ソースコードの削減可能量計測ツールの開発

Developing a tool for estimating the amount of refactorable clones

上村 恭平 * 吉田 則裕 † 崔 恩瀞 ‡ 飯田 元 § 曲 生国 ¶ 秋庭 真一 ‖

あらまし ソースコードの品質診断サービスやマイグレーションサービスでは，コードクローン検出ツールの出力からコードクローン量を計測し，サービス利用者に提示することで，コードクローン削減の必要性の判断を支援する．しかし，マイグレーション事業を進めるなかで，サービス利用者はコードクローン量だけではなく，コードクローン削減の判断のために削減可能なコードクローン量や類似プログラム集合の数を必要とすることがわかってきた．本研究では，コードクローン検出ツールの出力を利用して，これら 2 つの数値を求めるツールを開発した．

1 はじめに

ソフトウェア保守を困難にする要因の 1 つとして，コードクローンが指摘されている [1] [2]．コードクローンとは，ソースコード中のコード片のうち，同一または類似したコード片を持つものを指す [3]．コードクローンは，既に開発されたコード片のコピーとペーストによる再利用や定形処理の実装などが理由で作成される [4] [5]．特に，Linux や JDK (Java Development Kit) などの大規模ソースコードには，大量のコードクローンが含まれていることが報告されている [3] [6]．

これまでに，ソースコード中のコードクローンを自動検出するツールが数多く提案されてきた [3] [6] [7]．これらコードクローン検出ツールは，ソースコードをツール毎に決められているプログラム表現（例えば，トークン列や構文木）に変換し，そのプログラム表現上において，等価な部分をコードクローンとして検出する．

コードクローン検出ツールを用いることで，ソースコードに含まれているコードクローンの行数やトークン数を容易に計測できる [3]．近年，コードクローン分析のビジネス化が進んできており，マイグレーションサービス [1] やソースコードの品質診断サービス [2] では，コードクローン量を計測し，サービス利用者に提示することで，コードクローン削減の必要性の判断を支援する．

しかし，マイグレーションサービス事業を進めるなかで，サービス利用者はコードクローン量だけではなく，コードクローン削減の判断のために下記数値を必要とすることがわかってきた．

(A) 全コードクローン量のうち，削減可能なコードクローン量
(B) 類似プログラム集合の数

(A) が必要な理由は，コードクローン検出ツールの出力の中には，同一トークン列が複数のコードクローンに含まれている場合があり，全コードクローンの行数の和を削減可能量と見積もると，過剰な見積もりになってしまう場合があるからである．

*Kyohei Uemura, 奈良先端科学技術大学院大学
†Norihiro Yoshida, 名古屋大学
‡Eunjong Choi, 奈良先端科学技術大学院大学
§Hajimu Iida, 奈良先端科学技術大学院大学
¶Shokoku Kyoku, 日立製作所
‖Shinichi Akiba, 日立製作所

[1] http://www.hitachi-gp.co.jp/service/migration/

[2] http://www.exmotion.co.jp/service/tool-exquto.html

そのため，同一トークン列が複数のコードクローンに含まれている場合を考慮した，削減可能なコードクローン量に見積もりが求められていることがわかってきた．(B) が必要な理由は，類似プログラム集合は類似した機能を実装している可能性が高く，不一致部分が含まれていたとしても，集約の対象になりうるため，コードクローン削減の必要性の判断に影響を及ぼすことがわかってきたからである．

本研究では，コードクローン検出ツール CCFinder [3] の出力を利用して，上記 (A)(B) を求めるツールを開発した．また，本ツールをオープンソースソフトウェアに適用した．

2　コードクローンとは

一般的に，類似機能の追加などを目的に，コードの一部をコピーアンドペーストすることでコードクローンが作られる [4] [5]．コードクローンを構成するコードの断片をクローン片，類似する 2 つのクローン片の組をクローンペア，同一のクローン片を含む複数のクローンペアをクローンセットと呼ぶ．あるコード片を修正するとき，そのコード片がクローンセットの一部であると，そのクローンセットに含まれるすべてのコード片を修正する必要がある．そのため，コードクローンはソフトウェアの保守性や品質を低下させる要因の 1 つであると言われている [1] [2]．こうした問題を解決するために，コードクローンの検出手法 [3] [6] [7] やコードクローンの除去支援技術 [4] [8]，一貫した修正支援技術 [9] [10] などが研究されている．

コードクローン検出手法はツールとして実装されており，公開されているものもある．代表的なコードクローン検出ツールとして神谷らの開発した CCFinderX が挙げられる [3]．CCFinderX は C/C++,Java,C#,Cobol のソースコードをトークン化し，トークンの並びが閾値以上の長さで一致する箇所をコードクローンとして検出する．

CCFinderX のコードクローンの検出結果はクローンペアの並びとして出力される．クローンペアは 2 つのクローン片から構成され，ファイルパスと始点と終点のトークン番号で各クローン片の位置情報を示す．クローンペアにはそれぞれクローン ID が割り振られるが，この際同一のクローンセットを構成するクローンペアには同一のクローン ID が割り振られる．また，トークン化の際に各ソースファイル毎にプリプロセスファイルが作成され，このファイルにはトークン番号と，ソースコード上の行番号やファイル先頭からの文字数の対応などが記録されている．CCFinderX は GUI フロントエンドである GemX が含まれており，GemX はこれらの情報を基にグラフィカルにソースコード上のクローンペアの対応関係などを表示する機能を持つ．

3　提案ツール

マイグレーション事業においてコードクローンを削減する必要性を判断するため，CCFinderX のコードクローン検出結果を元に下記の機能を実現するツールを実装した．

1. 重複のないコードクローンの検出
2. 類似プログラム集合の特定

以降，本章では各機能の目的と実現方法を説明した後，実装したツールを紹介する．

3.1　重複のないコードクローンの検出

一般に，CCFinderX などのコードクローン検出ツールの検出結果はクローンペアが互いに重複して同じ箇所を示すことがある．検出結果に重複箇所が含まれると，その分ソースコードの削減可能量が多く計測されてしまう．ソースコード削減可能量の計測を正確に行うために，CCFinderX の検出結果から重複部分を除外する機能を開発ツールに実装した．本機能は，石津らの手法 [11] に基づいて実装を行った．

重複のないコードクローンのリストの作成手順は次の通りである．まず，すべてのクローンペアのリストをトークン長の降順でソートする．その後，先頭から順にクローンペアを走査し，リスト上の自身より前にあるクローンペアと位置が重複している場合，そのクローンペアをリストから除外することで重複のないクローンペアのリストを作成する．クローンペアは3件以上重複している場合があるが，この手順で重複を除外することで，最もトークン長が長いクローンペアが残ることになる．

3.2 類似プログラム集合の特定

ソースコード中には，ファイル全体が類似しているプログラムが複数存在する場合がある．このような全体が類似するプログラム群を類似プログラム集合という．この類似プログラム集合に含まれるファイルを1つにまとめることで，効率的にソースコードの量を削減することができる．そのため，プログラムに含まれるファイルから類似するファイルの組を特定することが重要である．この際，複数の類似プログラム集合において各集合に含まれるファイルが重複していない必要がある．あるファイルが複数の類似プログラム集合に含まれている場合，削減可能量を多く見積もることになる．

共同研究先で利用されているコードクローン検出ツールであるCCFinderXはファイル中の類似するコードの断片をクローンセットとして検出するが，類似するプログラム集合を特定する機能までは有していない．そこで，重複のない類似ファイル集合を特定し，集合に含まれるファイルを集約した時の削減可能量を計測する機能を実装した．

類似プログラム集合の特定と削減可能量の計測手順を以下に示す．まず，すべてのファイル中から2つのファイル A，B を取る全組み合わせに対して，そのファイル間のクローン率 RC_{AB} および RC_{BA} を下記式1に従って計算する．

$$RC_{AB} = \frac{CLEN_{AB}}{LEN_A} \tag{1}$$

$CLEN_{AB}$ ： ファイルA，B間のコードクローンのトークン長
LEN_A ： ファイルAのトークン長

RC_{AB} あるいは RC_{BA} のどちらかが閾値より大きいとき，ファイル A とファイル B は類似していると判断する．次に，ファイルサイズを降順で並べ，先頭から順にファイルを走査し，次の手続きを行う．ファイル N に対して類似するファイルでかつ，まだいずれの類似プログラム集合にも含まれないファイルを集合 G_N に加える．この処理を繰り返すことですべての類似プログラム集合を特定する．

集合に含まれるファイルを全て集約した場合，集合に含まれる最も大きなファイルが残ると考えられる．従って，集合に含まれるファイルのサイズの合計から，最も大きなファイルのサイズを引いた値を削減可能量とする．ファイルサイズの計測方法には文字数，トークン数，行数など複数の計測方法があるが，今回実装したツールでは行数を用いた．

3.3 ツール概要

類似プログラム集合の特定機能，および重複のないコードクローンの検出機能をもつ，削減可能量計測ツールをC#でWindowsフォームアプリケーションとして作成した．

類似プログラム集合の特定機能における入力は，計測対象とするディレクトリ，結果の保存先，類似度の閾値の3つである．出力は，各集合に含まれるファイル名と行数を記述したテキストファイルが保存される．

重複のないコードクローンの検出機能においては，計測対象のディレクトリと結果の保存先を入力とする．検出結果は通常のCCFinderX，GemXが出力するccfxd

表 1: 分析対象

	バージョン	ファイル数	LoC
Linux Kernel	4.4.4	39485	18883827
Firefox	3.0.5	5456	3761792
Chrome	154.36	6403	1853757

表 2: 重複の除去によるクローンの量の変化

		総トークン長	クローンの数	クローン長	クローン率	重複率
Linux	重複有り	54,200,309	120,252	8,363,566	0.154	0.694
	重複無し		39,085	2,562,060	0.047	
Firefox	重複有り	10,503,569	24,790	1,826,156	0.174	0.634
	重複無し		8,177	668,829	0.064	
Chrome	重複有り	6,083,529	10,541	2,510,943	0.413	0.862
	重複無し		3,030	346,005	0.057	

ファイルと同様の形式で出力される．従って，CCFinderX や GemX から開き，コードクローンの分析や，削減可能量の計測を行うことができる．

4 分析事例の紹介

本章では開発したツールを用いて削減量の計測を行った分析事例を紹介する．分析の対象を表 1 に示す．Firefox については，CCFinderX がヘッダファイルのパースに失敗しコードクローン検出ができないため，ヘッダファイルを除くソースコードを対象としている．CCFinderX の設定は規定値を用いた [3]．

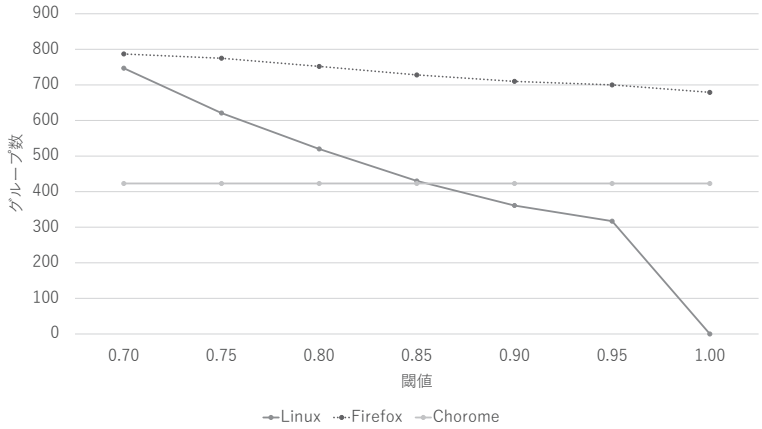
4.1 重複のないコードクローンの事例

各ソフトウェアの重複を含むコードクローンの量と，重複を除外した場合のコードクローンの量の変化を調査した．CCFinderX の出力するコードクローンの情報はトークンを単位としているため，コードクローンの長さや割合はトークン長で計算している．調査結果を表 2 にまとめる．いずれのソフトウェアも，コードクローンの 6-8 割以上が重複している．このような重複を含む検出結果を基に削減可能量を見積もると，見積もり値と実際に削減できる量が大きく乖離してしまう．重複を含む場合と取り除いた場合を比較すると，コード全体を示すコードクローンの割合は 1-3 割程度減少することが確認できる．開発したツールの検出結果を用いることで，より正確に削減可能量を見積もることができる．

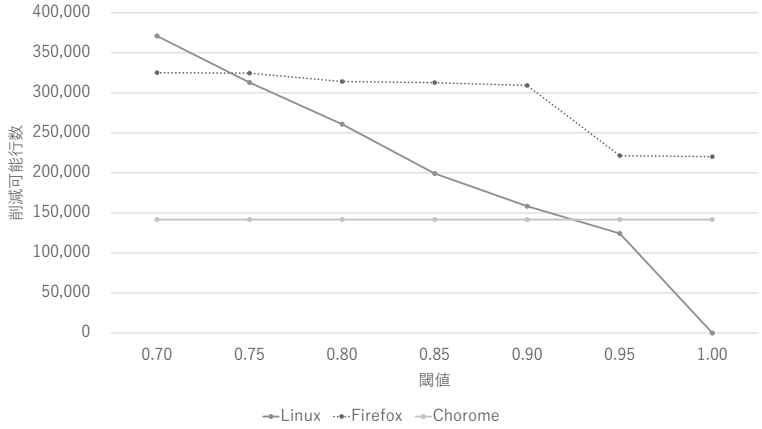
4.2 類似プログラム集合の事例

類似プログラム集合の特定を閾値を 0.70 から 0.05 刻みで 1.00 まで変化させながら実行した際の集合の数の変化と削減可能行数の変化を図 1 に示す．Chrome は 0.70 から 1.00 の範囲で集合の数，削減可能行数共に変化が無く一定である．つまり，Chrome において類似ファイルグループは類似度が 1.00 のファイルのみで構成されている．Chrome とは対照的に，Linux では，閾値 1.00 の時の集合の数は 0 個となる．Firefox は閾値 0.90 と 0.95 の間で集合の数の変化に比べて削減可能行数が著しく変化していることが確認できる．これは類似度 0.95 以上のファイルは非常に小さいファイルのみであることを示している． 次に閾値を 0.85 とした時の，削減対象とする類似プログラム集合の数に対する，実際に削減できる割合の推移を図 2

[3]Minimum Clone Length : 50, MinimumTKS : 12, Shaper Level : Soft Shaper, P-match Application : use, Prescreening Application : don't use



(a) 閾値の変化に対する集合の数の変化



(b) 閾値の変化に対する削減可能行数の変化

図 1: 閾値の変化に対する集合の数と削減可能行数の変化

に示す．類似プログラム集合は削減可能総行数で降順にソートとした順に選択するとする．いずれのソフトウェアでも上位 30 個の類似プログラム集合が削減可能量の 5 割を占めていることが確認できる．

5　おわりに

本研究では，ソースコードマイグレーション事業におけるコードクローン削減の判断のためのツールを開発した．開発したツールは重複を含まないコードクローンの検出と，類似プログラム集合の特定という 2 つの機能で構成されている．

本論では Linux Kernel，Firefox，Chrome を対象に開発ツールを用いた分析を行った．類似プログラム集合の特定機能による分析では，削減可能量全体の 5 割がコード量でソートした上位 30 件の集合に含まれることが確認できた．重複のないコードクローンの検出機能に関する分析として，重複の有無によるコードクローン量の変化について観察した．その結果，重複を除外した場合，全体のクローン率が 10-25%割程度低下することが確認できた．

今後の課題として，今回の分析を開発ツール上で行えるよう GUI 機能の拡張を行うことで，利便性を向上させることができると考える．また，今回重複のないコー

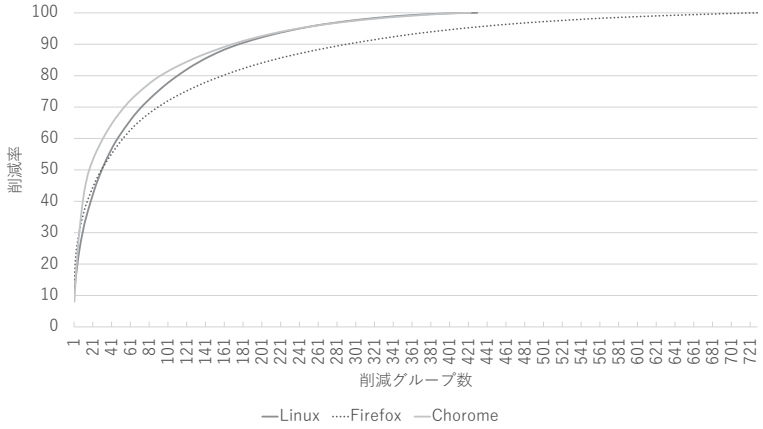


図 2: 削減の対象とする集合の数に対する削減率の推移

ドクローンの検出機能において，重複しているクローン片のうち最も長い箇所のみを採用することで重複するコードクローンを除外している．この際，除外されたクローンのうち，重複していない箇所についても同時に除外されており，コードクローンの量が少なく計測されている．より正確にコードクローンの量を計測するには，重複を含むクローン片を重複箇所とそうでない箇所で分割し，重複箇所のみを除外する必要がある．この処理による，削減可能量の計測値への影響の調査も今後の課題とする．

謝辞 本研究は JSPS 科研費 JP26730036，JP15H06344，JP16K16034 の助成を受けたものです．

参考文献

[1] Andreas Zeller. *Why Programs Fail*. Morgan Kaufmann Publishers, 2005.
[2] Brenda S. Baker. Finding clones with dup: Analysis of an experiment. *IEEE Trans. Softw. Eng.*, Vol. 33, No. 9, pp. 608–621, 2007.
[3] Toshihiro Kamiya, Shinj Kusumoto, and Katsuro Inoue. 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.
[4] Ira D. Baxter, Andrew Yahin, Leonardo de Moura, MarceloSant ' Anna, Lorraine Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM'98*, pp. 368–377, 1998.
[5] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. An ethnographic study of copy and paste programming practices in oopl. In *Proc. of ISESE'04*, pp. 83–92, 2004.
[6] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: finding copy-paste and related bugs in large-scale software code. *IEEE Trans. Softw. Eng.*, Vol. 32, No. 3, pp. 176–192, 2006.
[7] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stéphane Glondu. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In *Proc. of ICSE'07*, pp. 96–105, 2007.
[8] Norihiro Yoshida, Yoshiki Higo, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. On refactoring support based on code clone dependency relation. In *Proc. of METRICS'05*, pp. 10 pp.–16, 2005.
[9] Ekwa Duala-Ekoko and Martin P. Robillard. Tracking code clones in evolving software. In *Proc. of ICSE'07*, pp. 158–167, 2007.
[10] Michael Toomim, Andrew Begel, and Susan L. Graham. Managing duplicated code with linked editing. In *Proc. of VL/HCC'04*, pp. 173–180, 2004.
[11] 石津卓也, 吉田則裕, 崔恩瀞, 井上克郎. メタヒューリスティクスを用いた集約可能コードクローン量の推定. 情報処理学会研究報告, Vol. 2016-SE-193, No. 5, pp. 1–8, 2016.