Vol.2016-SE-193 No.25
2016/7/15

コードクローンの作成・利用過程における人的影響を調査するための追跡ツールの試作

大平 雅雄[1,a)]　久木田 雄亮[2,b)]

概要：コードクローンの特定手法や分析手法がこれまで盛んに研究されているが既存研究の多くは主に，ある時点のソフトウェアシステムに含まれるコードクローンとソフトウェアシステムの品質との関係に着目しているため，コードクローンが作成される過程や原因については未だ十分には明らかになっていない．特に，特定の開発者が作成したコードクローンと後に発見される不具合との関係や，開発者間でのコードクローンの利用（共有）とソフトウェア品質との関係など，コードクローンの作成および利用過程に関与した開発者の人的側面に着目したコードクローンの分析はほとんどおこなわれていない．コードクローンの作成および利用過程を開発者との関連性も含めて大規模な調査をおこなうことで，コードクローンの管理について有益な知見を提供できる可能性がある．本稿では，コードクローンの作成および利用過程における人的影響を調査するために現在構築中のツール CCT (Code Clone Tracer) を紹介し，今後の展開について議論する．

Masao Ohira[1,a)]　Yusuke Kukita[2,b)]

1. はじめに

ソフトウェア開発・保守の過程では，生産性の向上を目的として既存コードが頻繁に再利用されることがあるため，「同一の」または「類似する」コード片（本稿ではコードクローンあるいは単にクローンと呼ぶ）がソースコード全体に遍在することが少なくない．コードクローン中の欠陥を除去するなどの理由により，コードクローンに変更を加える必要性が生じた場合，変更箇所が多岐に渡るため通常よりも保守コストが増加したり，変更漏れにより新たに不具合を混入してしまう恐れがある．そのため，コードクローンの検出手法や分析手法がこれまで盛んに研究されてきた．

しかしながら，既存研究の多くは主に，ある時点のソフトウェアシステムに含まれるコードクローンあるいはコードクローンの変更がソフトウェアシステムの品質に与える影響に着目しているため，コードクローンが作成される過程や原因については未だ十分には明らかになっていない．特に，特定の開発者が作成したコードクローンと後に発見される不具合との関係や，開発者間でのコードクローンの利用（共有）とソフトウェア品質との関係など，コードクローンの作成および利用過程に関与した開発者の人的側面に着目したコードクローンの分析はほとんどおこなわれていない．近年のソフトウェアシステムは長期に渡り保守運用されることが多く担当者が入れ替わることも少なくない．コードクローンの作成および利用過程を開発者との関連性も含めて大規模な調査をおこなうことで，コードクローンの長期的かつ安全な管理方法について有益な知見を提供できる可能性がある．

本稿では，コードクローンの作成および利用過程における人的影響を調査するために現在構築中のツール (CCT: Code Clones Tracer) を紹介し，今後の展開について議論する．続く 2 章ではまず，関連研究を紹介し本研究の立場を明らかにする．3 章では，現在構築中の CCT の実装について詳述する．4 章では，CCT を用いて今後実施する予定の分析内容について議論するとともに，ツールの限界についても言及する．最後に 5 章で本稿をまとめる．

[1] 和歌山大学システム工学部
Sakaedani 930, Wakayama 640-8510, JAPAN
[2] 和歌山大学大学院システム工学研究科
Sakaedani 930, Wakayama 640-8510, JAPAN
[a)] masao@sys.wakayama-u.ac.jp
[b)] s161018@sys.wakayama-u.ac.jp

2. 関連研究

2.1 コードクローンの追跡

コードクローンが多数存在するソフトウェアシステムの保守においては，1 つのコードクローンの変更により他の多数のコードクローンを同時に変更する必要性が生じる場合がある．そのため，保守コストの増加や変更漏れによる品質低下を招くなど，コードクローンの存在については従来否定的な意見が支配的であった [15], [16]．しかしながら，近年の研究によりコードクローンがソフトウェア保守にとって必ずしも負の影響を与えるものではないことも明らかになりつつある [11]．

これまでの研究は主に，ある時点でソフトウェアシステムに含まれるコードクローンとソフトウェアシステムの品質との関係を調査するものがほとんどであったが，今後はコードクローンを適切に管理するためにコードクローンの作成・利用過程に関する知見を蓄積することが重要と考えられる．

コードクローンの作成・利用過程を分析するために必要となるコードクローンの追跡手法はすでにいくつか提案されている [3], [4], [5] が，Git リポジトリをはじめとする近年の分散型版管理システムで活用されているブランチとマージによる版管理には対応していないため，本研究ではまず Git リポジトリに対応可能な追跡ツールを構築することにした．

2.2 コードクローンの作成・利用過程における人的影響

近年のソフトウェアシステムは長期に渡り保守運用されることが多く担当者が入れ替わることも少なくない．コードクローンの作成および利用過程を開発者との関連性も含めて大規模な調査をおこなうことで，コードクローンの長期的かつ安全な管理方法について有益な知見を提供できる可能性がある．

コードクローンの作成および利用に関しては，開発者への直接のインタビューやアンケート調査により，コピーアンドペーストの利用方法や頻度，理由などが分析されてきた [7], [8]．ただし，コードクローンの作成および利用過程に関与する開発者の人的影響に着目した定量的な研究はまだ多くは存在しない．

森脇ら [13], [14] は，開発者ごとのコードの再利用傾向や，再利用されやすいコードを記述する開発者や積極的に再利用を行う開発者の特徴について分析している．辻ら [12] は，コードクローンの管理の観点から，コードクローンの編集に関与する開発者の数やコードクローンの Ownership [1] を調査している．いずれの研究も分析対象プロジェクトの規模や数の制約から一般性の高い知見を導き出すまでには至っていない．また，コードクローンの作成および利用過程に関与した開発者の人的影響とソフトウェアシステムの品質との関係についても分析されていない．

本研究では，Github に登録されている膨大な数のプロジェクトを分析可能にすることを狙い，Git リポジトリに対応した追跡ツールを構築することにした．また，コードクローンの追跡だけでなく，不具合管理システムから収集した不具合データを用いて不具合が混入したコードクローンを特定する機能を追跡ツールに付与することで，コードクローンを作成・利用する開発者がソフトウェアシステムの品質にどのような影響を与えるのかを分析できるようにした．

3. コードクローンの作成・利用過程における人的影響を調査するための追跡ツール

本章では，コードクローンの作成・利用過程における人的影響を調査するために現在構築しているツール CCT (Code Clone Tracer) を紹介する．CCT は，数千から数万リビジョンからなるソフトウェアシステムに存在するコードクローンの追跡を想定しており，高速にクローンを特定することができる粗粒度なクローン特定方法 [9] を用いてクローンを追跡する [*1]．CCT は主に，すべてのリビジョンに対して，(1) コード片を特定する処理，(2) クローンを特定する処理，(3) クローンを追跡する処理，(4) クローンの作成・利用に関与した開発者を特定する処理，(5) 不具合が混入したクローンおよび不具合が修正されたクローンを特定する処理を提供する．以降では，各処理の詳細を述べる．

3.1 Step 1: 各リビジョンのコード片特定

Step 1 では，分析対象ファイル中に含まれるすべてのコード片を特定する．具体的には，以下の手順で追加・変更・削除されたファイルをすべてのリビジョンに対して特

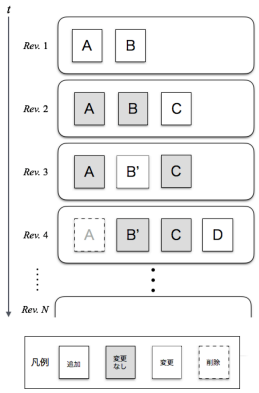

図 1 ファイル変更情報の取得

[*1] 実際，クローンの特定までの処理（コード片抽出，正規化，ハッシュ化）は，CRD (Clone Region Descriptors)[3] ベースのクローン追跡ツール ECTEC (Enhancement of CRD-based Clone Tracking)[4] の一部を再利用して実装している．

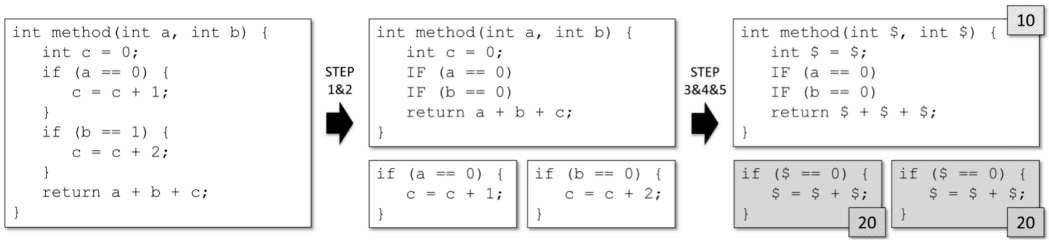

図 2 ブロックの検出・正規化・ハッシュ化の流れ（[9] より抜粋）

定し（図 1），[9] と同様にブロック単位でコード片を特定する（図 2）．

3.1.1 各リビジョンのファイル変更情報（差分）の取得

まず，分析対象リポジトリのコミット履歴から，リビジョン番号，コミット日時，コミッタ名，追加・変更・削除されたファイルについての情報（コミット情報と呼ぶ）を取得しておく．次に，分析対象リポジトリの最初のリビジョン（リポジトリに初めてコミットされたファイル群から構成されるリビジョン）を調査し，最初のリビジョンを構成する全ファイルを特定する（図 1 における *Rev.*1）．コミット情報を用いて，各リビジョンに加えられたすべての変更（ファイルの追加・変更・削除）を，最初のリビジョンを起点として順次遡り最新のリビジョンまで特定する．なお，ソースコード以外のファイルについては分析対象ではないため除外する．

3.1.2 差分情報を用いたコード片特定

次に，最初のリビジョンを構成する全ファイルに対してブロック単位でコード片を特定する（図 2 の STEP1&2）．ここでのブロックとは，クラスやメソッド，for 文や if 文などのブロック文を指す．次に，最初のリビジョンを起点として以降のリビジョンを構成するファイルについてもコード片を順次特定する．このとき，最初のリビジョン以降のリビジョンでは，ファイルの変更情報に基づいて，変更が加えられたファイルのみに対してコード片を特定し，ファイルの変更がコード片の追加・変更・削除のいずれによるものかを特定する．変更が加えられたファイルに対してのみコード片を特定することで，各リビジョンに含まれるすべての構成ファイル群に対してコード片を特定するよりも大幅に処理時間を短縮することができる．なお，Git リポジトリでは，ブランチのマージが頻繁に行われるため，マージコミットの情報に基づいて変更の衝突が生じている場合には，マージ元の 2 つのコミットとの差分情報を取得し，変更が加えられたファイルに対してのみコード片を特定している．

3.2 Step 2: クローンの特定

Step 1 により，すべてのリビジョンに対してブロック単位ですべてのコード片が特定される．Step 2 ではまず，

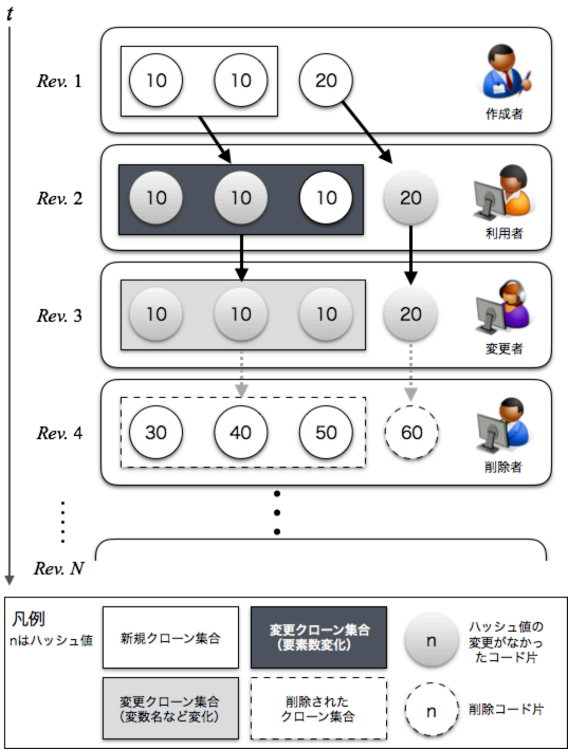

図 3 コードクローンの追跡と開発者の分類

[9] と同様の方法（図 2 の STEP3&4&5）で，特定されたブロックを定められたフォーマットに整形する．また，ブロックに含まれる変数名とリテラルは特殊文字することで正規化し，変数名やリテラルのみが異なるようなクローンを特定できるようにしておく．さらに，正規化されたブロックの文字列をもとにハッシュ値を算出する．ブロック単位でのコードクローン特定 [9] では，ハッシュ値の一致するブロックをクローンと見なす．最後に，ハッシュ値の比較によりクローンを特定する．なお，クローンの特定においても，最初のリビジョンではすべてのコード片に対してクローンの特定処理をおこなうが，以降のリビジョンについては変更が加えられたファイルに含まれるコード片のみを対象にハッシュ値を求めクローンを特定する．

3.3 Step 3: クローンの追跡

Step 2 により，すべてのリビジョンに対してブロック単位ですべてのクローンが特定される．Step 3 ではまず，最

初のリビジョンに含まれるすべてのクローンのハッシュ値を比較し，同じハッシュ値を持つクローンの集合をクローン集合としてまとめ，以降のリビジョンでのクローン集合の前後関係を特定できるようにする．

具体的には，以下の方法でクローン集合の前後関係を特定しクローン集合をリビジョン間で追跡する（図 3）．まず，次のリビジョンで追加・変更・削除されたコード片のハッシュ値を調べ，直前のリビジョンのクローンのハッシュ値と比較する．(1) ハッシュ値が一致した場合は，前のリビジョンに存在するクローン集合の要素であるので要素数を前後のリビジョンで比較する．要素数の増加によるものか，ハッシュ値が変化しない変数名やリテラルの変更によるものかを判別後，前のリビジョンのクローン集合を親集合，次のリビジョンのクローン集合を子集合と解釈し，クローン集合間にリンクを貼る．(2) ハッシュ値が一致しない場合は，新たなクローン集合が作成されたものと解釈し，該当クローン集合を以降のリビジョンでの追跡対象とする．(3) 最後に，前のリビジョンにのみ存在するクローン集合は削除されたものと解釈して追跡対象から除外する．

3.4 Step 4: クローンの利用・作成に関与した開発者の特定

Step 3 により，クローン集合の作成・削除・変更（要素数の増減およびハッシュ値の変更を伴わない変更）を含むクローンの作成・利用過程が追跡できる．Step 4 では，クローンの作成・利用過程に関与した開発者を以下の 4 種類に分けて特定する（図 3）．

- 作成者：オリジナルのクローンを作成した開発者（親となるクローン集合を持たないクローンを作成した開発者）
- 利用者：既存のクローンを他の場所に移植した開発者（クローン集合のクローン要素を増やした開発者）
- 変更者：ハッシュ値の変更を伴わないクローンに対する変更をおこなった開発者
- 削除者：ファイルの削除などによりクローンを削除した開発者

Step 1 により，どのリビジョンにどの開発者が関与したかは分かっているため，Step 3 の結果を用いて 4 種類の開発者とリビジョン番号とを紐付ける．

3.5 Step 5: 不具合混入クローンの特定

Step 1 から Step 4 までは Git リポジトリから得られる情報に基づいた処理であり，クローンの作成・利用過程に関与した開発者が混入した不具合に関する情報は別途，不具合管理システム（Bugzilla や Jira など）から取得する必要がある．本ツールには，SSZ アルゴリズム [6] を用いて不具合混入コミットを特定し，不具合を混入した開発者を分析する機能を実装した．具体的には，以下の処理により不具合が混入したクローンと開発者とを特定し紐付けをおこなう．

- あらかじめ収集しておいた不具合票から Issue ID と不具合が報告された日時を取得する．
- コミットメッセージに Issue ID が記録されている（該当する不具合を修正したことを意味する）コミットから，リビジョン番号と変更されたファイルを特定する．
- 特定したリビジョン番号を Step 3 までの結果に照らし合わせ，不具合修正のために作成・削除・変更が行われたクローン集合を特定する．
- 特定したファイルには不具合報告の時点で不具合が混入しているはずなので，同ファイルが以前変更された時点に不具合が混入したと解釈し，以前変更された時点のリビジョン番号を特定する．
- 特定したリビジョン番号を Step 3 までの結果に照らし合わせ，不具合が混入したクローン要素を特定する．

4. 今後の展開について

CCT を用いることで，コードクローン・開発者・ソフトウェア品質の 3 つの観点を組み合わせた分析が可能となる．また，Git リポジトリの解析を可能にしたことで Github に登録されている膨大な数のプロジェクトを分析対象とすることができる．これらの利点を活かし，今後は多数のプロジェクトを対象として，以下のような分析を実施する予定にしている．

- コードクローンの作成と利用に関与した開発者の数とソフトウェア品質：特定のファイルに存在するコードクローンを一人の開発者が保守した場合と複数人で保守した場合でソフトウェア品質にどのような影響があるのかについて調べる．辻らの研究 [12] では，単独の開発者が保守する場合が多いことや主要な開発者が存在しないクローン集合が存在することを明らかにしているが，ソフトウェア品質との関係については調査していない．保守担当者が入れ替わることでソフトウェア品質にどのような影響があるのかを調査することにより，長期に渡り保守運用されることが多くなったソフトウェアシステムの保守活動におけるコードクローンの適切な管理方法について有益な知見を提供できる可能性がある．
- 開発者の社会的関係とソフトウェア品質：大規模 OSS 開発では依存関係のあるような特定のモジュール群を共同で作成するサブグループが形成されることが知られている．また，密に協働するサブグループが作成するモジュールは，その他のモジュールに比べて品質が高い傾向がある [2]．CCT を用いることでコードクローンの共同保守における開発者の社会的関係とソフトウェア品質との関係についても調査できる．すなわ

ち，これまでの研究はファイルやモジュール単位での調査であったのに対して，コードクローンというより粒度の細かな分析が可能となる．コードクローンの共同保守において，他者が作成したコードクローンを変更したり利用する際に留意すべき事柄などについて有益な知見を提供できる可能性がある．

- コードクローンの変更間隔および変更担当者とソフトウェア品質：作成するソフトウェアあるいはそれに含まれるモジュールの特性により，コードクローンが編集される頻度は一定ではなく，場合によっては，数ヶ月から数年の間一度も変更されなかったコードクローンを編集する必要が生じる場合がある．長い間変更されなかったコードクローンを変更することにより予期しない不具合が生じることもありえる．さらに，保守担当者が変わっている場合が多いと考えられる．このように，コードクローンの変更間隔および変更担当者とソフトウェア品質の関係を調査することにより，ソフトウェア保守におけるコスト見積もりや影響調査にとって有益な知見を提供できる可能性がある．

なお，CCT は粗粒度なクローン特定方法 [9] に基づいてコードクローンを特定するため，既存ブロックへのコード追加といった編集操作が加わるとクローンの追跡が正確におこなえなくなる可能性が生じる．堀田らの分析 [9] では，粗粒度なコードクローン特定手法は細粒度な特定手法に比べ，検出できるコードクローンの数は劣るものの，検出精度は同等とみなせることを明らかにしている．今後の分析を通じて，クローンの追跡に失敗する割合についても調査するとともに，[4], [10] などを参考にクローン追跡の失敗を回避するための方法についても検討する必要がある．

5. まとめ

本稿では，コードクローンの作成および利用過程における人的影響を調査するために現在構築中のツール CCT (Code Clone Tracer) を紹介した．近年のソフトウェアシステムは長期に渡り保守運用されることが多く，運用保守に携わる担当者が入れ替わることも少なくない．コードクローンの作成および利用過程を開発者との関連性も含めて大規模な調査をおこなうことで，コードクローンの長期的かつ安全な管理方法について有益な知見を提供できる可能性がある．今後は本稿で議論した調査を実施するとともに，ツールの改良にも取り組む予定である．

謝辞 本研究の一部は，文部科学省科学研究補助金（基盤 (C): 15K00101）による助成を受けた．本稿で紹介した CCT は，ECTEC[4] の一部を再利用して実装した．

参考文献

[1] Bird, C., Nagappan, N., Murphy, B., Gall, H. and Devanbu, P.: Don't Touch My Code!: Examining the Effects of Ownership on Software Quality, *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2011)*, pp. 4–14 (2011).

[2] Bird, C., Pattison, D., D'Souza, R., Filkov, V. and Devanbu, P.: Latent Social Structure in Open Source Projects, *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE2008)*, pp. 24–35 (2008).

[3] Duala-Ekoko, E. and Robillard, M. P.: Clone Region Descriptors: Representing and Tracking Duplication in Source Code, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 20, No. 1, pp. 3:1–3:31 (2010).

[4] Higo, Y., Hotta, K. and Kusumoto, S.: Enhancement of CRD-based Clone Tracking, *Proceedings of the 2013 International Workshop on Principles of Software Evolution (IWPSE2013)*, pp. 28–37 (2013).

[5] Kim, M., Sazawal, V., Notkin, D. and Murphy, G.: An Empirical Study of Code Clone Genealogies, *Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2005)*, pp. 187–196 (2005).

[6] Śliwerski, J., Zimmermann, T. and Zeller, A.: When Do Changes Induce Fixes?, *Proceedings of the 2005 International Workshop on Mining Software Repositories (MSR2005)*, pp. 1–5 (2005).

[7] Yamashita, A. and Moonen, L.: Do developers care about code smells? An exploratory survey, *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE2013)*, pp. 242–251 (2013).

[8] Zhang, G., Peng, X., Xing, Z. and Zhao, W.: Cloning practices: Why developers clone and what can be changed, *Proceedings of the 28th International Conference on Software Maintenance (ICSM2012)*, pp. 285–294 (2012).

[9] 堀田圭佑，楊　嘉晨，肥後芳樹，楠本真二：粗粒度なコードクローン検出手法の精度に関する調査，情報処理学会論文誌，Vol. 56, No. 2, pp. 580–592 (2015).

[10] 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，Vol. 113, No. 159, pp. 127–132 (2013).

[11] 堀田圭佑，肥後芳樹，楠本真二：生成抑止，分析効率化，不具合検出を中心としたコードクローン管理支援技術に関する研究動向，コンピュータソフトウェア，Vol. 31, No. 1, pp. 14–29 (2014).

[12] 辻　健二，崔　恩瀞，吉田則裕，春名修介，井上克郎：コードクローン編集者数に着目した開発履歴の分析，情報処理学会研究報告，Vol. 2014-SE-186, No. 1, pp. 1–7 (2014).

[13] 森脇匠哉，堀田圭佑，井垣　宏，井上克郎，楠本真二：複数のプロジェクトを対象としたクローンの系譜にもとづくソースコード再利用分析手法の提案，電子情報通信学会技術研究報告，Vol. 114, No. 510, pp. 61–66 (2015).

[14] 森脇匠哉，井垣　宏，山中裕樹，吉田則裕，井上克郎，楠本真二：ソフトウェアリポジトリにおけるコードクローン作成者・利用者関係分析手法とその適用，電子情報通信学会技術研究報告，Vol. 113, No. 24, pp. 37–42 (2013).

[15] 神谷年洋，肥後芳樹，吉田則裕：コードクローン検出技術の展開，コンピュータソフトウェア， Vol. 28, No. 3, pp. 29–42 (2011).

[16] 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. 91, No. 6, pp. 1465–1481 (2008).