Vol.2014-SE-186 No.1
2014/11/13

情報処理学会研究報告
IPSJ SIG Technical Report

コードクローン編集者数に着目した開発履歴の分析

辻 健二[1,a)]　崔 恩瀞[1]　吉田 則裕[2]　春名 修介[1]　井上 克郎[1]

概要：コードクローンとは，互いに一致もしくは類似したコード片のことである．一般的に，あるコードクローンが編集されると，同じクローンセット（コードクローンの関係となっている全てのコード片の集合）に属されている他のコードクローンも同様に編集される必要がある．しかし，異なる開発者がクローンセット内の各コードクローンの編集を担当している場合は，コードクローンに対する一貫した編集が困難である．この問題を防ぐために，開発者間でのコードクローンの編集情報の共有を支援ツールが開発されている．しかし，このツールは複数人でコードクローンを編集する体制を前提としたものであり，実際のソフトウェア開発現場でのコードクローン編集管理が複数人によって行われているとは限らない．従って，コードクローンを編集している開発者の人数について調査する必要がある．本研究ではある成果物に対して最も編集を行っている開発者がどの程度編集を行っているかを示す Ownership メトリックを用いてオープンソース開発履歴に対してクローンセット毎の編集者の傾向を分析した．

キーワード：コードクローン，ソフトウェア保守，Ownership メトリック

A Version History Analysis Focused on the Number of Developers Editing Code Clones

Abstract: A code clone is a code fragment that is identical or similar to other code fragments in source code. Generally, when a code clone is changed, all the code clones in the same clone set (i.e. a set of code clones that are identical or similar to each other) should be changed consistently. However, if clone codes in the same clone set are changed by multiple developers, it is difficult to change these code clones consistently. To facilitate this problem, previous study developed a tool that notifies the change information of code clones to the developers. The basic premise of this tool is that code clones are changed by multiple developers, however, little is known about number of developers who changes the code clones in practice. In this study, we analyzed the tendency of developers who changed each clone set using Ownership (i.e. a metric that represents ratio of developers editing each component).

Keywords: Code Clone, Software Maintenance, Ownership Metric

1. まえがき

ソフトウェアの保守工程における大きな問題の一つとして，コードクローンが指摘されている．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部分を持つコード片のことである．コードクローンは，主に既存コードのコピーペーストによって生成される [1]．また，互いにコードクローンとなっているコード片の集合はクローンセットである．

コードクローンとなっているコード片が編集された場合，同じクローンセットに属している他のコードクローンに対して一貫した編集が行われる必要がある．しかし，開発者が編集対象のコード片がコードクローンであることに気付かず，そのコード片のみに修正する可能性は非常に高い [2], [3]．この問題を防ぐためには，システムに関わる全ての開発者の間で，コードクローンの編集情報が共有される必要がある．

山中らは開発者間でのコードクローンの編集情報の共有を支援するため，Clone Notifier を開発した [4]．Clone Notifier は，対象システム内にコードクローンが編集され

[1] 大阪大学
Osaka University
[2] 名古屋大学
Nagoya University
[a)] k-tsuji@ist.osaka-u.ac.jp

た際，登録している開発者全員に対してクローン編集の通知を行うツールである．Clone Notifier はコードクローンが複数の開発者によって編集されている開発体制では有用である．しかし，コードクローンが単独または複数の開発者によって編集されているかに関しての十分な調査は行われていない．Clone Notifier の有用性を検証するためには，コードクローンを編集する開発者の人数に関する詳細な調査が必要である．

Harder はコードクローンの編集を行う開発者が単独か複数かを調査し，コードクローンが主に同一の開発者によって編集を行われることが分かった [5]．しかし，この研究は編集割合の最も多い開発者の情報のみ焦点を当てた上に，最も多い開発者のコードクローンの編集割合についても考慮できていないという問題点がある．

これらの問題点を解決するために，本研究は，コードクローンに対する編集者の数を，Ownership[6] というメトリックを用いて調査している．Ownership とは，あるコンポーネントに対して行われた全ての編集のうち，最も多く編集した開発者が行った編集の割合を示すメトリックであり，そのコンポーネントを担当する強さの度合いを示すものとなっている．本研究で Ownership メトリックを用いることで，コードクローンの編集を担当している開発者が単独か複数かを判断できるだけでなく，その編集割合についても理解することが期待されている．Ownership メトリックを用いて，オープンソースソフトウェアである WildFly を対象に各クローンセット内のコードクローンの編集回数や最も多く編集を行った開発者の編集割合を計測した結果，半数以上のクローンファイルが，複数人によって編集されていることが分かった．

2. 背景

2.1 コードクローンの検出

ソースコード中からコードクローンを検出するために，さまざまな類似度に基づく手法が提案されている [7], [8]．

Baker は，コードクローン検出する手法を提案し，Dup を開発した [9]．Dup はトークンの類似度に基づいて，ユーザ定義名や予約語を除く一致箇所のコードクローンを検出する．Jiang らは抽象構文木の類似度に基づいてコードクローンを検出する DECKARD を開発した [10]．DECKARD は，入力されたソースコードを抽象構文木に変換し，その後の各部分木を特徴ベクトルに変換する．また，LSH(Locality-Sensitive Hashing) アルゴリズム [11] を用いて，特徴ベクトル間の類似度を求めることによって，コードクローンの検出を行うことができる．Kamiya らはソースコードのトークン列の類似度に基づいてコードクローンを検出する CCFinder を開発した [12]．CCFinder は，入力されたソースコードをトークン列に変換する．そのとき，変数名や識別子名などのユーザ定義名を特殊文字に置き換えることによって，ユーザ定義名や予約語が異なるコード片でもコードクローンとして検出することができる．CCFinder は検出精度が高いことで知られており，さまざまな研究や会社で利用されている．しかし，大規模のソースコードが入力された場合，コードクローンを検出するために，膨大な時間がかかるという短所を持っている．

この問題を解決するために，Choi らは CCFinder の前処理や後処理を行うことでコードクローンの検出時間を減らす手法を提案した [13]．彼らの手法は，各入力ファイルの MD5 ハッシュ値 [14] を計算し，そのハッシュ値が一致したファイルの集合を完全一致ファイルセットとして検出する．また，特殊なハッシュ値のファイルや，各完全一致ファイルセットから選ばれた 1 つのファイル (代表ファイル) を CCFinder の入力とし，コードクローンの検出を行っている．その後，代表ファイルからクローンが検出された場合，そのファイルのクローン情報に基づいて，マッピングを行い同じ完全一致ファイルセットに属するファイルクローンの位置を特定している．

本手法では Choi らの手法を拡張し，ファイル単位でのコードクローンを検出した．つまり，各入力ファイルを一つのトークン列に連結した．その際，変数名や識別子名などのユーザ定義名は特殊文字に置き換えている．その後は，Choi らの手法と同様，各変換された入力ファイルの MD5 ハッシュ値を計算を行い，各ファイルのハッシュ値に基づきファイルクローンを検出した．

2.2 コードクローンの編集管理

コードクローンが編集された際，そのコードクローンの編集に関わる全ての開発者がその編集内容について把握することで修正漏れやバグの混入などを防ぐことができると考えられる．そのため，コードクローン編集の情報を開発者間で共有することで開発を支援するツール Clone Notifier[4] が開発されている．Clone Notifier は，バージョン間でコードクローンの差分情報を検出し，その情報をツールに登録している開発者に対してメールを送ることで通知するツールである．Clone Notifier が行った企業でのソフトウェア開発への適用実験の結果，開発者たちが意図していなかったコードクローン編集を検出し，それを通知することができた．しかし，Clone Notifier は複数人でコードクローンを編集する開発体制を前提としたものであり，実際のソフトウェア開発現場でのコードクローン編集が複数人によって行われているとは限らない．従って，コードクローンを編集している開発者の人数の調査が必要である．

2.3 コードクローン編集者の分析

コードクローンの編集者とは，コードクローンを生成，もしくは編集した開発者のことを指す．Harder はコードクローン編集者を分析し，同じクローンセット内のコード

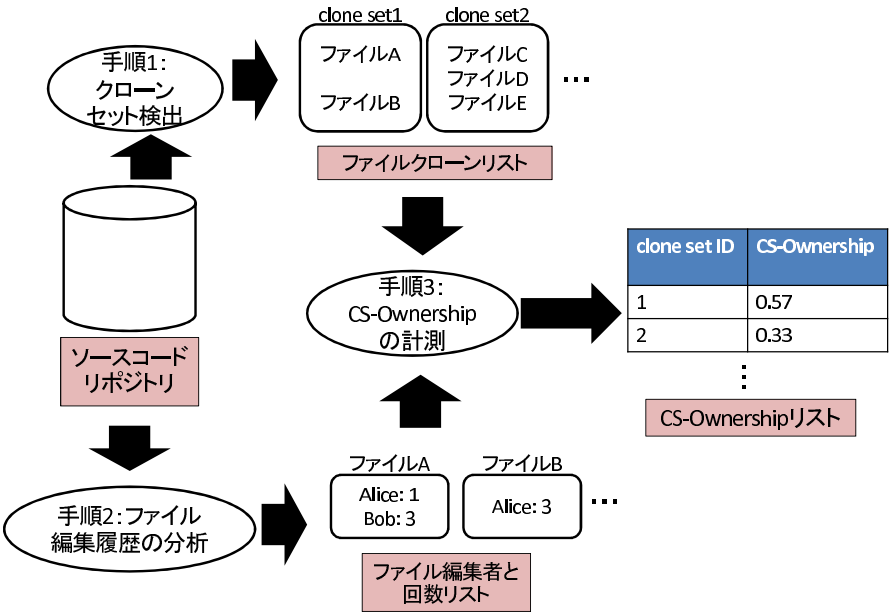

図 1　調査手順の概要
Fig. 1　Overview of analysis process

クローンの編集を行った開発者が単独かまたは複数かを調査した [5]．彼の研究では，トークン単位で編集を担当している開発者の特定を行い，その情報を用いてコードクローンの編集者を決定している．

この研究の結果，対象とした全てのソフトウェアにおいて，単独で編集されているクローンセットの数が複数人のものよりも多いことが明らかになった．しかし，この研究では，編集割合の最も多い編集者の情報のみに焦点を置いている．また，編集者のみに着目しているので，クローンセットに対して行われた編集の割合については考慮されていない．

本研究では，これらの問題点を考慮し，コードクローンに対する編集者の数を，Ownership メトリックを用いて調査している．このメトリックは，あるコンポーネントに対して，開発者が明確に責任を持っているかどうかを表すものである．Ownership メトリックを用いることで，コードクローンの編集を担当している開発者が単独か複数かを判断できるだけでなく，その編集割合についても理解することができる．また，コード片単位でコードクローンを扱うと記述言語にイディオムを多く含むことがあり，検出されるコードクローンはコピーペーストによって生成された以外のものも含まれてしまう．一方でファイル単位のコードクローンは，コピーによって作成されている可能性が高いと考えられる．このため，本研究ではファイルクローンに着目して調査を行っている．

3. 調査手順

本研究では，複数開発者による管理支援の必要性を検証するため，コードクローンの編集を行う開発者が主に単独か複数かをその担当割合にもとづいて調査する．図 1 は本研究の調査実験の手順の概要を示している．この図で表しているように，まず最初に開発履歴を記録しているソースコードリポジトリの，ある時点でのソースコードを対象にファイルクローンセットの検出を行う．また，リポジトリのコミットログから，ファイル編集履歴の分析を行い，全てのファイルの編集者と編集回数を分析する．最後に，Ownership メトリックをファイルクローンセットに適用した *CS-Ownership* を，ファイルクローンセットとファイルの編集者，編集回数の情報を用いて計測する．*CS-Ownership* については，3.3 節にて後述する．調査実験の対象となるシステムは，バージョン管理システムである Subversion または Git を用いて開発履歴が管理され，Java 言語で記述されているオープンソースソフトウェアプロジェクトである．

本節では，本研究で行う各調査手順について具体的に説明している．

3.1 手順 1 クローンセット検出

対象システムの Subversion リポジトリから，コードクローンとなっているクローンファイルを検出する．本研究で扱うコードクローンは，変数名や関数名などのユーザ定義名を除いて一致するコードクローンである．検出したクローンファイルのクローンセットをクローンファイルセットと定義する．各クローンファイルセットには，少なくとも 2 つ以上のファイルが含まれている．ファイル名が同じでも，パス名が異なるクローンファイルについては，両者

は別のファイルであるものとして扱っている．

図 1 の中央上段は，クローンセットの検出によって得られたファイルクローンリストを示した図である．ファイル A とファイル B は Clone set1 に属しているコードクローンである．Clone set2 は，ファイル C，ファイル D，ファイル E の 3 つのファイルがコードクローンとなっているクローンセットである．

3.2 手順 2 ファイル編集履歴の分析

対象システムの開発履歴を元にコミットログを分析し，各ファイル毎の編集者とその編集回数を取得する．コミットログの各リビジョンには編集を受けたファイルが記録されており，そのリビジョンをコミットした開発者をそのファイルの編集者とする．また，あるファイルのコミット回数を，そのファイルの編集回数としている．図 1 の中央下段は，分析後の出力であるファイル編集者と回数のリストである．ファイル A に対して，開発者 Alice が 1 回，Bob が 3 回の編集を加えており，ファイル B については Bob が 1 回のみ編集を行っていることを表している．

本実験では，コードクローンの生成する過程ではなくコードクローンに対する編集の回数を取得するのが目的であるため，コードクローンを検出したバージョン以降のリビジョンを対象にファイル編集履歴の分析を行う．そのため，1 度も編集が検出されないファイルが含まれる可能性もある．

本研究での開発履歴の分析は，Subversion のコミットログを用いて行っている．そのため，Git で開発履歴の管理を行っているシステムに関しては，Subversion のリポジトリに変換した後に分析を行っている．

3.3 手順 3 *CS-Ownership* の計測

手順 1 と 2 で得られたデータを元に，各クローンセットに対する Ownership メトリックの計測を行う．手順 1 で検出したクローンセット内の全てのクローンファイルの編集者とその回数の総計を，手順 2 で求めたファイル編集履歴を用いて計算する．このうち，最も編集回数の多い開発者についての Ownership メトリックを計算する．

Ownership メトリックとは，あるコンポーネントに対して行われた全ての編集のうち，最も多く編集した開発者が行った編集の割合を示すメトリックであり，開発者がそのコンポーネントを担当する責任の強さの度合いを示すものとなっている [6]．単純に編集者の人数を数えるだけではなく，Ownership メトリックを用いる利点として，以下のメリットが挙げられる．

- Ownership メトリックの値を見て，関わっている開発者が単独か複数かを直感的に判別可能である．値が 1 であれば単独での編集，そうでなければ複数の開発者が編集に関わっていることが容易に理解できる．
- 複数人での編集であった場合，Ownership メトリック値はその分担の度合いを反映することが可能である．例えば，Ownership メトリックの値が 0.25 であった場合，複数の開発者が編集に関わっていること以外にも，そのコンポーネントに対して，全ての開発者の編集回数が 1/4 以下であることが把握できる．

一般的に，あるコンポーネントの Ownership メトリックの値は，コンポーネントの総コミット数を C_{total}，そのコンポーネントの最多コミットを行った開発者のコミット数を C_{max} とすると，以下の式で算出することができる．

$$Ownership = \frac{C_{max}}{C_{total}}$$

Ownership メトリックのとり得る値は，$0 < Ownership \leq 1$ となる．Ownership メトリックの値が 1 に近いほど，1 人の開発者がそのコンポーネントに対して大きな責任を持ち，0 に近いほど複数の開発者が均等に編集する機会を持つことを表している．Ownership メトリックが 1 のとき，1 人の開発者が完全にそのコンポーネントを管理していることを示している．

本研究では，ファイルクローンセットを対象とした Ownership メトリックとして，*CS-Ownership* を定義する．ファイルクローンセットに含まれる全てのファイルについて，そのコミット数の総計を CS_{total}，全コミットのうち最多の開発者コミット数を CS_{max} とすると，ファイルクローンセットの *CS-Ownership* は以下の式によって算出できる．

$$CS\text{-}Ownership = \frac{CS_{max}}{CS_{total}}$$

図 1 右側の表は，各クローンセットと，その *CS-Ownership* 計測結果の例を表している．クローンセット 1 に含まれる全てのファイル (ファイル A，ファイル B) に対する編集回数を合計すると，Alice が計 4 回，Bob が計 3 回となる．この場合，最も多い編集回数は 4 回なので，クローンセット 1 の *CS-Ownership* は $4/(3+4) \approx 0.57$ となる．

4. 調査結果と考察

本節では，本研究で行ったクローンセット編集者の調査結果を示し，それについての考察および妥当性への脅威について述べる．

4.1 調査対象システム

本研究は，WildFly[*1] を調査対象システムとして実験を行った．対象システムの概要を表 1 に示す．総リビジョン数はバージョン管理システム内に記録されている全てのリビジョン数，調査リビジョン数は対象システム中の，ファイル編集者の分析を行ったリビジョンの数を示している．

[*1] http://wildfly.org/

表 1　調査対象のシステム
Table 1　Characteristic of analyzed system

対象システム	WildFly
総リビジョン数	25256
バージョン	7.0.0
調査リビジョン数	23693
調査対象期間	3 年 10 ヶ月

表 2　調査結果の概要
Table 2　Investigation result

ファイルクローンセット個数	14
クローンセット内ファイル数	最大 3 最小 2 平均 2.2
延べ編集回数	153
平均 *CS-Ownership*	0.53

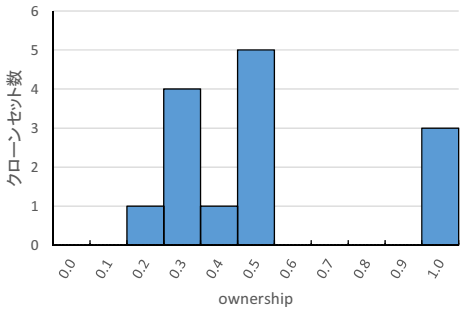

図 2　*CS-Ownership* の分布
Fig. 2　Frequencies of *CS-Ownership*

これは，同じく表に記載している対象システムのバージョンがリリースされた段階でのソースコードからコードクローン検出を行っているため，その段階以降にコミットされたリビジョンの数を表す．調査対象期間は，ファイル編集者の分析を行ったリビジョンがコミットされている期間のことを表している．今回対象としたシステムには，開発者の代わりに自動的にコミットされたリビジョンは存在しなかった．

4.2　*CS-Ownership* 計測結果

CS-Ownership の計測結果を表 2 に示す．延べ編集回数とは，全てのファイルクローンに対する編集回数の合計を示している．本実験では，全てのクローンファイルに対して 1 回以上の編集が確認できた．

図 2 は，*CS-Ownership* によるクローンセットの度数分布を示している．横軸は *CS-Ownership* の値，縦軸はその値を取るクローンセットの数を示している．横軸の下についている数値は *CS-Ownership* が 1 であるファイルクローンセットはいくつか存在するが，1 ではないファイルクローンセットの方が多数を占めていることがわかる．つまり，多数のファイルクローンが，複数の開発者によって編集されていることを示している．また，0.50 より大きく 1 より小さな値の *CS-Ownership* を持つファイルクローンセットは見つからなかった．

表 3，表 4 は，WildFly に存在したクローンセット内に含まれるファイルと，その編集者 ID，編集回数を示している．表 3 では，編集者 brian.stansberry が最も多くファイルの編集を行っているが，クローンセット内の 3 つのファイルのうち，1 つのファイルしか編集していない．また，表 4 では，jason.t..greene が 11 回，david.m..lloyd が 8 回の編集を行っており，他の開発者は 1 回か 2 回のみの編集しかしていない．

4.3　結果の考察

CS-Ownership の値が 0.50 より大きく 1 未満の場合，複数の開発者でコードクローンを編集しているが，強く責任を持っている開発者が存在する，といった開発体制であると考えられる．このような開発体制の下では，中心となる開発者がコードクローンの編集情報を把握できれば良いため，他の開発者がコードクローンの編集情報を共有する必要性が薄い．しかし，図 2 より，そのような *CS-Ownership* の値を取るファイルクローンが存在しないことから，上記の開発体制を取っていないことが理解できる．*CS-Ownership* が 0.50 のクローンセットは 5 つ見られるが，いずれも 2 人の開発者による編集であった．2 人の間で開発の連携が取れていればコードクローン編集の情報共有を支援する必要は無いが，連携が困難な場合はコードクローン編集の情報共有を支援する必要がある．14 のクローンセットのうち，6 つが 0.50 を下回る *CS-Ownership* であった．この状態は，複数人がコードクローンを編集しているが中心となる開発者がいないことを示している．互いに連携してコードクローンの編集情報を共有することは困難であると考えられるため，コードクローン編集情報の共有を支援するツールの必要性は高いと考えられる．

表 3 は，*CS-Ownership* が 0.21 と最も低かったクローンセットの詳細を示している．3 つのファイル全てを編集できている開発者はおらず，2 つのファイルを編集している開発者も jason.t..greene だけである．最も編集回数の多い brian.stansberry も，1 つのファイルに対する編集しか行っていない．よって，このファイルクローンセットは，一貫した編集ができていないと言える．このようなコードクローンの開発には，コードクローン編集情報を共有する必要があると考えられる．表 4 は，*CS-Ownership* は 0.41 と低めの結果だが，ファイルクローンセット内の全 2 ファイルを編集している開発者が，最も多く編集を行っている jason.t..greene を含めて 3 人存在している．また，その 3 人ともが 2 ファイルをほぼ均等に編集している．*CS-Ownership* の値は低いが現状では，コードクローンに対する一貫した編集は行われていると言える．しかし，分担して編集を行っているため，将来的に人員異動などの要因で編集者が変更された際に連携が取れなくなる恐れがあ

表 3　ファイルクローンセットの詳細データ（$CS\text{-}Ownership = 0.21$）
Table 3　Detailed data on file clones set（$CS\text{-}Ownership = 0.21$）

ファイル名	編集者 ID	編集回数
/server/src/main/java/org/jboss/as/server/ SystemExiter.java	heiko.braun	1
	markus.gaisbauer	2
	frank.langelage	4
	matus.abaffy	2
	martin.kouba	2
	scott.marlow	2
	jaikiran	2
	jason.t..greene	6
	kabir.khan	2
	brian.stansberry	10
	radoslav.husar	4
	stuart.douglas	8
/server-manager/src/main/java/org/jboss/ as/server/manager/SystemExiter.java	emanuel.muckenhuber	1
/testsuite/protocol/modules/src/main/java/org/jboss/test/ as/protocol/support/server/manager/SystemExiter.java	kabir	1
	jason.t..greene	1

表 4　ファイルクローンセットの詳細データ（$CS\text{-}Ownership = 0.41$）
Table 4　Detailed data on file clones set（$CS\text{-}Ownership = 0.41$）

ファイル名	編集者 ID	編集回数
/jmx/src/main/java/org/jboss/as/jmx/tcl/ SecurityActions.java	heiko.braun	1
	kabir	1
	jaikiran	1
	jason.t..greene	5
/sar/src/main/java/org/jboss/as/ service/SecurityActions.java	heiko.braun	1
	david.m..lloyd	8
	jaikiran	2
	jason.t..greene	6
	brian.stansberry	2

るので，クローン編集情報の共有支援ツールを検討する必要はある．

4.4　妥当性への脅威

本実験では，バージョン管理システムである Subversion もしくは Git を用いて開発履歴の管理を行っているオープンソースシステムに対してログの分析を行っている．バージョン管理システムの機能として，ファイルの移動やリネームなどによるパス名の変更も開発履歴として記録できるが，本研究の調査手順ではファイルの移動やリネームによるパス名の変更には対応できていない．今後は，パス名の変更があった際もそのファイルを同じものと扱うことが必要であるがバージョン管理システムを用いずにそのような操作が行われていた場合は，以降のリビジョンで同じものとみなすことができなくなる．

また，オープンソースシステムである WildFly のコードクローンに対して調査を行ったが，今後，他のオープンソースシステムに対しても調査を行い，その結果を比較する必要がある．

5.　まとめと今後の課題

本研究では，複数の開発者に対するコードクローン管理支援の必要性を検証するため，オープンソース開発履歴に対してクローンセット毎の編集者数を，Ownership メトリックに基づいたメトリック *CS-Ownership* を定義することでクローンファイル単位で分析し，クローンセットの編集を行う編集者数を調査した．そして，同じクローンセット内のクローンファイルの編集を行う開発者が単独であることよりも，複数であることの方が多いという結果を得ることができた．また，*CS-Ownership* の値から，複数の開発者によるコードクローン編集情報の共有を支援するシステムの必要性があることを確認した．

今後の課題として，以下が挙げられる．

- 本研究では 1 つのシステムに対して調査を行ったが，他のシステムについても同様の調査を行い，今回の結果と比較することが必要であると考えられる．

- ファイル単位でのコードクローンに着目していたが，機能別に編集管理を行っている場合もある．そのため，関数単位での Ownership メトリックを計測することも課題であると考えられる．

謝辞　本研究は，JSPS 科研費 25220003，26730036 の助成を受けたものである．

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] Jiang, L., Su, Z. and Chiu, E.: Context-based Detection of Clone-related Bugs, *Proc. of ESEC-FSE '07*, pp. 55–64 (2007).
[3] Li, Z., Lu, S., Myagmar, S. and Zhou, Y.: CP-Miner: finding copy-paste and related bugs in large-scale software code, *Software Engineering, IEEE Transactions on*, Vol. 32, No. 3, pp. 176–192 (2006).
[4] Yamanaka, Y., Choi, E., Yoshida, N., Inoue, K. and Sano, T.: Applying clone change notification system into an industrial development process, *Proc. of ICPC '13*, pp. 199–206 (2013).
[5] Harder, J.: How Multiple Developers Affect the Evolution of Code Clones, *Proc. of ICSM '13*, pp. 30–39 (2013).
[6] Bird, C., Nagappan, N., Murphy, B., Gall, H. and Devanbu, P.: Don'T Touch My Code!: Examining the Effects of Ownership on Software Quality, *Proc. of ESEC/FSE '11*, pp. 4–14 (2011).
[7] Roy, C. K. and Cordy, J. R.: A survey on software clone detection research, School of Computing TR 2007-541, Queen' s University, Vol. 541 (2007).
[8] Rattan, D., Bhatia, R. and Singh, M.: Software clone detection: A systematic review, *Inform. Softw. Tech.*, Vol. 55, No. 7, pp. 1165 – 1199 (2013).
[9] Baker, B. S.: On Finding Duplication and Near-Duplication in Large Software Systems, *Proc. of WCRE '95*, pp. 86–95 (1995).
[10] Jiang, L., Misherghi, G., Su, Z. and Glondu, S.: DECKARD: Scalable and accurate tree-based detection of code clones, *Proc. of ICSE '07*, pp. 96–105 (2007).
[11] Indyk, P. and Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality, *Proc. of STOC '98*, pp. 604–613 (1998).
[12] Kamiya, T., Kusumoto, S. and Inoue, K.: 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).
[13] Choi, E., Yoshida, N., Higo, Y. and Inoue, K.: A Clone Detection Approach for a Collection of Similar Large-scale Software Products, *IEICE Technical Report*, Vol. 112, No. 275, pp. 117–121 (2012).
[14] Rivest, R. L.: The MD5 Message Digest Algorithm, Internet RFC 1321 (1992).