コードクローンのコミット時間の分析と調査

2020SE032 間瀬京太郎　2020SE075 渡辺達也
指導教員：井上 克郎

1　はじめに

近年，社会生活を支えているソフトウェアは巨大化し続けており，それらを構成しているソースコードも膨大なものとなっている．それに伴い，ソフトウェアの保守活動における負担も増加している．そのため，ソフトウェアの保守活動における支援や，負担軽減の研究が盛んに行われている．なかでも，コードクローンは保守活動の負担増加の一因となっている．

コードクローンに関しては多くの研究が行われており，現在までに，様々なコードクローンの検出手法が提案されている [1]．しかしながら，検出されたコードクローンの特性に関しての詳しい分析，特に，コードクローンのコミット時間についての分析はあまり行われていない．

本研究では，GitHub 上で公開されているプロジェクトのコードクローンを検出した後，コードクローンのコミット時間の調査を行い，コミット時間がコードクローンに与える影響を明らかにしていく．

本研究で作成したコードクローンのコミット時間の類似度と危険なコードクローンを求める分析ツールは，バグを引き起こす可能性のあるコードクローンの発見や，適切に管理されていないコードクローンの検出に役立つことが期待される．

2　既存研究

2.1　コードクローン

コードクローン（もしくはクローン）とは，ソースコード中に含まれる同一，または類似したコード片のことである [1]．互いにクローンの関係にあるコード片の組をクローンペア，クローンの関係にあるコード片の集合をクローンセットと呼び，それらを構成するコード片をクローン片と呼ぶ．

コードクローンは主に，コピー＆ペーストによるソースコードの再利用によって作成される [3]．ソースコード中にコードクローンが存在する場合，保守を複雑にしたり，予期しないバグを引き起こしたりする可能性がある．そのため，コードクローンが存在するソースコードに関しては，特に慎重に編集を行う必要がある．

2.2　コードクローン検出ツール

CCFinderX, Decard, Nicad など数多くのコードクローン検出ツールが開発されている [1]．なかでも，代表的なコードクローン検出ツールの 1 つとして，CCFinderSW がある [2]．

CCFinderSW は，コードクローンが検出されたファイルパス，ファイル名，コードクローンの開始行，終了行などの情報をファイルに出力することができる．また，実行する際に，最低トークン数，トークンの最低種類数，非繰り返し部分の最低割合などのオプションを指定することで，検出することができるコードクローンの大きさや種類を指定することが可能である．

本研究では CCFinderSW を用いてコードクローンの検出を行い分析を行う．CCFinderSW を実行すると，分析結果ファイルが出力される (図 1)．分析結果ファイルには，cloneID，ファイルパス，ファイル名，コードクローンの開始行，終了行の情報が出力される．

図 1 の場合，cloneID:3 として 2 つのクローン片が存在し，5 番目の ConversationManagerTest.java の 97 行目の 62 文字目から 118 行目の 16 文字目のコード片と，同じファイルの 112 行目の 62 文字目から 134 行目の 13 文字目のコード片が，クローンペアであることを示している．

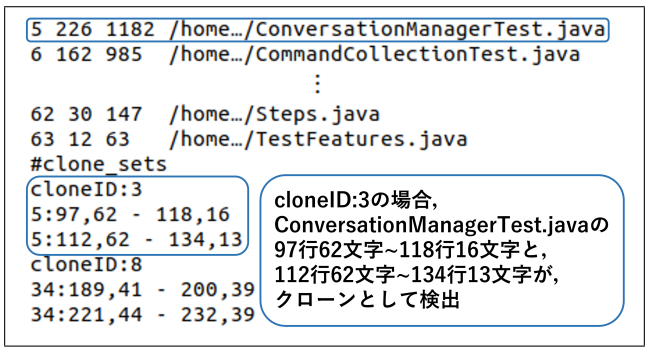

図 1　CCFinderSW の実行結果の例

2.3　コードクローンの作者に関する分析

検出されたコードクローンと作者の関係を調査するために，小島，佐藤らは [4] で GitHub 上の複数のプロジェクトに対して，CCFinderSW を用いてコードクローンの検出を行った後，git blame コマンドを用いて，最後にコードクローンを編集した作者を求める方法を提案した．

調査の結果，コードクローンを作成した作者が，同一である割合の平均値は 44.3% と分かった．また，複数の人物によって作成されたコードクローンは，1 人の人物によって作成されたものより，行数が長く，複雑なものが多いことも分かった．一方で，作成されるコードクローンの数や行数などの特性は，違いがないことが分かった．

この研究では，コードクローンの作者に関しての分析を行い，その特性の違いを明らかにしたが，コードクローンの作者以外にも，検出されたコードクローンの特性に影響を与える要因が存在するかどうかを調査するため，本研究では，GitHub 上のプロジェクトに対して，コードクローンのコミット時間に焦点を当てて調査を行う．

3 研究の方針

3.1 背景

前述のように，コードクローンの検出手法に関する研究は広く行われているが，検出されたコードクローンの性質に関する研究はあまり行われていない．とりわけ，検出されたコードクローンと，コミットされた時間の関係については研究されておらず，コミットされた時間がコードクローンに与える影響を明らかにしていくことで，ソフトウェアの全体像の解明に繋げる．

3.2 目的

コードクローンが存在する場合，互いにクローン関係にあるコード片に対して，修正が行われた際, クローン関係にある他のコード片に対しても, 必要であれば修正を行う．必要な修正がされていないなど，適切な管理がされていない場合，予期しないバグを引き起こす可能性がある．

本研究では，GitHub 上のプロジェクトのコミット時間を調査し，コードクローンが適切に管理されている場合と，そうでない場合の特性の違いを明らかにし，その成果をソフトウェアの保守の負担軽減やバグの追求に役立てる．

3.3 研究の概要

本研究では，まず GitHub 上のプロジェクトのコードクローンの検出を行う．次に，git log コマンドを実行し，コードクローンとして検出された部分のコミット時間を取得する．そして，コードクローンの各クローン片の 2 つからなる全ての組み合わせに対して，コミット時間が互いにどの程度一致しているかという類似度を，Jaccard 係数を用いて求めることにより，プロジェクト内のコードクローンが適切に管理されているかを確認する．

4 コミット時間の調査のための分析ツールの実現

4.1 ツールの概要

分析ツールは，CCFinderSW の分析結果ファイルを読み込み，git log コマンドを実行するための情報を抽出する．次に，git log コマンドを実行し，コードクローンのコミット時間を取得する．そして，コミット時間の簡約化を行った上で，コミット時間の類似度を Jaccard 係数で求める．Jaccard 係数は以下の数式で求めることができる．

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (1)$$

ここで A はコードクローン対を構成するコード片 S_A のコミット時間の集合 $\{t_{A1}, t_{A2}, ..., t_{An}\}$，$B$ は同様にコード片 S_B のコミット時間の集合 $\{t_{B1}, t_{B2}, ..., t_{Bm}\}$ とする．

4.2 コミット時間の特徴によるコードクローンの分類

分析を行うにあたり，コミット時間の特徴からコードクローンを以下の 6 つに分類を行った．

1. コミット時間が完全一致（perf sync）.
2. コミット時間が完全一致しない（no sync）.
3. コミット時間が途中まで一致（sync to mid）.
4. コミット時間が途中から一致（sync from mid）.
5. コミット時間が最初と最後は一致（sync first and last）.
6. いずれのパターンにも当てはまらない（other）.

この分類において，コードクローンは同時に編集されることが望ましく，同様に，最後のコミット時間も一致していることが望ましい．そのため，4，5，6 の 3 つに分類されたクローンペアに関しては，最後のコミット時間が互いに一致しておらず，修正が同時期に行われていない．そのため，適切な管理がされておらず，全体に悪影響を及ぼすコードクローンである可能性がある．

以上の理由から，最後のコミット時間が一致していないクローンペアを危険なコードクローンとし，類似度に加えて，上記の分類方法を用いて分析を行う．

4.3 分析手順

コミット時間の分析を以下の手順で行った (図 2).

1. 対象とするプロジェクトを GitHub より入手.
2. 最低トークン数 t を 100，トークンの最低種類数 tks を 12, 非繰り返し部分の最低割合 rnr を 0.5 に設定し，CCFinderSW を実行して，分析結果ファイルを入手.
3. 分析ツールを実行し，分析結果ファイルを読み込み, コードクローンごとの cloneID，ファイルパス，ファイル名，コードクローンの開始行，終了行を取得する.
4. ツール内から git log コマンドを実行し，cloneID ごとのコードクローンのコミット時間を取得する.
5. クローン片のコミット時間の簡約化を行う.
6. コミット時間の類似度を Jaccard 係数で求める.
7. コミット時間の特徴からコードクローンを分類.

4.4 コミット時間の簡約化

git log コマンドでコミット時間を取得する場合，通常は「Date: 2014-10-03 20:27:52」のように，1 日ごとに 1 秒単位まですべて出力される．しかしながら，本研究では，日，時，分，秒の情報を削除し，「Date: 2014-10」のように，月単位でコミット時間をまとめ，あるクローン片がコミットされた月に，クローン関係にある他のクローン片が，その月に 1 度でもコミットされていれば，コミット時間が一致していると判断する，

コードクローンはその性質上，コミット時間が時，分，秒の単位まですべて一致することはあまりない．また，日を跨いで修正を行う場合もある．そのため，コミット時間

をそのまま用いた場合，必要な修正がほぼ同等に行われているにも関わらず，コミット時間が一致しないために，類似度が低く出力されてしまう．

以上の理由から，コミット時間の簡約化を行い，月単位でのコミット時間の分析を行った．

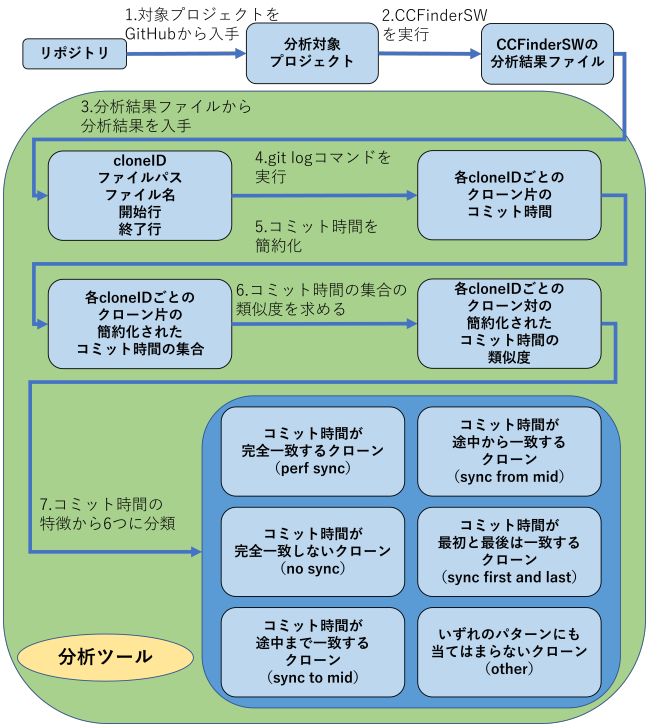

図 2　分析手順

4.5　分析例

クローン片の簡約化されたコミット時間，コミット時間の類似度，分類された項目が図 3 のように出力される．

図 3 の場合，クローン片が 3 つ存在するため，コミット時間の類似度が 3 通り出力される．例えば，クローン片 1 と 3 の類似度は 1.00 であるため，ほぼ同時期に修正が行われている．一方で，クローン片 1 と 2，2 と 3 は，類似度が 0.33 であるため，およそ 3 分の 2 の修正が同時期に行われておらず，修正が十分かどうかの検討が必要である．

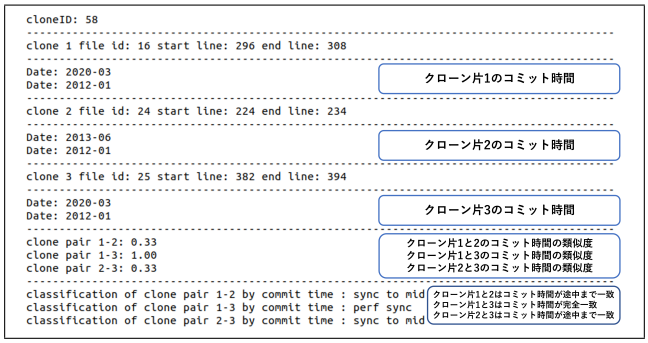

図 3　分析例

5　コミット時間の分析と調査

5.1　分析対象プロジェクト

分析対象プログラムとして，GitHub 上で公開されているプロジェクトを 40 個選択した．

5.2　リサーチクエスチョン

本研究では，コードクローンとコミット時間の特性を調べるために，3 つのリサーチクエスチョンを設けた．

RQ1：コードクローンのコミット時間の基本的な特性は何か

RQ2：プロジェクトごとのコミット時間の類似度はどのようになるか

RQ3：プロジェクト中にどのくらい危険なコードクローンが含まれているか

5.3　分析結果

5.3.1　RQ1

プロジェクトごとのコードクローンのコミット回数の分布を図 4 に示す．プロジェクトごとのコードクローンのコミット回数の平均値は，3 回から 4 回の間で最も多く分布していた．また，プロジェクトごとのコミット回数の中央値は，2 回から 3 回の間で最も多く分布していた．

5.3.2　RQ2

プロジェクトごとのコードクローンの類似度の分布を図 5 に示す．プロジェクトごとのコードクローンのコミット時間の類似度の平均値は，0.50 から 0.70 の間で多く分布していた．また，プロジェクトごとのコミット時間の類似度の中央値は，1.00 が最も多く分布していた．

5.3.3　RQ3

プロジェクトごとの危険なコードクローンの割合を図 6 に示す．最後のコミット時間が一致していない危険なクローンペアの割合は，20% から 30% のプロジェクトが最も多かった．また，半数のプロジェクトで，危険なクローンペアの割合が 20% を超えていることが分かった．

6　議論

6.1　考察

リサーチクエスチョンの結果から，コードクローンのコミット回数は，平均値が 3 回から 4 回，中央値が 2 回から 3 回が最も多いことが分かった．また，コミット時間の類似度は，全体的に高い水準で分布していることが分かった．

一方で，半数のプロジェクトで，危険なクローンペアが，20% 以上存在していることが分かった．しかしながら，危険なクローンペアが含まれている割合は，プロジェクトごとに異なることが分かった．

また，プロジェクトごとのコミット時間の類似度の平均値と，危険なコードクローンの割合の関係を図 7 に示す．結果として，コミット時間の類似度と危険なコードクロー

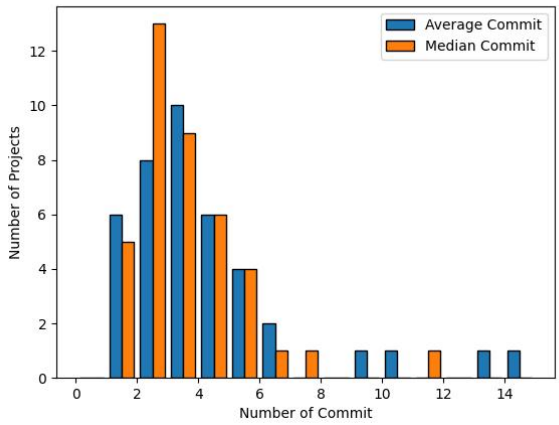

図 4　プロジェクトごとのコミット回数の分布

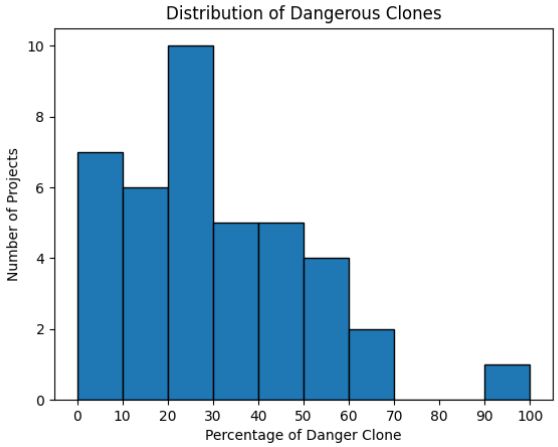

図 6　プロジェクトごとの危険なコードクローンの割合

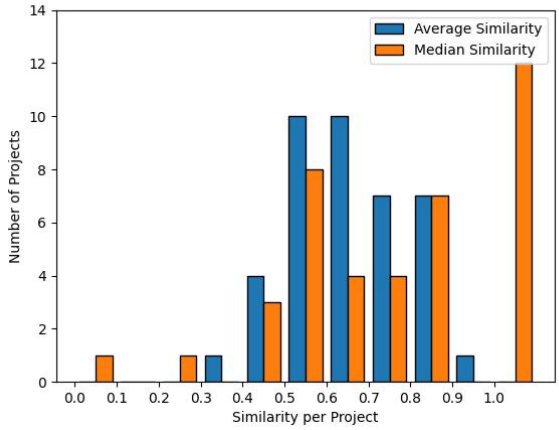

図 5　プロジェクトごとのコミット時間の類似度の分布

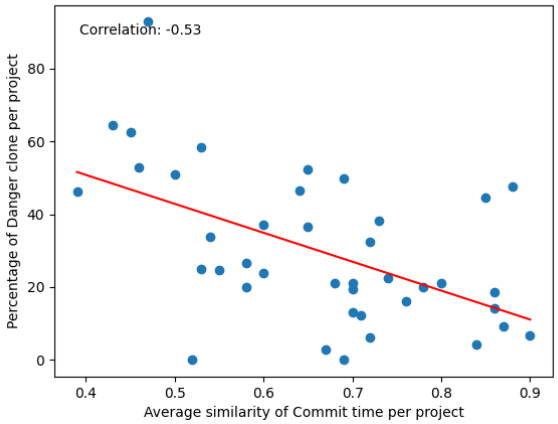

図 7　類似度と危険なコードクローンの割合

ンの割合の間には負の相関がみられた．そのため，必要な修正が行われていないなど，適切に管理されていないプロジェクトに関しては，同様に最後のコミット時間も一致していない可能性が高く，危険なコードクローンを多く含むプロジェクトであることが考えられる．

7　おわりに

本研究はコミット時間の調査を行い，コードクローンの特性を分析する方法を提案した．また，コミット時間の類似度を求め，悪影響を及ぼす可能性のある危険なコードクローンの分類を行う分析ツールを作成した．

今回，GitHub 上の 40 個のプロジェクトに対して，コミット時間の類似度や危険なコードクローンの割合を求め，コミット時間とコードクローンの関係を調査できた．

本研究の成果は，コピーされた時間から編集されるまでの時間がかなり離れている場合，作者に警告を出す機能などに応用できると考えている．また，分析ツールは，適切に管理されていない，もしくは，バグを引き起こす可能性のあるコードクローンの検出に役立つことが期待される．

今後の課題として，コミット時間の特徴から危険なコードクローンの分類を行ったが，それらの危険性について調査できていないため，危険なコードクローンと実際に発生するバグとの関係の調査を行っていきたい．

参考文献

[1] Katsuro Inoue, Chanchal K. Roy: ”Code Clone Analysis: Research, Tools, and Practices”, Springer, ISBN 978-981-16-1926-7, Aug., 2021.

[2] Yuichi Semura: CCFinderSW, https://github.com/YuichiSemura/CCFinderSW, 2019.

[3] Jeffrey Svajlenko, Chanchal K. Roy: ”A Survey on the Evaluation of Clone Detection Performance and Benchmarking”, arXiv-cs.SE, 2006.15682, 2020.

[4] 小島 愛由, 佐藤 優里愛: ”コードクローンの作者に関する定量的分析”, 南山大学理工学部 2022 年度卒業論文, 2023.