様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　２ 年　 ６ 月　 ８ 日現在

機関番号：１４４０１
研究種目：基盤研究(B)（一般）
研究期間：2017～2019
課題番号：１７Ｈ０１７２５

研究課題名（和文）ソフトウェア品質に悪影響を与えうるコードクローンの取得

研究課題名（英文）Retrieving code clones that are harmful for software quality

研究代表者
肥後　芳樹（Higo, Yoshiki）

大阪大学・情報科学研究科・准教授

研究者番号：７０４５２４１４

交付決定額（研究期間全体）：（直接経費）　　5,500,000円

研究成果の概要（和文）：ソフトウェア品質に悪影響を与えるクローンを容易に取得することを目的として，本研究では3つの研究課題に取り組んだ．1つ目は，自動生成ファイルの特定である．自動生成ファイルから検出されたクローンが分析を必要とするクローンの存在を隠してしまう，という状況を避けることできる．2つ目は，無害なクローンの特定である．無害であるクローンを事前に取り除くことができれば，開発者による判断が必要なクローンを削減できる．3つ目は，ギャップを含むクローンのクローンセットの特定である．分析が必要なギャップを含むクローンを，その情報を利用しやすい形態であるクローンセット形式で開発者に提示できる．

研究成果の学術的意義や社会的意義
本研究の成果は，ソフトウェア開発やソフトウェア工学の研究で役立つ．例えば，調査すべきコードクローンの数が減るため，開発者が1つずつ目視で調査を行えるようになる．分析例としては，本来はコードクローンを持たないはずのファイル間（モジュール間）にコードクローンがあることがわかれば，ソフトウェアの実装がその設計と乖離している問題の発見につながる．また，コードクローンの量やコードクローンの含有率をソフトウェアメトリクスとして用いることにより，問題のあるファイルやモジュールを発見することができる．分析例としては，コードクローンを多く含むファイルを特定し本当にその状態が正しいのかを調査できる．

研究成果の概要（英文）：In order to facilitate the acquisition of clones that adversely affect software quality, three research questions were addressed in this study. The first is the identification of automatically generated files. It avoids the situation where the clones detected in the autogenerated file hide the presence of clones that need to be analyzed. The second is the identification of harmless clones. Being able to remove harmless clones beforehand would reduce the number of clones that need to be determined by the developer. The third is the identification of the clone set of clones containing the gap. It will be possible to present clones containing gaps that need to be analyzed to developers in a clone set form, a form in which the information is easily accessible.

研究分野：ソフトウェア工学

キーワード：コードクローン　ソースコード解析　ソフトウェアリポジトリマイニング

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９（共通）

１．研究開始当初の背景

コードクローンとは，ソースコード中に存在する類似したコード断片を表す言葉である．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要因となりうる．多くのコードクローンはコピーアンドペーストを用いた開発により発生する．コピーアンドペーストにより，必要な機能を短い時間で実装することができるが，コピー元のコードにバグが含まれていた場合，そのバグが拡散してしまう．また，コードクローンになっているコードに対して変更を加える場合，それと対応するコードクローンを開発者が認識していない場合は，修正漏れが発生してしまう．

上述のようにコードクローンは問題を含みうるコードである．つまり，コードクローンを検出しそれらの調査することにより，ソフトウェアの品質に悪影響を与える可能性が高いコードを発見できる．ここで潜在的なバグが存在しているコードとは，すでにバグは発生しているが開発者やユーザはまだそれに気がついていない状態や，将来バグが発生する可能性が高いコードを指す．プログラムのソースコードから目視によりコードクローンを検出することは現実的ではないため，ツールを用いた自動的な検出が行われる．これまでに数多くのコードクローン検出手法が提案されており，無償で利用できる検出ツールもある．

既にコードクローンに関するさまざまな研究が行われており，本研究に関連する範囲で列挙すると下記の事項が明らかになっている．

- ソフトウェアの品質に悪影響を与えるコードクローンは確かに存在しているが，全てのコードクローンが悪影響を与えるわけではない．
- 検出ツールを用いると非常に多くのコードクローンが見つかる傾向にあるため，人間が検出されたコードクローンを 1 つずつ目視で調査していくことは現実的ではない．例えば，最もよく利用される Web サーバである Apache HTTPD（約 236,000 行）には約 72,000 のクローンペアが存在する（2016/10/20，研究代表者が HTTPD の最新版 2.4.23 のソースコードに対して，最も広く利用されている検出ツール CCFinder を利用して得た数値である）．
- クローンペア内の名前の整合性をチェックすることにより，潜在的なバグが存在しているコードを発見することはできる．しかし，将来バグが発生する可能性が高いコードを知ることはできない．

２．研究の目的

研究代表者が所属する研究グループは，これまでにコードクローン検出ツールを企業に無償で配布している．配布した組織の数は 200 を超える．しかしながら，継続的にツールを使っている組織の割合は決して高いとはいえない．その主な原因は，「非常に多くのコードクローンが見つかりどう対処していいのかわからない」や「検出したコードクローンをいくつか調べてみたが特に問題と思われるコードはそれほど見つからなかった」などである．このような経験から，品質に悪影響を与えうるコードクローンのみを開発者に与えることが（しょうもないクローンを見せないことが）非常に大切であると研究代表者は実感している．

本研究では，検出ツールにより検出された大量のコードクローンから，ソフトウェアの品質に悪影響を与える可能性が高いコードクローンを自動的に取得する方法の考案とそれを実現した実用性の高いシステムの開発を行う．本研究の成果は，例えば以下のように利用できる．

- 調査すべきコードクローンの数が減るため，開発者が 1 つずつ目視で調査を行えるようになる．例えば，本来はコードクローンを持たないはずのファイル間（モジュール間）にコードクローンがあることがわかれば，ソフトウェアの実装がその設計と乖離している問題の発見につながる．
- コードクローンの量やコードクローンの含有率をソフトウェアメトリクスとして用いることにより，問題のあるファイルやモジュールを発見することができ．例えば，コードクローンを多く含むファイルを特定し本当にその状態が正しいのかを調査できる．
- ソフトウェアの品質に関する他の研究で利用できる．例えば，ソフトウェアのバグの発生はパレートの法則（80 対 20 の法則）に従うことがわかっている．つまり，80%のバグは 20%のコードにおいて発生する．このことからバグを多く含むコードを予測する研究が活発に行われている．多くの研究はファイル単位での予測，つまりバグを多く含むファイルを予測する．それらの予測には，ファイルのサイズや複雑度，ファイルを変更した開発者数，これまでの改版数等の種々の情報が利用されているが，まだ十分な精度には達成されていない．本研究で得られたコードクローンの情報を追加で利用することにより，より精度の高い予測が達成できる可能性がある．

３．研究の方法

ソフトウェア品質に悪影響を与えるコードクローンを容易に取得するために，本研究では 3.1～3.3 に示す 3 つテーマに取り組んだ．

３．１．コードクローンを検出する必要がない自動生成ファイルの自動特定

コードクローン検出において，検出対象のソフトウェアに含まれるソースコードファイルの中には自動生成ファイルが含まれており，検出されたコードクローンの分析を行う際に自動生成ファイルが弊害となることがある．これは自動生成ファイルから検出されるコードクローンは自動生成ツールにより生成されたコードであり，コピーアンドペーストによって生成されたコードではないためである．実際に既存研究ではライブラリ化可能なコードクローンを検出する研究で自動生成ファイルを対象から手動で除去して実験している．

通常，自動生成ファイルにはそれ自身が自動生成ファイルであると明示するためのコメント文が残されている．ゆえに，そのようなソースコードに対しては grep コマンドを用いれば特定および除去することができる．しかしソースコードを修正していく過程でそのコメント文が消されてしまう場合がある．その場合 grep コマンドによる特定は難しく，またコメント文が消された自動生成ファイルを目視などで特定するのは時間的コストが大きい．したがって，そのようなコメント文が消された場合も含めて自動生成ファイルを自動的に特定することが必要となる．

コメント文が消された自動生成ファイルを自動的に特定するためには，自動生成ファイルにおける何らかの特徴を用いる必要がある．しかし任意の自動生成ファイルに共通する特徴を目視などで発見するのは困難である．本研究では，N-gram 言語モデルを用いてコメント文の有無にかかわらず自動生成ファイルか否かを自動的に判定する手法を提案した．N-gram 言語モデルとは，既存のコードの字句の並びからモデルを生成することにより，未知のファイルのモデルに対する自然さ，すなわちモデルらしさを数値として出力する統計的言語モデルである．提案手法では自動生成ファイルだと判明しているソースファイル，及び自動生成でないファイルと判明しているソースファイルをそれぞれ収集する．収集したソースファイルから，自動生成ファイルの言語モデルと自動生成でないファイルの言語モデルを生成する．それらを用いて未知のソースコードの自動生成ファイルとしての自然さと自動生成でないファイルとしての自然さをそれぞれ求め，これらを比較することによって自動生成ファイルかどうかを判定する．

３．２．無害なコードクローンの自動特定

既存研究では，機械学習を用いることでクローンセット（互いにコードクローンとなっているコード片の集合）が有害か否かを判定する手法を提案している．この手法は複数人の開発者に検出されたクローンセットの一部を有害か判定してもらい，その結果を学習することで残りのクローンセットが有害であるか判定するというものである．しかしクローンセットが有害であるかの基準はプロジェクトやその開発者によって異なるため，プロジェクトの開発者に判定してもらう必要がある．そのため，学習用のデータセットを構築するのに膨大なコストがかかる．一方でどのプロジェクトにおいても無害であることが自明なクローンセットが存在する．例えば言語の仕様から生じる定型処理のクローンは長期間クローンとして存在しやすいため，ソフトウェア開発に悪影響を及ぼさない無害なクローンである．無害であることが自明なクローンセットを事前に取り除くことができれば，開発者による判断が必要なクローンセットを削減することができる．また言語の仕様から生じる定型処理のクローンはプロジェクトによらず似た処理であるため，複数プロジェクトにまたがって類似した記述がされやすい．

複数プロジェクトから別々にコードクローンを検出し，検出されたクローンセット間の類似度を用いることで，無害であることが自明なクローンセットと判断が必要なクローンセットを分類することが可能であると研究代表者は考えた．本研究ではクローンセットを自動で分類する手法を提案した．自動分類はクローンセット間の類似度に着目したグラフを作成し，クリークを形成するか否かで分類する．一つのクローンセットをグラフの一頂点とし，クローンセット間の類似度が閾値以上の場合辺を存在させる．これにより類似しているクローンセットは辺で結ばれ，互いに類似し合っているクローンセット間でクリークを形成する．複数プロジェクトにまたがってクリークを形成するクローンは類似した定型処理のクローンであると考えられるため，無害なクローンとみなすことが可能となる．

３．３．ギャップを含むコードクローンのクローンセットの特定

クローンの中でも文の挿入や削除などの編集がなされたクローンは，ギャップを含むクローンと呼ばれている．既存研究ではギャップを含むクローンはソフトウェアに存在するクローンで最も多く存在すると報告されている．したがってギャップを含むクローンは最も検出の必要が

あるコードクローンである．ギャップを含むクローンを検出可能な検出器も多数開発されている．しかしこれらの検出器の多くはクローンペアの検出のみ可能であり，クローンセットの検出が可能な検出器は限られている．例えば，SourcererCC は大規模なソフトウェアに対してギャップを含むクローンのペアを検出できるが，クローンセットの検出はできない．一方 CCFinder は suffix tree アルゴリズムによりクローンセットを検出するが，ギャップを含むクローンは検出できない．iClones は suffix tree アルゴリズムを拡張してギャップを含むクローンセットを検出するが，検出には膨大なメモリを必要とし，ギャップが大きいクローンは検出できない．NiCAD はギャップを含むクローンセットを検出できる検出器であるが，クローンペアで連結しているクローンを全て 1 つのクローンセットとみなす．これによりクローンセット内に全く類似しないクローンが存在する場合があり，検出された全てのクローンが 1 つのクローンセットとして検出される可能性がある．そのため任意の検出器で検出されたギャップを含むクローンペアからクローンセットを検出する手法が必要である．

そこで本研究ではギャップを含むクローンペアからギャップを含むクローンセットを検出する手法を提案した．提案手法では極大クリーク列挙アルゴリズムを利用する．クリークとは任意の 2 頂点間に辺が存在する部分グラフである．提案手法は頂点をクローン，辺をクローンペアとみなして生成されたグラフから極大クリーク列挙を行う．得られるクリーク内の頂点集合は互いに類似しているクローンとなる．そこで提案手法ではこの頂点集合に含まれるコード片を 1 つのクローンセットとして出力する．

４．研究成果

ここでは，取り組んだ 3 つの研究テーマについて，その評価結果を述べる．

４．１．コードクローンを検出する必要がない自動生成ファイルの自動特定

4 種類の自動生成ファイル（ANTLR，JavaCC，JFlex，SableCC）を収集し，それらを対象に評価実験を行った．その結果，全ての場合で適合率，再現率ともに 99%以上と，高い精度で自動生成ファイルを特定できていることを確認した．また 4 種類の自動生成ファイルを混ぜた場合の評価実験も行った．その結果に関しても，適合率，再現率ともに 98%以上と，高い精度で自動生成ファイルを特定できていることを確認した．加えて別種類の言語の自動生成ファイルを特定できるかの評価実験を行った．その結果，適合率，再現率ともに 98%以上と，高い精度で TypeScript から生成された JavaScript を特定できていることを確認した．

４．２．無害なコードクローンの自動特定

提案手法を 4 つのプロジェクト（JFreeChart，Eclipse.jdt.core,Lucene，Tomcat）に対して適応した結果，3,993 種類のクローンセットから 145 個の言語固有のコードクローンと 507 個のプロジェクト固有のコードクローンを抽出することができた．また抽出された 言語固有のコードクローンの一部を確認するとコンストラクタや多くのクラスで定義されるメソッドなど言語固有な実装が分類されていることを確認した．

４．３．ギャップを含むコードクローンのクローンセットの特定

実験では提案手法を実装したツールを 5 つのオープンソースソフトウェア Eclipse.jdt.core，Guava，JFreeChart，RxJava，Tomcat）に対して適用した．その結果，各プロジェクトからギャップを含まないクローンセットが 60 個から 907 個検出され，ギャップを含むクローンセットは 382 個から 2,625 個検出された．さらに実行時間に関しても各プロジェクトから 130 ミリ秒以下で高速に検出できた．また他のオープンソースソフトウェアから提案手法によって検出されたクローンセットを用いてリファクタリングを行った．リファクタリングによる修正内容をプルリクエストで開発者に提案したところ，2020 年 3 月 13 日時点で 7 個}のプルリクエストのうち 2 個がマージされた．

５．主な発表論文等

〔雑誌論文〕　計5件（うち査読付論文　5件／うち国際共著　0件／うちオープンアクセス　0件）

１．著者名 Higo Yoshiki、Hayashi Shinpei、Kusumoto Shinji	４．巻 165
２．論文標題 On tracking Java methods with Git mechanisms	５．発行年 2020年
３．雑誌名 Journal of Systems and Software	６．最初と最後の頁 110571～110571
掲載論文のDOI（デジタルオブジェクト識別子） 10.1016/j.jss.2020.110571	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 林 純一、肥後 芳樹、楠本 真二	４．巻 J102-D
２．論文標題 メソッドレベルセマンティックバージョニングの提案と評価	５．発行年 2019年
３．雑誌名 電子情報通信学会論文誌D 情報・システム	６．最初と最後の頁 730～739
掲載論文のDOI（デジタルオブジェクト識別子） 10.14923/transinfj.2018JDP7074	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 土居真之，肥後芳樹，有馬諒，下仲健斗，楠本真二	４．巻 60
２．論文標題 言語モデルによるソースコードの「自然さ」を利用した自動生成ファイルの特定	５．発行年 2019年
３．雑誌名 情報処理学会論文誌	６．最初と最後の頁 542--650
掲載論文のDOI（デジタルオブジェクト識別子） なし	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

１．著者名 幸佑亮，肥後芳樹，楠本真二	４．巻 59
２．論文標題 多粒度コードクローンの検出と評価	５．発行年 2018年
３．雑誌名 情報処理学会論文誌	６．最初と最後の頁 1192--1202
掲載論文のDOI（デジタルオブジェクト識別子） なし	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

1．著者名 有馬諒，肥後芳樹，楠本真二	4．巻 35
2．論文標題 不適切に分割されたコミットに関する研究	5．発行年 2018年
3．雑誌名 コンピュータソフトウェア	6．最初と最後の頁 164--170
掲載論文のDOI（デジタルオブジェクト識別子） なし	査読の有無 有
オープンアクセス オープンアクセスではない、又はオープンアクセスが困難	国際共著 －

〔学会発表〕　計22件（うち招待講演　0件／うち国際学会　10件）

1．発表者名 Masayuki Doi
2．発表標題 A Code Clone Curation --Towards Scalable and Incremental Clone Detection--
3．学会等名 the 7th International Workshop on Quantitative Approaches to Software Quality（国際学会）
4．発表年 2019年

1．発表者名 Tasuku Nakagawa
2．発表標題 How Compact Will My System Be? A fully-automated way to calculate LoC reduced by clone refactoring
3．学会等名 the 26th Asia-Pacific Software Engineering Conference (APSEC2019)（国際学会）
4．発表年 2019年

1．発表者名 Tetsushi Kuma
2．発表標題 Improving the Accuracy of Spectrum-based Fault Localization for Automated Program Repair
3．学会等名 the 28th International Conference on Program Comprehension（国際学会）
4．発表年 2020年

１．発表者名
Akira Fujimoto

２．発表標題
Staged Tree Matching for Detecting Code Move across Files

３．学会等名
the 28th International Conference on Program Comprehension（国際学会）

４．発表年
2020年

１．発表者名
Junnosuke Matsumoto

２．発表標題
GenProg Meets Cluster Computing

３．学会等名
the 10th International Workshop on Empirical Software Engineering in Practice（国際学会）

４．発表年
2019年

１．発表者名
九間 哲士

２．発表標題
欠陥限局に適したテストスイートに関する考察

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2020年

１．発表者名
藤本 章良

２．発表標題
抽象構文木を利用したファイル間のコード移動検出

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2020年

１．発表者名
土居 真之

２．発表標題
複数プロジェクトから高速にコードクローンを検出するキュレーションの提

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2019年

１．発表者名
九間 哲士

２．発表標題
自動バグ限局に適したテスト自動生成に向けて

３．学会等名
ソフトウェアエンジニアリングシンポジウム2019

４．発表年
2019年

１．発表者名
土居 真之

２．発表標題
大規模なプロジェクト群を対象とした高速にコードクローンを検出するキュレーションの提案

３．学会等名
ソフトウェアエンジニアリングシンポジウム2019

４．発表年
2019年

１．発表者名
松本淳之介

２．発表標題
分散処理を用いた自動プログラム修正の高速化

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2019年

１．発表者名
Junnosuke Matsumoto

２．発表標題
Beyond GumTree: A Hybrid Approach to Generate Edit Scripts

３．学会等名
the 16th International Conference on Mining Software Repositories (MSR2019)（国際学会）

４．発表年
2019年

１．発表者名
Junichi Hayashi

２．発表標題
Impacts of Daylight Saving Time on Software Development

３．学会等名
the 16th International Conference on Mining Software Repositories (MSR2019)（国際学会）

４．発表年
2019年

１．発表者名
土居真之

２．発表標題
コードクローン間の類似度に基づく無害なコードクローンの自動判定手法

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2019年

１．発表者名
中川将

２．発表標題
コードクローンに対する集約結果に基づいた削減可能なソースコード行数の測定手法

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2019年

１．発表者名
松本淳之介

２．発表標題
行単位の差分情報を考慮した抽象構文木のノード単位の差分出力

３．学会等名
電子情報通信学会ソフトウェアサイエンス研究会

４．発表年
2019年

１．発表者名
Ryo Arima

２．発表標題
A Study on Inappropriately Partitioned Commits --How Much and What Kinds of IP Commits in Java Projects? --

３．学会等名
the 15th International Conference on Mining Software Repositories (MSR2018)（国際学会）

４．発表年
2018年

１．発表者名
Ryo Arima

２．発表標題
Toward Refactoring Evaluation with Code Naturalness

３．学会等名
the 26th International Conference on Program Comprehension (ICPC2018)（国際学会）

４．発表年
2018年

１．発表者名
Masayuki Doi

２．発表標題
On the Naturalness of Auto-generated Code --Can We Identify Auto-Generated Code Automatically? --

３．学会等名
the 26th International Conference on Program Comprehension (ICPC2018)（国際学会）

４．発表年
2018年

１．発表者名 有馬 諒
２．発表標題 ソースコードの“自然さ”を用いたリファクタリング評価手法の検討
３．学会等名 電子情報通信学会ソフトウェアサイエンス研究会
４．発表年 2018年

１．発表者名 土居 真之
２．発表標題 ソースコードの「自然さ」を利用した自動生成ファイルの特定
３．学会等名 電子情報通信学会ソフトウェアサイエンス研究会
４．発表年 2018年

１．発表者名 有馬 諒
２．発表標題 リファクタリングによる自然さの変化に関する調査 --自然さによるリファクタリング支援を目指して--
３．学会等名 ソフトウェアエンジニアリングシンポジウム2018
４．発表年 2018年

〔図書〕　計0件

〔産業財産権〕

〔その他〕

―

６．研究組織

	氏名 （ローマ字氏名） （研究者番号）	所属研究機関・部局・職 （機関番号）	備考