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自分以外の開発者が書いたソースコードを特定及び分析し，その特徴を示した．

主な用語

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

目 次

1 まえがき **5**

2 背景 **7**

2.1 ソフトウェアの再利用 7

2.1.1 Code Authorship 8

2.1.2 Origin Analysis 9

2.2 版管理システム 11

2.3 コードクローン 12

2.3.1 発生原因 12

2.3.2 コードクローンの定義 13

2.3.3 Authorship 14

2.4 コードクローン変更管理 14

2.5 コードクローン及びクローンセットの分類 16

2.5.1 コードクローンの分類 16

2.5.2 クローンセットの分類 16

3 提案手法 **19**

3.1 概要 19

3.2 STEP1：リポジトリのマージ 19

3.2.1 入力 19

3.2.2 マージ手法 19

3.2.3 出力 20

3.3 STEP2：クローンの系譜の導出 20

3.3.1 入力 20

3.3.2 クローンの系譜導出手法 20

3.3.3 出力 21

3.4 STEP3：コードクローン作成者と利用者の特定 21

3.4.1 導出方法 21

3.4.2 出力 22

4 実装 **23**

4.1 データベース 28

4.2 ECTEC の PostgreSQL への対応 29

4.3 並列処理 29
4.4 開発者情報の取得 29

5 ケーススタディ **30**
5.1 概要 30
5.2 リポジトリの説明 30
5.3 実験環境 31
5.4 分析項目 31
5.4.1 分析結果 33
5.4.2 パフォーマンス 34

6 考察 **35**

7 むすび **36**

謝辞 **37**

参考文献 **38**

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　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 [11]．そのため，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 [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

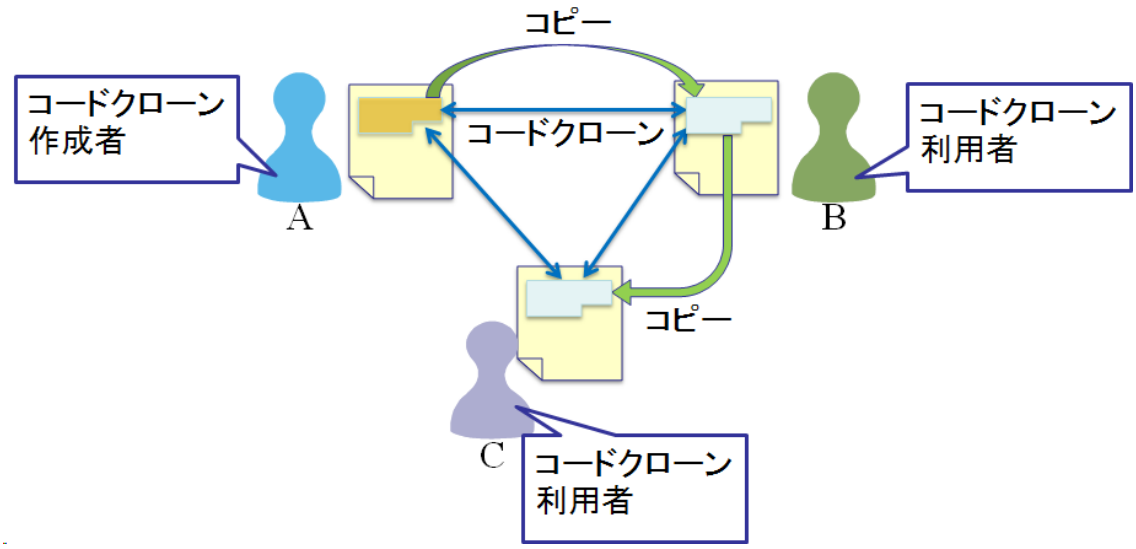


図 1: コードクローン利用者と作成者

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，再利用元のコード片を書いたのが誰であるか知ることができる．プログラムのソースコードが Subversion[2]や Git[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

[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://subversion.tigris.org/
[3] http://git-scm.com/

て Code Authorship を導出することができる．Subversion では blame コマンドを用いることで Code Authorship の取得を行う．図 2 に Code Authorship についての説明と blame コマンドでの出力例を示す.

図 2 (a) では，DeveloperA がリビジョン 1 においてコミットしたファイルにバグが見付かり，次リビジョンで 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) にはリビジョン 1 及びリビジョン 2 において blame コマンドを実行した場合のそれぞれの出力例を示す．例示したように，blame コマンドでの出力には Code Authorship とコード内容が含まれている．そして，リビジョン 2 における出力を見ると，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

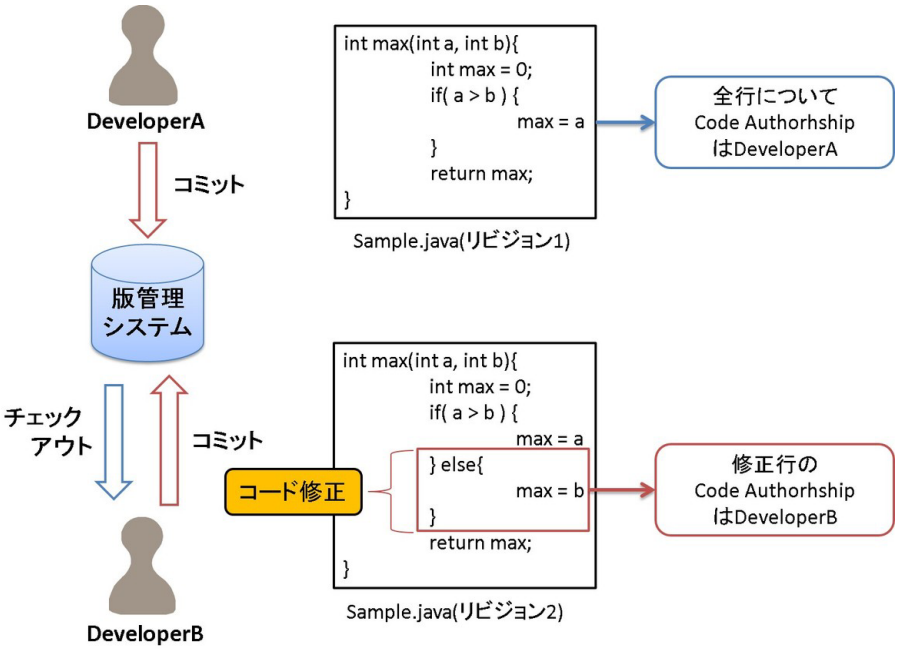
Michael Godfrey らは，あるバージョンで新しいソフトウェアエンティティが生成された際，そのソフトウェアエンティティが新たに出来たものであるか，あるいは以前のバージョンのものを再利用して作られたものかを判断する手法 (Origin Analysis) について研究を行っている [12]．Origin Analysis は，Entity Analysis と Relationship Analysis の二つの段階に分けられる．

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

```
$ svn blame Sample.java@r1
   1  DeveloperA     static int max(int a, int b){
   1  DeveloperA          int max = 0;
   1  DeveloperA          if( a > b ){
   1  DeveloperA               max = a;
   1  DeveloperA          }
   1  DeveloperA          return max;
   1  DeveloperA     }
```

```
$ svn blame Sample.java@r2
   1  DeveloperA     static int max(int a, int b){
   1  DeveloperA          int max = 0;
   1  DeveloperA          if( a > b ){
   1  DeveloperA               max = a;
   2  DeveloperB          }else{
   2  DeveloperB               max = b;
   2  DeveloperB          }
   1  DeveloperA          return max;
   1  DeveloperA     }
```

(b) blame コマンド実行

図 2: blame コマンド出力例

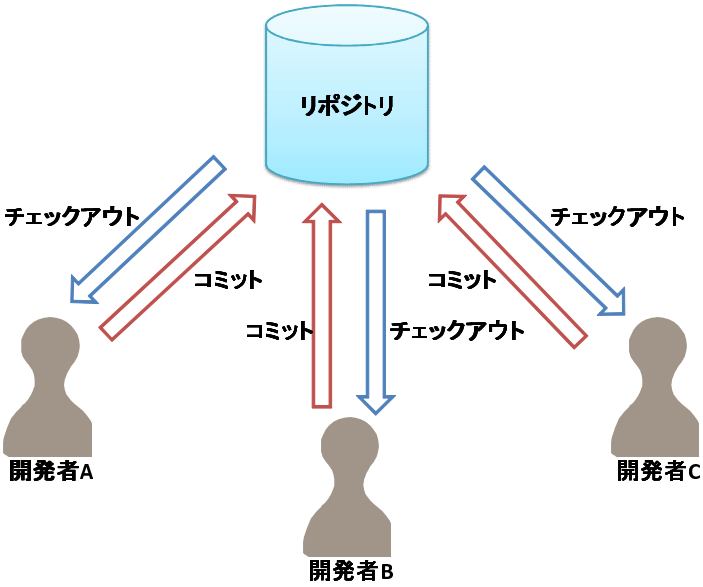


図 3: 版管理システム

2.2 版管理システム

版管理システムとは Apache Subversion，Git などの，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．

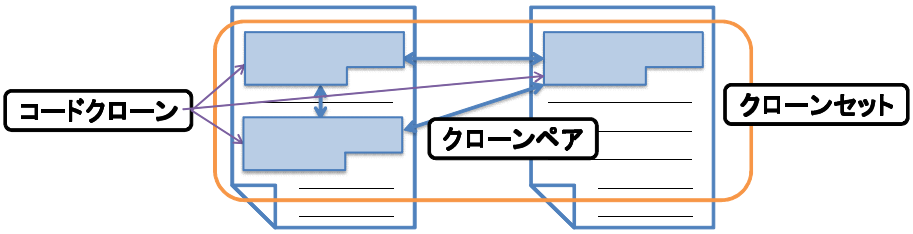


図 4: コードクローン

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 [13]．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 [14, 15]．

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 [16]．

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 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 [17]．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．

コードクローン作成者

あるクローンセットにおける Code Clone Authorship の内，ソースコードの実装日時が最も古いものが元コードを記述した開発者である．そのような開発者をコードクローン作成者と定義する．

コードクローン利用者

あるクローンセットにおける Code Clone Authorship の内，コードクローン作成者である Code Clone Authorship 以外のものは元のコード片を再利用した開発者である．そのような再利用を行い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．

2.4 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，クローンの系譜 (Code Clone Genealogies) を分析する手法を用いた研究を行っている [13]．クローンの系譜とは，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．

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 [18]．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，堀田らのクローンの系譜検出手法を参考にしている [19, 20, 21]．堀田らは，CRD(Clone Region Descriptors)[22] というコードクローンを構成するコード片の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．

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．

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．

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 [23] 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.

2.5 コードクローン及びクローンセットの分類

本研究では，再利用の特定を行うため，コードクローン及びクローンセットの分類を行う．なお，本研究では山中らの分類方法を参考にしている [24]. 表 1 に説明に用いる記号を示す.

2.5.1 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.

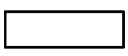
Deleted Clone

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.

2.5.2 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”，

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
			

“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．“New Clone Set”及び “Added Clone”を含む “Changed Clone Set”に着目することで，再利用の特定が可能である．

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．

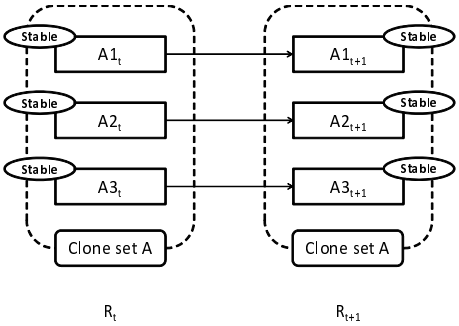
New Clone Set

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．

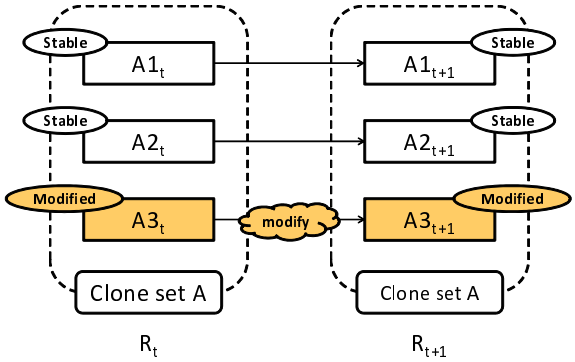
Deleted Clone Set

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．

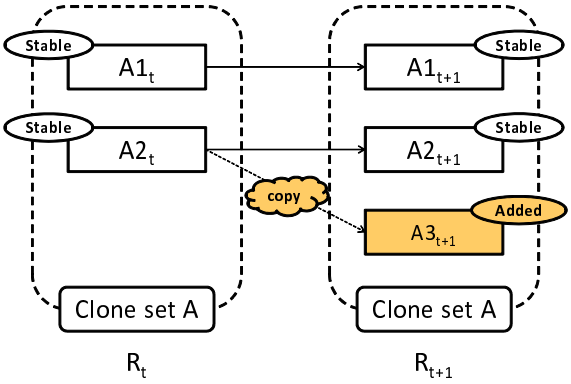
次節では，これらのコードクローン技術を応用し，開発者ごとの再利用動向を分析する手法について詳述する．



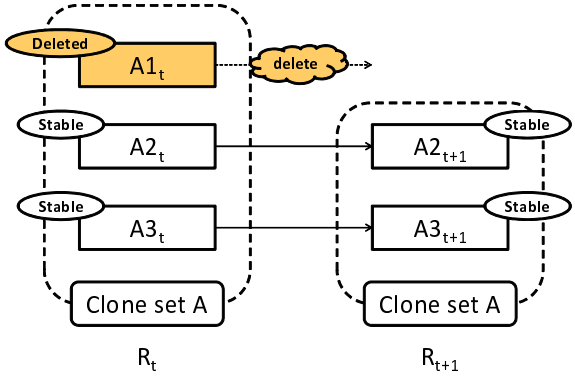
(a) Stable Clone Set



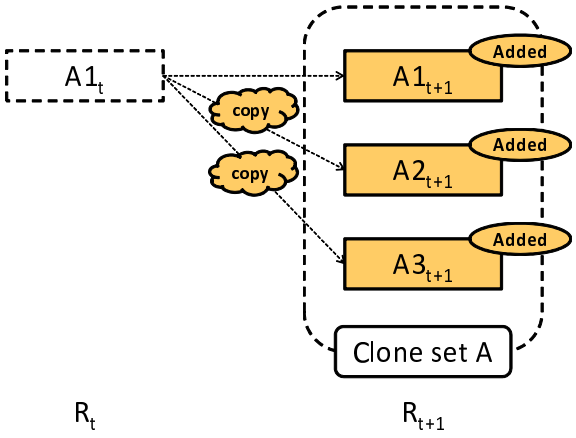
(b) Changed Clone Set [Modified]



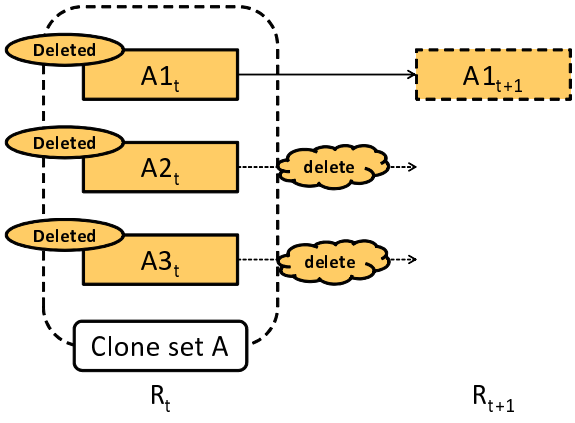
(c) Changed Clone Set [Added]



(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，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：**コードクローン作成者と利用者の特定**

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 6 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

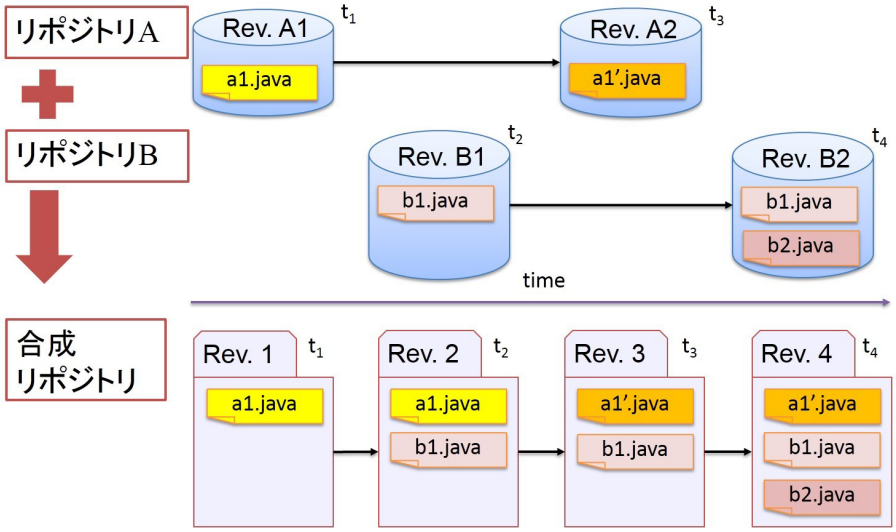


図 6: リポジトリのマージ手法

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．

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 2.4 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：コードクローン作成者と利用者の特定

あるクローンの系譜について，そのクローンの系譜の開始リビジョンにおけるクローンセットからコードクローンの作成者及び利用者を導出する．そして，そのクローンの系譜を時系列順に辿り，あるリビジョンにおいてコードクローンの追加があった際，そのコードクローンの Code Clone Authorship を求めコードクローン利用者を特定する．以上の処理を全クローンの系譜について行うことで，コードクローン作成者と利用者の特定を行う．

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を行うことで，再利用された回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．

3.4.1 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．

リビジョン R でコードクローンが新たに発生したとする．そのコードクローンの集合 C_r の内，最も実装日時が古いコード片の Code Clone Authorship である開発者がコードクローン作成者である．そして，それ以外のコード片の Code Clone Authorship がコードクローン利用者である．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ことができる．

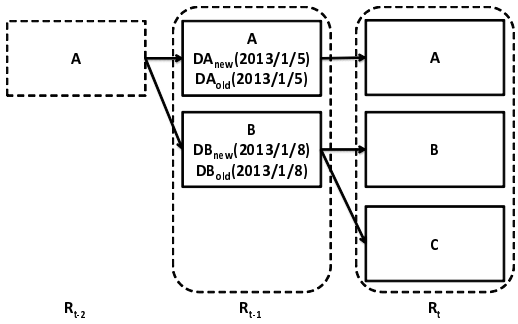
ここで，コードクローン A とコードクローン B が属するクローンセットにコードクローン C が追加されたと仮定して説明を行う．コードクローン A 内で最も古いコミット日時の行のコミット日時を DA_{old} と定義し，最も新しいコミット日時を DA_{new} とする．同様に，コードクローン B 内で最も古いコミット日時の行のコミット日時を DB_{old} ，最も新しいコミット日時を DB_{new} とする．

CASE1

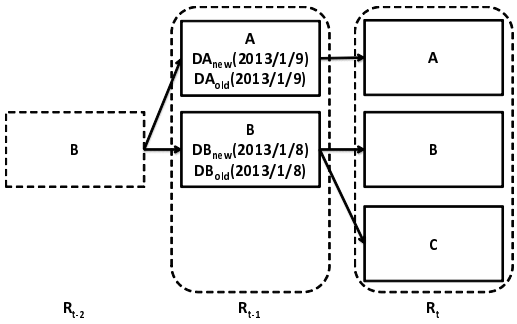
図 7 (a) に示すように，DA_{new} が DB_{old} より古い場合，コードクローン C の作成者はコードクローン A の Code Authorship である開発者である．

CASE2

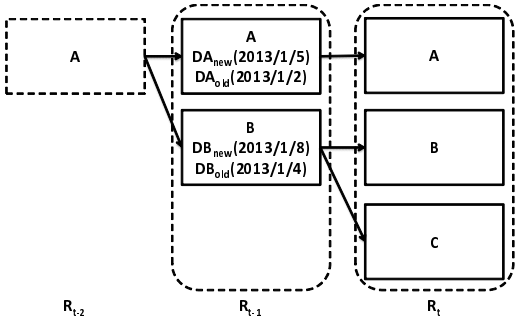
図 7 (b) に示すように，DB_{new} が DA_{old} より古い場合，コードクローン C の作成者はコードクローン B の Code Authorship である開発者である．



(a)CASE1



(b) CASE2



(c) CASE3

図 7: コードクローン作成者導出

CASE3

図 7 (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．

3.4.2 出力

全クローンの系譜におけるコードクローン作成者及び利用者の情報がデータベースに出力される．

4 実装

本研究では，提案手法についての実装を Java で行った．以下に開発環境を示す.

- CPU : Intel Xeon X5675 @3.07GHz 3.06GHz(2 プロセッサ)
- メモリ (RAM) : 48[GB]
- OS : Windows 7 Professional, Service Pack 1(64 ビット)
- JDK : 1.7.0_45
- Eclipse : 4.3.2
- PostgreSQL: 9.3

本プログラムは堀田の開発したツール ECTEC[4]をスケールしたものである．ECTEC とは，班管理システムの開発履歴情報を入力とし，コード片及びクローンの系譜を出力するシステムである.

本プログラムは，ECTEC と同様に Subversion 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クローンの系譜や Code Authorship を取得することで再利用情報の出力を行う．システムによって得られた再利用についての情報はデータベースに出力される.

[4] https://github.com/kusumotolab/ECTEC

表 2: REVISION テーブルのスキーマ

カラム名	型	備考
REVISION_ID	LONG	primary key
REVISION_IDENTIFIRE	TEXT	nor null, unique

表 3: COMBINED_REVISION テーブルのスキーマ

カラム名	型	備考
COMBINED_REVISION_ID	LONG	primary key
REVISION_ID	LONG	external key

表 4: VCS_COMMIT テーブルのスキーマ

カラム名	型	備考
VCS_COMMIT_ID	LONG	primary key
BEFORE_REVISION_ID	LONG	external key
AFTER_REVISION_ID	LONG	external key
BEFORE_REVISION_IDENTIFIRE	TEXT	
AFTER_REVISION_IDENTIFIRE	TEXT	
REPOSITORY_ID	LONG	
YEAR	INTEGER	
MONTH	INTEGER	
DAY	INTEGER	
MINUTE	INTEGER	
SECOND	INTEGER	

表 5: COMBINED_COMMIT テーブルのスキーマ

カラム名	型	備考
COMBINED_COMMIT_ID	LONG	primary key
BEFORE_COMBINED_REVISION_ID	LONG	
AFTER_COMBINED_REVISION_ID	LONG	
VCS_COMMIT_ID	LONG	external key

表 6: FILE テーブルのスキーマ

カラム名	型	備考
FILE_ID	LONG	primary key
FILE_PATH	TEXT	
REPOSITORY_ID	LONG	external key
START_COMBINED_REVISION_ID	LONG	external key
END_COMBINED_REVISION_ID	LONG	external key
ADDED_AT_START	INTEGER	
DELETED_AT_END	INTEGER	

表 7: CODE_FRAGMENT テーブルのスキーマ

カラム名	型	備考
CODE_FRAGMENT_ID	LONG	primary key
OWNER_FILE_ID	LONG	external key
OWNER_REPOSITORYID	LONG	external key
CRD_ID	LONG	external key
START_COMBINED_REVISION_ID	LONG	external key
END_COMBINED_REVISION_ID	LONG	exteral key
HASH	LONG	
HASH_FOR_CLONE	LONG	
START_LINE	INTEGER	> 0
END_LINE	INTEGER	> 0
SIZE	INTEGER	> 0
FILE_ADDED_AT_START	INTEGER	
FILE_DELETED_AT_END	INTEGER	

表 8: CODE_FRAGMENT_LINK テーブルのスキーマ

カラム名	型	備考
CODE_FRAGMENT_LINK_ID	LONG	primary key
BEFORE_ELEMENT_ID	LONG	external key
AFTER_ELEMENT_ID	LONG	external key
BEFORE_COMBINED_REVISION_ID	LONG	external key
AFTER_COMBINED_REVISION_ID	LONG	exteral key

表 9: CLONE_SET テーブルのスキーマ

カラム名	型	備考
CLONE_SET_ID	LONG	primary key
OWNER_COMBINED_REVISION_ID	LONG	external key
ELEMENT	LONG	

表 10: CLONE_SET_LINK テーブルのスキーマ

カラム名	型	備考
CLONE_SET_LINK_ID	LONG	primary key
BEFORE_ELEMENT_ID	LONG	external key
AFTER_ELEMENT_ID	LONG	external key
BEFORE_COMBINED_REVISION_ID	LONG	external key
AFTER_COMBINED_REVISION_ID	LONG	exteral key

表 11: CLONE_SET_LINK_FRAGMENT_LINK テーブルのスキーマ

カラム名	型	備考
CLONE_SET_LINK_ID	LONG	primary key
CODE_FRAGMENT_LINK_ID	LONG	primary key

表 12: CLONE_GENEALOGY テーブルのスキーマ

カラム名	型	備考
CLONE_GENEALOG_ID	LONG	primary key
START_COOMBINED_REVISION_ID	LONG	external key
END_COMBINED_REVISION_ID	LONG	exteral key

表 13: CLONE_GENEALOGY_ELEMENT テーブルのスキーマ

カラム名	型	備考
CLONE_GENEALOGY_ELEMENT_ID	LONG	primary key
CLONESET_ID	LONG	primary key

表 14: CLONE_GENEALOGY_LINK_ELEMENT テーブルのスキーマ

カラム名	型	備考
CLONE_GENEALOGY_LINK_ELEMENT_ID	LONG	primary key
CLONE_SET_LINK_ID	LONG	exteral key

表 15: CRD テーブルのスキーマ

カラム名	型	備考
CRD_ID	LONG	primary key
TYPE	TEXT	not null
HEAD	TEXT	not null
ANCHOR	TEXT	not null
NORMALIZED_ANCHOR	TEXT	not null
CM	INTEGER	>0
ANCESTORS	TEXT*	not null
FULL_TEXT	TEXT	not null

表 16: DEVELOPER テーブルのスキーマ

カラム名	型	備考
DEVELOPER_ID	LONG	primary key
NAME	TEXT	
HEAD	TEXT	not null
REPOSITORY_ID	LONG	external key
COMMIT	LONG	

表 17: AUTHOR テーブルのスキーマ

カラム名	型	備考
AUTHOR_ID	LONG	primary key
CLONE_GENEALOGY_ID	LONG	external key
COMBINED_REBVISION_ID	LONG	external key
DEVELOPER_ID	LONG	external key

表 18: REUSE テーブルのスキーマ

カラム名	型	備考
REUSE_ID	LONG	primary key
CLONE_GENEALOGY_ID	LONG	external key
CODE_GENEALOGY_ID	LONG	external key
REUSED_CODE_GENEALOGY_ID	LONG	external key
COMBINED_REBVISION_ID	LONG	external key
DEVELOPER_ID	LONG	external key

4.1 データベース

本研究では，以下のデータベーステーブルを定義し，使用する．

- REVISION
- COMBINED_REVISION
- VCS_COMMIT
- COMBINED_COMMIT
- FILE
- CODE_FRAGMENT
- CODE_FRAGMENT_LINK
- CLONE_SET
- CLONE_SET_LINK
- CLONE_SET_LINK_FRAGMENT_LINK
- CLONE_GENEALOGY
- CLONE_GENEALOGY_ELEMENT
- CLONE_GENEALOGY_LINK_ELEMENT
- CRD
- DEVELOPER
- AUTHOR
- REUSE

表 2 から表 18 に各テーブルのスキーマを示す．それぞれのテーブルは LONG 型の ID を主キーとしている．

以下，実装についての説明を行う．

4.2 ECTEC の PostgreSQL への対応

ECTEC では分析によって得られた結果を格納するデータベースに sqlite を利用している．本研究では，PostgreSQL に対応させることによって，複数の仮想マシンからの同時データベースアクセスを可能とした．

4.3 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，再利用情報の導出において，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．

4.4 開発者情報の取得

本実験では，Subversion のコマンドを用いて開発者情報の取得を行った．以下，実装した Subversion コマンドの機能を示す．なお，コマンドの実装には SVNKit[5]を用いた．

- svn log
  リポジトリのログ情報を取得する．
- svn blame
  ソースコードの各行の Code Authorship とコミットリビジョンを取得する．

[5] http://svnkit.com/javadoc/index.html

5　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，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．

5.1　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，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．

5.2　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 19 に示す．なお，表 19 の情報は 2015 年 1 月 31 日時点のものである．

表 19: 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/omegat/`
[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[6]から選定した．benten と mergedoc には共通の開発者 A が存在し，itext と mailmarket について，共通の開発者 B 及び C 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．

[6] `http://sourceforge.jp/projects/mergedoc/`

5.3 実験環境

本ケーススタディでは，AWS(Amazon Web Services)[7]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 20 に示す.

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.

表 21 に実際に仮想マシンがどのリビジョンに対して処理を行ったかを示す.

5.4 分析項目

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 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
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

表 20: 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

[7] http://aws.amazon.com/jp/

表 21: 仮想マシンへの分割単位の割当て

仮想マシン名	合成リビジョンの割当て [開始-終了]	クローンの系譜の割当て [開始-終了]
worker00	1-560	1-179
worker01	561-1020	180-359
worker02	1121-1680	360-539
worker03	1681-2240	540-719
worker04	2241-2800	720-899
worker05	2801-3360	900-1079
worker06	3361-3920	1080-1259
worker07	3921-4260	1260-1438
worker08	4261-4600	1439-1618
worker09	4601-4940	1619-1798
worker10	4941-5280	1799-1978
worker11	5281-5620	1979-2158
worker12	5621-5960	2159-2338
worker13	5961-6300	2339-2518
worker14	6301-6640	2519-2698
worker15	6641-6865	2699-2877
worker16	6866-7090	2878-3057
worker17	7091-7315	3058-3237
worker18	7316-7540	3238-3417
worker19	7541-7765	3418-3597
worker20	7766-7990	3598-3777
worker21	7991-8215	3778-3957
worker22	8216-8440	3958-4136
worker23	8441-8665	4137-4316
worker24	8666-8890	4137-4316
worker25	8891-9115	4497-4676
worker26	9116-9340	4677-4856
worker27	9341-9565	4857-5036
worker28	9566-9790	5037-5216
worker29	9791-10009	5217-5396

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．

5.4.1 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 8 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 9 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．そして，10 個すべての

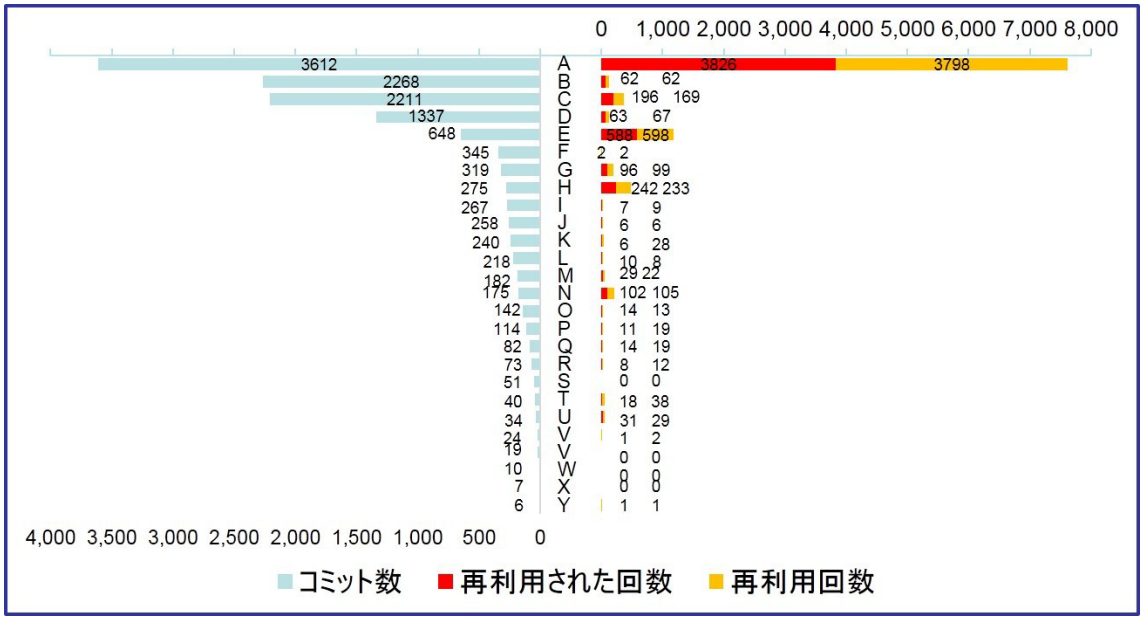


図 8: 再開発者ごとのコードクローン作成数と再利用数

再利用について，両プロジェクトに所属する開発者がコードクローン作成者であった．また．9 個の再利用について，両プロジェクトに所属する開発者がコードクローン利用者であった．

5.4.2 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 22 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 22: 処理時間

処理	プログラム実行 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

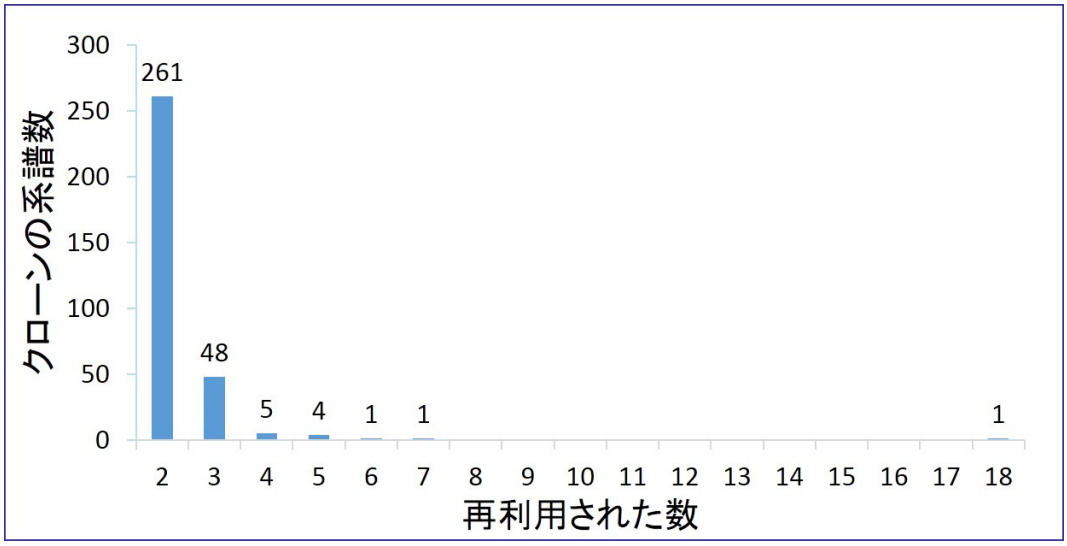


図 9: クローンの系譜の被再利用回数の分布

6 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．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が分かった．そのため，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 について，再利用は自分のソースコードによって行うことが多く，再利用を頻繁に行う開発者は再利用されやすいということが分かった．また，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．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．

7　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，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，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．まら，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．いつも優しくお声掛け頂けたこと励みになりました．

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．スライド及び論文の推敲，実験環境構築についての助言，AWS の利用環境設定など，本当にお世話になりました．

本研究において，適切な御指導およびご助言頂きました名古屋大学大学院情報科学研究科情報システム学専攻 吉田則裕 准教授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 堀田圭佑 氏に深く感謝いたします．堀田氏の助言がなければツールのスケールはできていませんでした．

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及びソフトウェア設計学講座楠本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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.