修士論文

メソッド単位クローンの消失パターンに基づく分類と考察

常木　健介

2019 年 3 月 15 日

奈良先端科学技術大学院大学
情報科学研究科 情報科学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士（工学）授与の要件として提出した修士論文である．

常木　健介

審査委員：

飯田　元 教授　　（主指導教員）
松本　健一 教授　（副指導教員）
市川 昊平 准教授　（副指導教員）
崔 恩瀞　助教　　（副指導教員）

メソッド単位クローンの消失パターンに基づく分類と考察*

常木　健介

内容梗概

コードクローンとは，ソースコード中に存在する同一または類似した部分をもつコード片のことを指す．主に，開発者が行うコピーアンドペーストによって生成される．コピーアンドペーストによるコード片の複製は，欠陥の発生や拡散につながる可能性があるため，コードクローンの存在はソフトウェアの保守性を低下させるひとつの要因であると言われている．そのため，コードクローンがソフトウェア開発に与える影響を分析する必要がある．その手法の一つとして，コードクローンの履歴情報を分析する方法が挙げられる．コードクローンの履歴情報を分析することで，でコードクローンがどのように作成，変更，消失したのかといった情報を得ることができる．しかしながら，既存研究ではどのようにコードクローンが消失したのかは考慮していない．コードクローンの消失内容を分析することで，無駄なコードクローンの生成を回避することが期待できる．そこで，本研究ではコードクローンが履歴上で消失する際のパターンをクローンセット内のクローン片の変更内容の差分結果を用いてクローンセット消失パターンとして定義し，実プロジェクトのコードクローン履歴上での各消失パターンの出現頻度を調査した．実際に 8 つのクローンセット消失パターンが確認でき，消失パターンの出現頻度からプロジェクトの開発傾向の考察を実施した．例えば，全てのクローン片が削除される Delete パターンが頻発するプロジェクトでは計画的ではないコード片の生成ポリシーの元，開発されている可能性が高いと議論できる．本手法で提案した，クローンセット消失パターンを用いることで，コードクローン履歴をより詳細に分析することが可能である．

*奈良先端科学技術大学院大学 情報科学研究科 情報科学専攻 修士論文,
2019 年 3 月 15 日.

キーワード
コードクローン，ソフトウェアリポジトリマイニング，オープンソースソフトウェア

Classification and Consideration based on the Delete Pattern of Method Code Clone*

Kensuke Tsuneki

Abstract

The code clone is a code fragment having the other code fragment which is similar or match in the source code. Mainly cause of gererate code clone is the copy and paste operation by the developer. It is said that the existence of code clones is one ot the factor that decreases the maintainability of software, because there is a possibility of causing defects and spreading. Therefore, it is necessary to analyze the influence of code clone on software development. One of the method is analyzing history information of code clones. By analyzing the history information of the code clones, it is possible to obtain information on how the code clone is created, changed or deleted. In this research, we analyze histories of code clone in 10 open source projects, and defined delete pattern of code clones. In addition, we disucussed development policy of the project using delete pattern of code clones.

Keywords:

Code Clone,Mining Software Repositories,Open Source Software

*Master's Thesis, Department of Information Science, Graduate School of Information Science, Nara Institute of Science and Technology, March 15, 2019.

目次

図目次

表目次

第 1 章　はじめに

コードクローンとは，ソースコード中に存在する互いに一致，または類似したソースコードの断片を指し，主にソフトウェア開発者によって頻繁に実施されるコピー＆ペーストによって発生する [1].

一般的に，コードクローンの存在は，ソフトウェアの品質に悪影響を与えると言われている. たとえば，コピー元のコード断片に，欠陥が混入していた場合，コードクローン関係にあるすべてのコード断片にも同様の欠陥が含まれている可能性が高く，開発者はすべてのコードクローンに対して，修正の是非を検討する必要がある．

しかし，大規模なソフトウェアの場合，手作業で全てのコードクローンを見つけることは困難である．そのため，コードクローンを効率的に検出するためのツールが数多く開発されている．[2, 3, 4]. また，コードクローンの検出だけでなく，検出したコードクローンがソフトウェア開発にどのような影響を与えているのかを分析する必要がある．

コードクローンがソフトウェア開発に与える影響を分析するための手法として，コードクローンの履歴情報を追跡・分析する方法が提案されている [5, 6]. コードクローンの履歴情報を分析することで，コードクローンがどのように作成され，どのように変更，削除されたのかを明らかにすることができる．

Kim らは，バージョン管理システム上で記録されたコードクローンの変更パターンを Clone Genealogy と名付け，その分析手法を提案している [5].

Uemura らは，コードクローン検出時の粒度の 1 つであるメソッド単位のコードクローンに着目し，コミット単位のコードクローンの履歴抽出手法を提案し，メソッド単位のコードクローンについての定量的な調査を実施した [6]．結果として，コードクローンの履歴上で変更のみが実施されるコードクローンは少なく，最終的に全体の半数のコードクローンが履歴上で削除される傾向にある，と報告している．しかしながら，コードクローンが履歴上でどのような変更パターンで削除され，どのように消失したのかは分類されていない．そこで，本研究ではコードクローンが履歴上で消失する際のパターンをクローンセット内のクローン片の変更内容の差分結果を用いてクローンセット消失パターンとして定義し，実プロジェクトのコードクローン履歴上での各消失パターンの出現頻度を調査した．定義した 14 パターン

のクローンセット消失パターンの内，実際の開発プロジェクト上で観測できたのは 8 つであった．そこで，実際に出現した 8 つのクローンセットパターンの頻度の傾向からプロジェクトの開発傾向の考察を実施した．例えば，全てのクローン片が削除される Delete パターンが頻発するプロジェクトでは計画的ではないコード片の生成ポリシーの元，開発されている可能性が高いと議論できる．本手法で提案した，クローンセット消失パターンを用いることで，コードクローン履歴をより詳細に分析することが可能である．

本論文の構成について述べる．まず，2 章でコードクローンに関連する用語及び関連研究の説明を行い，3 章ではコードクローンの履歴情報を用いたコードクローンの調査手法について述べる．4 章では，調査結果及びその考察内容について述べ，5 章ではまとめと今後の課題について述べる．

第 2 章　背景

本章では，コードクローンに関連する用語及び関連研究の説明を行い，研究の背景，目的について述べる．2.1 節では，コードクローンについて述べる．2.2 節では既存のコードクローンに関する調査，2.3 節では，コードクローンの履歴分析手法について述べる．さらに，本研究の調査で用いるメソッド単位のコードクローンの履歴情報の分析手法及び関連研究について述べたうえで，本研究の目的を述べる．

2.1　コードクローンとは

コードクローンとは，ソースコード中に存在する，互いに一致または類似したコード片のことを指す [1]．互いに一致あるいは類似するクローン片の集合をクローンセットと呼ぶ．主にソフトウェア開発者によって実行されるコピーアンドペーストによって発生する．コピーアンドペーストによるコード片の複製は，類似機能を実装するために，ソフトウェア開発において，頻繁に実施される．

一般的に，コードクローンの存在は，ソフトウェアの保守性を低下させる一つの要因であるとされている．たとえば，あるコード片に欠陥が含まれていた場合，開発者はそのコード片とコードクローン関係にあるすべてのコード片に対して修正の是非を検討しなければならず，保守作業量の増加や，修正すべき欠陥の見逃しが増加する可能性が高い．そのため，コードクローンを適切に管理する手法が求められる．

2.1.1　コードクローンの分類

コードクローンは類似度の度合いによって，以下のように分類される [7]．

タイプ 1
: 空白やタブの有無、括弧の位置，コメントなどのコーディングスタイルごとに異なる記述を除き、完全に一致するコードクローン．

タイプ 2

変数名や関数名などのユーザー定義部分、変数の型や識別子などの一部の予約語のみが異なるコードクローン.

タイプ３

タイプ 2 における差異に加え、文の挿入や削除、変更が行われることによってコード片の一部に一致しない部分を持つコードクローン.

また，近年ではタイプ 1-3 のほかに，タイプ 4 のコードクローンについても検出及び調査が実施されている [4]．タイプ 4 のコードクローンとは，ソースコードの構文上の実装は異なっているが，類似処理を実行しているコードクローンのことである．タイプ 4 のコードクローンは処理内容の類似性に着目しているため，ソースコードの構文や，文法の類似度によって算出されるタイプ 1 - 3 とは区別される場合がある．

2.1.2 コードクローンの検出

大規模なソフトウェアの場合，ソースコード中に存在するすべてのコードクローンを手作業で見つけることは困難である．そのため，効率的にコードクローンを検出するためのツールが数多く開発されている [2, 4, 8]．また，コードクローンは様々な粒度で作成されるため，主にファイル単位，メソッド単位，ブロック単位，行単位のコードクローンに分類されたうえで，検出や調査が行われている．

代表的なコードクローンの検出ツールとして Kamiya らが開発した CCFinder[2] が挙げられる．CCFinder はソースコード中の字句をトークンに変換し，そのトークンの並びが一致するコード片又は類似したコード片をコードクローンとして自動的に検出することができる．行単位のコードクローンに着目し，変数名や関数名などのユーザー定義部分をトークン化するため，タイプ 1 だけでなく，タイプ 2 のコードクローンも検出することが可能である．

また，別の検出手法を用いた検出ツールとして，CloneDetector[4] が挙げられる．ソースコード中の字句を特徴ベクトルに変換し，その類似度を用いてコードクローンを検出する．この手法では，ソースコード中のなどで囲まれたコードブロックに着目し，ブロック単位のコードクローンを検出対象とすることで，タイプ 4 までのコードクローン検出に対応している．

2.2　コードクローンに関する調査

コードクローンを効率的に検出するために多くのコードクローン検出ツールが開発されたため，コードクローンの情報を自動かつ高速に収集することが可能である．そのため，検出したコードクローンを分析し，コードクローンがソフトウェア開発に与える影響を分析した研究がこれまでに数多く行われている．たとえば，危険度の高い欠陥の混入とコードクローンとの関係を調査を実施した研究が存在する [9, 10].

中山らは，コードクローンの位置関係と欠陥混入傾向の関係性を調査し，コードクローン近傍のコード片に多くの欠陥が混入している傾向を示した [9].

Mondal らは，コードクローンに対する欠陥修正作業とコードクローンのタイプに着目し，３つのタイプ毎に欠陥修正の重要度が異なるという特徴を発見し，タイプ３のコードクローンが一番欠陥が含まれる可能性が高く，優先的に保守対象として注目すべきだと述べている [10].

2.3　コードクローンの履歴分析

コードクローンがソフトウェア開発において与える影響を詳細に分析するために，コードクローンがどのように作成され，どのように変更，削除されているかといったコードクローンの履歴情報を調査することが重要である．これまでに，コードクローンの履歴を追跡し，分析するための手法がいくつか提案されている [5].

Kim らは，バージョン管理システム上で記録されたコードクローンの変更パターンを Clone Genealogy と名付け，その追跡・分析手法を提案している [5]．Kim らの手法では，コードクローンを２つのバージョン毎に検出し，検出されたコードクローンから，同一のものを探索し，バージョン毎のコードクローンの変更を追跡することで，コードクローンの履歴情報の作成を実現している．

2.3.1　メソッド単位のコードクローンの履歴情報の追跡・分析手法

Uemura らによって，より軽量なコードクローンの履歴分析手法が提案されている [6]．Uemura らは，コードクローン検出の粒度としてメソッド単位のコードクローンに着目し，コードクローンの履歴の追跡及び分析を行った．その際に,Hata らによって提案された構文情報リポジトリである，Historage[11] を用いている．通常，Git などのバージョン管理システムのリポジトリでは，ソースコード内容をファイル単位でのみ記録している．Historage では，ソースコードの構文情報を解析し，クラスとメソッドに細分化し，クラス情報や名前空間をディレクトリシステム上で表現し，1 つのメソッドを，1 つのファイルに対応付けて記録している．そのため，通常の Git システムよりも，細かい粒度でメソッドなどの差分情報を検出することができる．Uemura らはこの Historage のシステムを拡張し，コードクローンの履歴を考慮したメソッド単位のコードクローン検出を行った．

結果として，コードクローンの履歴上で変更のみが実施されるコードクローンは少なく，最終的に全体の半数のコードクローンが履歴上で削除される傾向にある，と報告している．しかしながら，コードクローンが履歴上でどのような変更パターンで削除され，どのような削除内容，消失パターンであるかまでは明らかにされていない．

2.3.2　クローンセットの消失

既存のコードクローン履歴の調査では，多数のクローンセット（同じクローンの集合）が最終的に消失していると報告されている．既存研究のコードクローン履歴分析手法では，前後のリビジョンのクローンセットを比較し直前のリビジョンで検出されていたクローンセット（同じクローンの集合）が次のリビジョンで検出されなくなった場合に，クローンセットが消失したと定義している．

実際にはクローンセットが消失する原因は複数考えられるが，既存研究では考慮されていない．例えば，クローンセット内のクローン片すべてが同時に削除された場合や，リファクタリング操作によって削除された場合，クローン片が個別に改変（一貫性のない変更）されたことによってクローンでなくなった可能性などが考えられる．

コードクローンの履歴を分析する際に，コードクローンがどのように消失したのかを明らかにすることができれば，無駄なクローンの生成を回避することができるため，コードクローンがソフトウェアに与える悪影響を低下させ，ソフトウェアの開発・保守効率の向上につながることが期待できる．そこで，本研究ではコードクローンが履歴上でどのように消失したのかを既存の開発プロジェクトのコードクローン履歴から分析し，どのようにコードクローンが消失したのかを明らかにする．

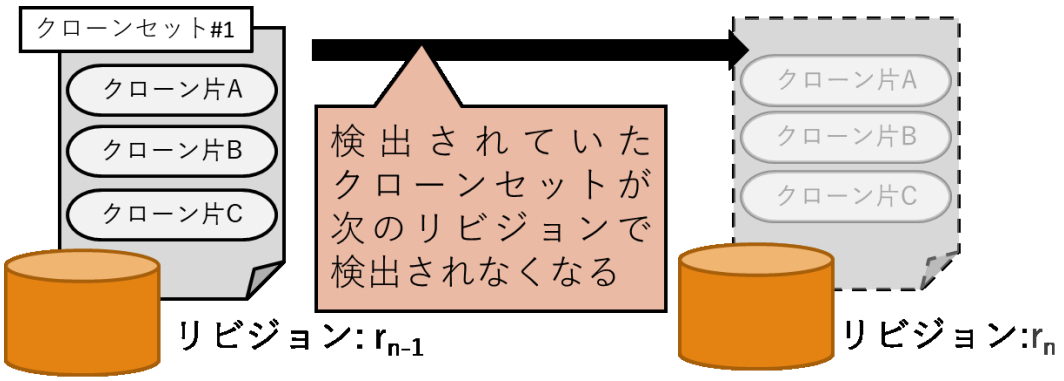

図 2.1　コードクローンの履歴上でのクローンセットの消失

第 3 章　調査手法

コードクローンが履歴上でどのように消失したのかを明らかにするために，コードクローンが履歴上で消失する際のパターンをクローンセット内の変更内容の差分結果から定義し，実プロジェクトのクローン履歴上での各消失パターンの出現頻度を調査し，分析を実施した．本章では，クローンの履歴情報から，コードクローンが消失した際の消失パターンの算出する手法と定義，分析する際の調査手順について述べる．3.1 節では，分析対象のデータセットの詳細について述べ，3.2 節ではクローンセット消失パターンの定義手法について説明し，3.3 節では調査手順，調査対象プロジェクトの詳細について述べる．

3.1　分析対象のデータセット

本研究では，Uemura らが作成したコードクローンの履歴情報が格納されているデータセットを基に，コードクローンの履歴情報の分析を実施した [6]．データセットは，JSON 形式のデータで記述されており，辞書形式のデータが格納されている．以下にそのデータセットの内容を示す．

cloneset：　クローンセットに関連するデータの集合

- fragment_hex：トークン化されたコード片のハッシュ値
- historage_commit_hex：Historage に変換された，git リポジトリのコミットのハッシュ値
- originarl_commit_hex：git リポジトリのコミットのハッシュ値
- fragments：クローンセット内に含まれるクローン片のリスト
- diff：file-diff の結果を参照するためのキー情報
- prev：直前のコミットでのクローンセット情報 (空の場合はそのコミットで作られたコードクローン)
- next：次のコミット上でのクローンセット情報 (空の場合はそのコミットで削除されたコードクローン)

commit： コミットに関するデータの集合

- historage_commit_hex： Historage に変換された，git リポジトリのコミットハッシュ値
- original_commit_hex： Historage 変換前の git リポジトリのコミットのハッシュ値
- prev：直前のコミットを参照するためのキー情報
- next：次のコミットを参照するためのキー情報

file-diff： git-diff によって抽出された差分データの集合

- mode：add、delete、modify、move 等の git-diff の差分情報
- diff-historage_commit_hex: コミットに対する全変更を検出した際の記述内容
- diff-cloneset_fragment_hex-historage_commit_hex:クローンセットに対する変更

各リビジョン毎のクローンセットと前後のリビジョン情報を比較することで，コードクローン履歴の分析することが可能である．

3.2 クローンセット消失パターン

クローンセットがどのようにコードクローンの履歴上で消失したのかを分析するために，履歴上でクローンセットが消失したタイミングで，クローンセット内のクローン片に適用された変更内容の記録を分析し消失パターンを定義する．

3.2.1 クローン片の変更履歴の差分検出手法

クローンセットが消失する際には，同じクローンセットに属するクローン片に対して何らかの操作が適用されることで，クローン関係でなくなる．そのため，クローンセットが消失したタイミングでクローンセット内のクローン片に実施された変更内容を分析することでクローンセットがどのように消失したのかを明確化でき

る．本研究で用いるコードクローン履歴分析手法は，バージョン管理システム git のリポジトリシステムで管理されているため，各クローン片の変更内容は git-diff アルゴリズムを利用することで変更内容の差分結果を用いて検出可能である．本研究では，メソッドレベルに細分化されたリポジトリを利用するため，調査するコードクローンの粒度はメソッドレベルに限定する．以下に git-diff を用いたクローン片の変更内容の差分結果の一覧を示す．

Add
: 前後のコミットを比較した際に，クローン片（メソッド）が新規作成された場合

Delete
: 前後のコミットを比較した際に，クローン片が存在していなかった場合

Modify
: 前後のコミット間で，修正が実施された場合（クローン片として検出される範囲内の修正）

Move
: 前後のコミット間で，名称の変更，メソッドの移動と修正が実施された場合

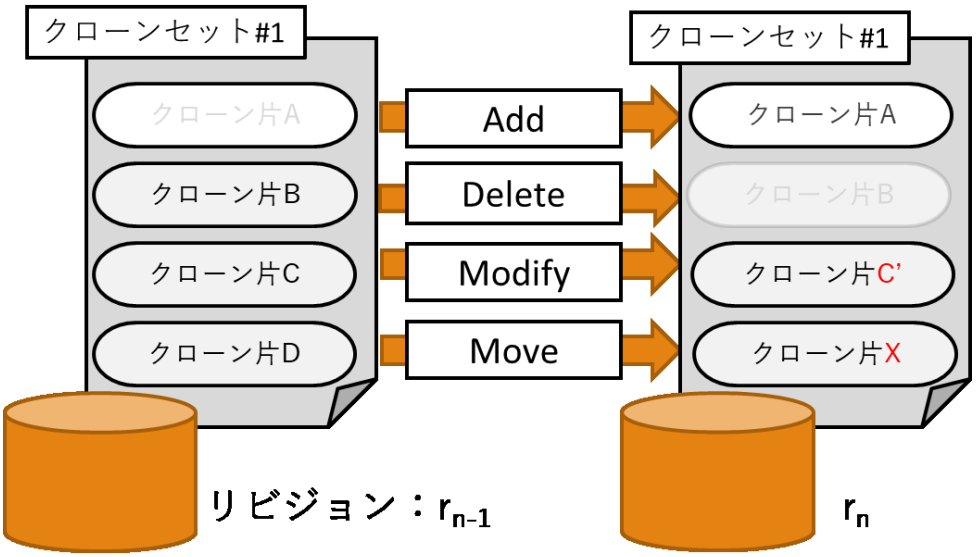

図 3.1　git-diff によるクローンセット内の各クローン片の変更内容の検出

本研究では，クローンの履歴上で消失したクローンセットにのみ着目するため，消失に関連のある,Dlete,Change(git-diff では Modify と識別),Move の分類結果を利用する．これらに加え，前後のコミット間で変更操作が実施されなかった場合を NotChange として記録し，4 操作の組み合わせに基づいて，クローンセットが消失した際のパターンを定義した．

3.2.2　クローンセット消失パターンの定義

クローンセットが消失した際のクローンセット内の各クローン片の git-diff の検出結果から，クローンセット消失パターンを定義する．クローンセット消失パターンの算出例を図 3.2 に示す．例えば，クローンセット内のクローン片が全て削除 (Delete) された場合は，Delete 操作のみの組み合わせによってクローンセットが消失したことになる．そのため，クローンセット消失パターン D(Delete パターン) として分類する．他の操作の組み合わせも同様にすべて分類し，クローンセット消失パターンとして定義する．

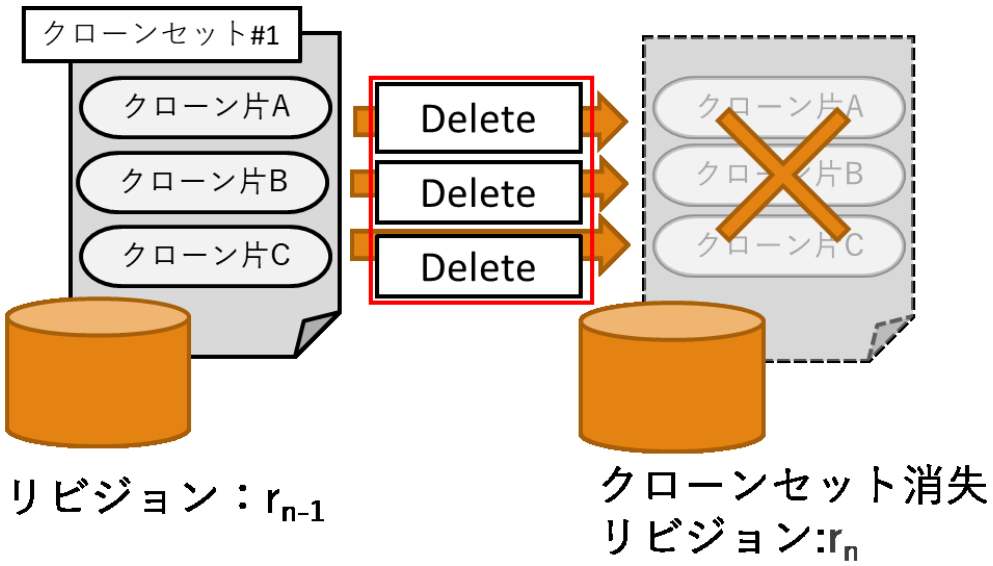

図 3.2　クローンセット消失パターンの算出例

消失パターン						
1種類	D	M	C			
2種類	D+C	D+M	D+N	C+M	C+N	M+N
3種類	D+C+M	D+C+N	D+M+N	C+M+N		
4種類	D+C+M+N					

図 3.3　クローンセット消失パターン

クローン片毎の変更内容の組み合わせを用いて消失パターンを定義した結果，図 3.3 に示す 14 パターンの組み合わせが考えられる．このクローンセット消失パターンを用いて，実際の開発プロジェクトの各消失パターンの出現頻度を調査した．

3.3 調査手順

本節では，3.1 節で述べたデータセットを利用し，クローンセットの消失パターンを抽出するための調査手順について述べる．

手順 1 履歴中で消失したクローンセットの抽出

JSON 形式で記述されたデータセットには，履歴上でクローンセットが検出された際のすべてのコミット情報，クローンセットの周辺情報，差分情報が格納されている．本研究ではクローンセットの消失時に着目して調査を行うため，履歴上で消失したクローンセットのみを抽出する必要がある．データセットには，クローンセットが消失したこどが認識できる識別子が付与されているため，その識別子が付与されたクローンセットのキー情報を抜き出すことで，履歴上で消失したクローンセットを抽出することができる．

手順 2 消失クローンセット内のクローン片の変更パターンの抽出

手順 1 で抽出したクローンセットが消失したコミットから，3.2.1 節で述べた手法を用いてクローン片毎の差分結果をデータセットから参照し，履歴上でクローン片ごとに実施された変更パターンを抽出する．

手順 3 クローンセットの消失パターンの判定

手順 2 で抽出したクローンセット内のクローン片に対して実施された変更パターンの分類から，クローンセットの消失パターンを判定する．

3.3.1 調査対象プロジェクト

本研究で調査を行ったプロジェクトの一覧と，プロジェクトごとの総コミット回数を表 3.1 に示す．調査対象のすべてのプロジェクトが Java 言語で記述されており，バージョン管理システムの一つである git で管理されたオープンソースソフトウェアのプロジェクトである．

表 3.1　調査対象プロジェクト

プロジェクト名	commit 回数
cassandra	4410
common-lang	5069
druid	3856
gson	1286
jEdit	7967
jUnit	1229
leafPic	632
okhttp	1776
picasso	649
poi	965

第 4 章　調査結果

本章では，3 章で述べた調査方法に基づいて，実際のプロジェクトの履歴中で消失したクローンセットの消失パターンの出現頻度を計測し，実際にどのような消失パターンが出現するのかを調査した．その結果を表 4.1 に示す．

また，履歴中で計測したクローンセット消失パターン数の割合から，プロジェクト内でクローンセットがどのように消失傾向であったのかを明らかにするために以下の内容について調査した結果を報告する．

- 実プロジェクト上で出現したクローンセットの消失パターン
- 実プロジェクトのクローンセット消失パターンの割合

表 4.1　プロジェクトごとのクローンセットの消失パターン

プロジェクト名	D	DN	C	CN	M	MN	DC	DM	合計
cassandra	132	90	42	61	2	10	33	14	384
common-lang	1113	80	63	72	1	0	4	14	1347
druid	76	35	42	129	0	2	3	7	294
gson	27	23	11	10	0	1	0	0	72
jEdit	129	112	60	50	0	4	25	5	385
jUnit	76	5	7	14	0	0	0	2	104
leafPic	18	16	4	16	0	1	2	8	65
okhttp	31	13	51	14	1	1	2	0	113
picasso	15	6	7	4	1	0	0	7	40
poi	4	17	0	11	0	0	1	0	33

4.1　実プロジェクト上で出現したクローンセットの消失パターン

定義した 14 パターンのクローンセット消失パターンを用いて，実際のプロジェクトでどのようなパターンでクローンセットが消失したのかを調査した．図 4.1 に，調査対象プロジェクト内で実際に出現したクローンセット消失パターンを示し，観測した消失パターンの内容について述べる．

消失パターン	解釈
D	直接的な削除によるクローンセットの消失パターン
DN	
C	変更操作によって、類似度が変化しクローンセットが消失したパターン
CN	
M	C/CNに加えてメソッドの移動，メソッド名の変更によるクローンセットの消失パターン
MN	
DC	削除、変更操作の組み合わせによるクローンセットの消失パターン
DM	

図 4.1　プロジェクト内で観測したクローンセット消失パターンの一覧

D(Delete) パターン

同じクローンセット内に属するすべてのクローン片が同時に削除されたことで，クローンセットが消失したパターン．

DN(Delete + NotChange) パターン

同じクローンセット内に属するクローン片の内，変更されなかった 1 つのクローン片以外のすべてのクローン片が削除されたことによって，クローンセットが消失したパターン．

C(Change) パターン

同じクローンセット内に属するクローン片すべてに異なる修正が実施された

ことにより，クローン片の類似性が失われ，クローンセットが履歴上で消失したパターン．

CN(Change + NotChange) パターン

同じクローンセット内に属するクローン片の内，変更されなかった 1 つのクローン片以外のすべてのクローン片にそれぞれ異なる修正が実施されたことで，履歴上でクローンセットが消失したパターン．

M(Move) パターン

クローンセット内のすべてのクローン片が，同時に名前の変更あるいはメソッドの移動を実施した場合．

MN(Move + NotChange) パターン

クローンセット内のクローン片のうち 1 つ以上のすべてのクローン片に対して，名前の変更あるいはメソッドの移動が実施されたが，1 つのクローン片が変更されなかったことによって，クローンセットが消失したパターン．

DC(Delete + NotChange) パターン

同じクローンセットに属するクローン片が削除か変更を経験したパターン．詳細には以下の 2 通りが考えられる．

- クローン片 1 つを残して削除された結果，クローンセットではなくなり，残る 1 つが変更されたパターン．
- 1 つ以上のクローン片が削除され，残る複数のクローン片に異なる変更が適用された結果，類似度が低下してクローンセットではなくなるパターン

DM(Delete + Move) パターン

同じクローンセットに属するクローン片が削除かメソッドの移動（名前の変更）を経験したパターン．詳細には以下の 2 通りが考えられる．

- クローン片 1 つを残して削除された結果，クローンセットではなくなり，残る 1 つが変更されたパターン．
- 1 つ以上のクローン片が削除され，残る複数のクローン片に異なるメソッドの移動（名前の変更）が適用された結果，類似度が低下してクローンセットではなくなるパターン．

4.1.1　実プロジェクトのクローンセット消失パターンの割合

次に，調査対象プロジェクト全体のクローンセット消失パターンの割合傾向について報告する．図 4.2 に示すように，クローンセット消失パターンの割合傾向はプロジェクト毎に異なるという結果になった．しかしながら，複数の消失パターンが実際に出現しており，一部の消失パターンの割合が高いプロジェクトが複数存在している．

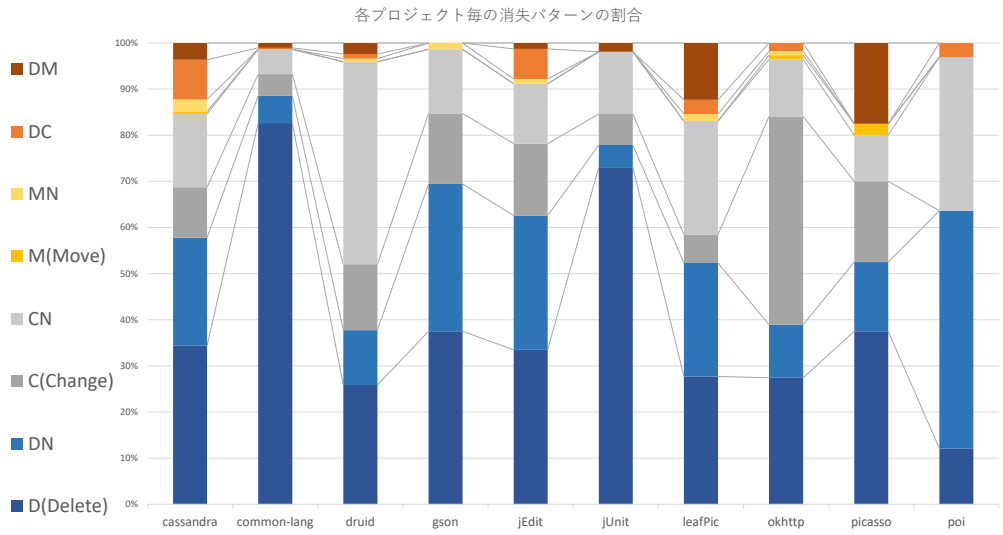

図 4.2　クローンセットの消失パターンの割合

4.2　考察

提案したクローンセット消失パターンのうち，実際に 8 つのクローンセット消失パターンを観測した．クローンセット消失パターンによって，クローンセットが消失した原因が異なると考えられるため，クローンセットが削除された際の状況を整理し，クローンセット消失パターンに基づいたプロジェクトの開発傾向を考察する．

既存研究で Uemura らは，削除（消失した）されたコードクローンと削除されていなかったコードクローンの割合から jEdit や common-lang ではコードクローン

の削除が積極的に行われており，コードクローンの整理を積極的に行うポリシーが確認でき，また逆に，druid ではそのような傾向がみられないと考察している [6]．本調査の消失パターンの割合からは，jEdit については Uemura らと同様のことが考察できる．一方で common-lang については，別の見方ができる．commons-lang では D パターンの割合が顕著に多いことが確認できる．D パターンはすべてのコード片を一括で削除するような消失パターンであり，コードクローンの整理としては質が低い，そのため，D パターンの割合が高いプロジェクトでは，あまり計画的ではないコード片の生成ポリシーの元で開発されている可能性が考えられる．一方で，類似したクローン片は全て削除されているため，一貫性のないクローン操作による保守性の低下リスクは低く保たれている．本調査で分類した消失パターンを用いることで，より細かく，削除されたコードクローンについての分析及び議論を行うことができる．

DN(Delete + NotChange) パターンの割合が高いプロジェクトとしては，poi が挙げられる．パターン DN は，1 つ以上の削除されたクローン片と，1 つの変更されなかったクローン片で構成されている．そのため，一時的な利用のために開発者がコード片を複製し，必要無くなったタイミングで削除したというクローンの利用が考えられる．この消失パターンでは，クローン片が全て削除されずに 1 つのクローン片が維持される．

okhttp は C(Change) パターンの割合が多いプロジェクトである．C パターンでは，異なる修正が同時にすべてのクローン片に対して適用されるため，クローン片それぞれが独自に進化する．そのため，コード片の追跡が困難になり，開発者による一貫性のある修正の適用が難しくなり，修正漏れの発生リスクが高くなると考えられる．ただし，タイプ 3 として検出される場合，クローンセットが維持される可能性があるため，タイプ 3 を検出可能なクローン検出ツールを用いた分析システムを構築することでより詳細に分析できると考えられる．

CN(Change + NotChange) パターンの割合が高い druid,poi プロジェクトでは，1 つのクローン片を除いた全てのクローン片に異なる修正が適用されているため，一貫性のない変更操作が頻繁に実施されていたことが読み取れる．そのため，将来的な修正漏れの発生リスクが高いと考えられる．

DC(Delete + NotChange) パターンの割合が高いプロジェクトとしては cassan-

dra が挙げられる．削除と修正作業の組み合わせのため，リファクタリング操作の可能性が考えられる．

DM(Delete + Move) では，Move 操作と Delete 操作によって別クラスへメソッド抽出された（集約）可能性が高く，DM の比率が高いプロジェクトはリファクタリング実施との関連性を議論することができる．

第 5 章　まとめ

本研究では，コードクローンが履歴上でどのように消失したのかを明らかにするために，１０件のオープンソースプロジェクトを対象としたメソッド単位クローンの履歴情報の解析を行った．コードクローンの履歴途中で消失したクローンセットに着目し，消失原因となった，クローンセット内のクローン片に対して行われた変更パターンから，クローンセット消失パターンを分類し，14 のクローンセット消失パターンを定義したうえで，実プロジェクトの分析及び考察を実施した．実プロジェクト上では，8 つのクローンセット消失パターンが観測され，実際に複数のクローンセット消失パターンでクローンセットが消失していることを確認した．各消失パターンの出現頻度の割合は，各プロジェクト毎に異なるが，特定のクローンセット消失パターンが頻出しているプロジェクトが複数存在している．また，定義したクローンセット消失パターンによって，クローンセットが消失した原因が異なると考えられるため，クローンセットが削除された際の状況を整理し，クローンセット消失パターンに基づいたプロジェクトの開発傾向を考察した．クローンセット消失パターン D に着目すると，同じクローンセット内のクローン片が同時にすべて削除されているという情報を得ることができるため，たとえば，パターン D の情報を用いてプロジェクトの考察を行うと，「削除を前提とした無計画なクローンセットの利用の仕方であり，コードクローンの整理を丁寧に行うポリシーのプロジェクトという見方はできない.」といった議論及び考察をすることができる．また，Uemura らの研究では，変更を経験したクローンセットは少ないと報告されてるが，消失パターンに着目すると，パターン C はタイプ 3 コードクローンなどの形で継続されている可能性がある．それらの追跡及び分析については，今後の調査課題の一つである．これらの追跡は Historage のアーキテクチャを用いることで，実現できる．

今後の課題として，削除だけでなく，履歴上で変更のみを経験したクローンセットに対しても調査を進める必要がある．変更のみを経験したクローンセットは少ないと報告されているが，長期間削除されずに，メンテナンスされ続けているコードクローンに着目することは重要であると考える．また，本研究で分類した抽出パターンは，削除されたコードだけてなく，変更のみを経験したクローンセットに対

しても同様に適用できるため，今後の調査に応用することが可能である．また，本研究で提案した 8 つのクローンセット消失パターンに対応する実際のソースコードの内容を確認することで，より詳細なクローンセット消失パターンの性質を調査するとができると考える．

謝辞

本研究を進めるにあたり，多くの方々のご指導，ご協力，ご支援を頂きました．ここに謝意を添えて御名前を記させて頂きます．

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研究室飯田元教授には，本研究全過程において熱心なご指導を賜りました．度重なる研究内容のご相談にも快く応じて頂き，感謝の念に堪えません．また，論文執筆，発表の仕方のご指導など，多くの御助言を頂きました．心より厚く御礼申し上げます．

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工学研究室松本健一教授には, 研究成果の報告会やなどで本研究に対し貴重な御指導, 御助言を賜りました. 心より感謝申し上げます.

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研究室市川昊平准教授には, 本研究に対し貴重な御指導を頂きました. また, 常日頃から研究活動や学生生活に関しても多くの御助言を頂きました. 心より感謝申し上げます.

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研究室崔恩瀞助教には, 本研究を進めるにあたり多くの御指導を頂きました. 研究方針だけでなく, 論文執筆, 発表方法，研究内容のアドバイスなどにおいても多くのご指導を頂きました. また, 常日頃から研究活動や学生生活に関しても多くの御助言を頂きました. 心より感謝申し上げます.

南山大学理工学部ソフトウェア工学科名倉正剛准教授には，学部時代から，研究方針についてのお話や，研究内容についての熱心なご指導を頂きました．また，就職活動や，学生生活面での支援をして頂き，心より感謝申し上げます．

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研究室の先輩方には, 常日頃から様々なご指導, 助言を頂き, 心より感謝いたします. 特に博士後期課程 3 年の上村さんには，研究内容のご相談，論文の添削，学生生活等の面で格別のご支援を頂き，心より感謝申し上げます．

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研究室の皆様には, 日頃より多くの御協力, 御支援を頂きました. 研究だけでなく, 日々の生活において多くのことを支えていただいたことを心より感謝申し上げます. 最後に, 大学院への進学にあたり多くの支援を頂いた両親に心より感謝申し上げます.

参考文献

[1] 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).

[2] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Transactions on Software Engineering, Vol. 28, No. 7,pp. 654-670, Jul, 2002.

[3] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. DECKARD : Scalable and Accurate Tree-Based Detection of Code Clones. In Proc. of ICSE'07, pp. 96-105, 2007.

[4] 横井一輝 ，崔恩瀞，吉田則裕，井上克郎：情報検索技術に基づくブロッククローン検出，情報処理学会研究報告,Vol.2017-SE-196, No.19, pp.1-8（2017).

[5] 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).

[6] Kyohei Uemura, Akira Mori, Eunjong Choi, Hajimu Iida. Tracking Method-Level Clones and A Case Study In Proceedings of 13th International Workshop on Software Clones (IWSC 2019), (to appear) , Feb. 2019 HangZhou, China.

[7] Roy et, al, Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach, Science of Computer Programming, 2009.

[8] 山中裕樹，崔恩瀞，吉田則裕，井上克郎：情報検索技術に基づく高速な関数クローン検出，情報処理学会論文誌 Vol. 55, No. 10, pp. 2245-2255 (2014).

[9] 中山直輝, 吉田則裕, 藤原賢二, 飯田元, 高田光隆, 高田広章. コードクローンとの位置関係に基づく欠陥発生傾向の調査. 情報処理学会論文誌, Vol. 57, No. 2, pp. 681-93, 2 2016.

[10] M. Mondal, C. K. Roy, K. A. Schneider, “A Comparative Study on the Bug-Proneness of Different Types of Code Clones”, Proc. ICSME, 2015,pp.

91 - 100.

[11] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Historage: Fine-grained version control system for java. In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution, IWPSE-EVOL '11, pp. 96-100, New York, NY, USA, 2011. ACM.

発表リスト

[1]　常木 健介, 崔　恩瀞, 飯田　元, 同時修正されたコードクローンとファイルパス距離の関係の調査 (ポスター発表）, ソフトウェアエンジニアリングシンポジウム 2017