社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

コードクローンの長さに基づくプログラム盗用確率の実験的算出

岡原 聖† 真鍋 雄貴‡ 山内 寛己† 門田 暁人† 松本 健一† 井上 克郎‡

†奈良先端科学技術大学院大学 情報科学研究科 〒630 - 0192 奈良県生駒市高山町 8916-5
‡大阪大学大学院 情報科学研究科 〒560-8531 大阪府豊中市待兼山町 1-3

E-mail: †{satoshi-o, hiroki-y, akito-m, matumoto}@is.naist.jp, ‡{y-manabe, inoue}@ist.osaka-u.ac.jp

あらまし 一般に，プログラム間で一致するコード列（コードクローン）が見つかった場合，コードの盗用もしくは流用の疑いがある．一方で，独立に開発されたプログラム間で偶然（もしくは定型処理など）によりコードクローンが生じることもある．本稿では，どの程度の長さのクローンであれば，偶然に生じたものではないと言えるか，その判断基準を実験的に導出する．実験では，独立に開発された（流用のない）多数のプログラム間で検出されるコードクローンの長さと個数を調査し，最大クローン長とクローン検出確率の関係を算出した．そして，偶然に生じうるコードクローンの検出確率を累乗近似により定式化した．導出した式により，2 つのプログラム間の最大クローン長を計測することにより，偶然や定型処理ではない，すなわち，盗用や流用が行われた確率を求めることが可能となった．

キーワード ソースコード盗用，ソースコード流用，ソフトウェアメトリクス，ソフトウェア計測

Experimentally Deriving Probability of Program Piracy based on Length of Code Clone

Satoshi OKAHARA†, Yuki MANABE‡, Hiroki YAMAUCHI†

Akito MONDEN†, Kenichi MATSUMOTO†, and Katsuro INOUE‡

† Graduate School of Information Science, Nara Institute of Science and Technology

8916-5 Takayama-cho Ikoma-shi, Nara, 630–0192 Japan

‡ Graduate School of Information Science and Technology Osaka University 1-3

Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

E-mail: †{satoshi-o, hiroki-y, akito-m, matumoto}@is.naist.jp, ‡{y-manabe, inoue}@ist.osaka-u.ac.jp

Abstract Generally, if a piece of code clone was found between two different programs, a plagiarism or a code reuse (program piracy) might be made. On the other hand, code clone also occurs accidentally or by code idioms. This paper experimentally derives the probability of program piracy based on length of code clone. In the experiment, we identified code clones among many different programs which we confirmed that program piracy has not been made, and formulated by power approximation the relation between the length of code clone and its derivation probability. By using this formula, we can compute the probability of program piracy from the maximum length of code clone derived from given two programs.

Keyword source code plagiarism, source code reuse, software metrics, software measurement

1. はじめに

近年，オープンソースソフトウェアの普及とともに，プログラム盗用が問題となっている[4][11-12]．プログラム盗用とは，プログラム製作者の意図に反したプログラムの流用のことである．（ただし，盗用と流用を技術的に区別することが困難なために，本稿における「盗用」は流用も含むこととする）．

プログラム盗用を防ぐために，プログラム盗用の検出技術が必要とされている．特に，ソフトウェアを開発する企業では，ソフトウェアの出荷前にプログラム盗用が無いことを確認したいというニーズがあり，プログラム盗用の検出技術が求められている[8]．

従来，プログラム盗用を検出する様々な手法として，ソフトウェアバースマークやコードクローンを用いた

プログラム盗用の検出手法などが提案されている．これらの既存研究は，部分的なプログラム盗用の検出が困難という問題や，提案した手法により検出された部分がプログラム盗用であるという判断基準については述べられていないという問題がある．例えば，２つのプログラムの類似度が算出できたとしても，盗用であると断定することは難しい．そこで，プログラム盗用の判断基準が必要とされる．

本稿では，部分的なプログラム盗用を検出するためにコードクローンに着目し，プログラム盗用と判断するための基準としてコードクローンの長さが利用できると考えた．

本稿では，プログラム盗用の判断基準を明確にすることを目的とし，ケーススタディとしてプログラム盗用以外の理由でソフトウェア間コードクローンの長さとコードクローンが検出される確率の関係を実験的に算出した．実験対象として，C または C++で記述された盗用のないことを確認したソフトウェアの集合を用いた．その結果から，最大のコードクローンの長さとコードクローンが検出される可能性の関係を累乗近似により定式化できた．また，この結果はソフトウェア数や，ソフトウェアのドメインに依存しないことを確認した．

以降，本稿の関連研究として，バースマークとコードクローンを説明する（2 章）．次に，本稿でのプログラム盗用の判断基準を求めるためのアプローチを説明する（3 章）．そして，コードクローンの長さに着目してプログラム盗用確率を実験的に求めるための実験とその結果についての考察を述べる（4 章）．最後にまとめと今後の課題について述べる（5 章）．

2. 関連研究

2.1. ソフトウェアバースマーク

バースマークは，対象となるソフトウェアが持つプログラムの特徴を抽象化し，表現したものである．2 つのプログラムからそれぞれバースマークを抽出し，それらが類似している場合に盗用が行われたと判断する．

バースマークは，特徴を取得できる状態によって 2 つに分類できる．1 つはプログラムを実行せずに取得可能な特徴を抽出するのを静的バースマーク，もう一つはプログラムの実行時から特徴を抽出するのを動的バースマークという．

- 静的バースマーク
  Tamada らは constant values in field variables ，sequence of method calls，inheritance structure，used classes を特徴抽出の対象として提案している[13].
  Myles らはオペコードのシーケンスをクラスファイルから取り出し，そこから k-gram を抽出し，得られた集合を k-gram birthmark としている[7].
- 動的バースマーク
  岡本らは Windows API の実行順序と実行頻度を動的バースマークとして提案している[9].
  Myles らは Java のバイナリ中の各基本ブロックの実行パターンを文脈自由文法で表現し，その表現をグラフ化したものをバースマークとしている[6].
  林らは実行系列中での処理間隔を利用した抽象化を用いてバースマークとしている[2].
  森山らは，岡本らと同様 Windows API の履歴などをバースマークとしているが，失敗呼び出しを無視することなどで攻撃耐性を向上させている[5].

その他，これらのバースマークの比較を行う類似度のアルゴリズムの提案など様々行われている．ただし，バースマークの性質上，プログラム全体が盗用された場合にはその検出に役立つが，プログラムの一部のみが盗用された場合には検出が困難である．

2.2. コードクローン

プログラムテキスト中の類似または一致するコードの断片をコードクローンと呼ぶ．コードクローンは，ソフトウェアの開発者や保守作業者により行われるコピー&ペーストや，定型処理，パフォーマンス改善のための意図的な繰り返し，コード生成ツール，偶然の一致により生じる[3]．部分的なプログラム盗用は主にコピー&ペーストにより行われると考えられるので，コードクローンの検出はプログラム盗用の発見にも役立つ．

コードクローンを用いたプログラム盗用検出手法として JPlag[10]がある．JPlag はプログラムを構文解析し，トークン列に変換し，トークン列間の類似度を求めることによりプログラム盗用の検出を行う．しかし，この手法は，プログラム間における全体的な類似度を算出する手法であり，また，類似度がプログラム全体の規模に依存するため，部分的なプログラムの類似度は算出できない．また，算出した類似度についてもプログラム盗用と判断する基準が存在しないため，この手法により算出した類似度を根拠にプログラム間でプログラム盗用が行われていると断定することは危険である．

3. コードクローン検出確率の算出

バースマークを用いてプログラム盗用を検出する手法には，部分的なプログラム盗用の検出が困難という問題があり，コードクローンを用いてプログラム盗

用を検出する手法には，プログラム盗用と判断する基準が無いという問題が存在する．プログラム盗用と判断をするためには，これらの問題を解決する必要がある．

本稿では，部分的なプログラム盗用を検出するためにコードクローンを用いる．これは，部分的なプログラム盗用を発見することが可能なためである．

本稿では，特に，ソフトウェア間コードクローンに着目する．コードクローンはソフトウェア内コードクローンとソフトウェア間コードクローンに分類することができる．ソフトウェア内コードクローンとは，コードクローンとして検出される 2 つ以上のコード列が同一のソフトウェアに存在するコードクローンである．ソフトウェア間コードクローンとは，コードクローンとして検出される 2 つ以上のコード列が異なるソフトウェアに存在するコードクローンである．

ソフトウェア内コードクローンは，主に同一のソフトウェア製作者によって生成されると考えられ，プログラム盗用により生じることはまずない．一方で，ソフトウェア間コードクローンは，一般的に異なるソフトウェア製作者によって生成されるため，プログラム盗用により生じる可能性がある．以上により，本稿ではソフトウェア間コードクローンのみ扱う．

プログラム盗用を判断する基準として，コードクローンの長さに着目した．異なる人が同様の処理を実装する場合，細部までソースコードが類似する確率は低いと考えられる．特に，コードクローンの長さが長いものが一致する可能性が低いと考えられるため，長いコードクローンが検出されるだけでプログラム盗用が疑われる．一方，プログラム盗用以外の理由によるコードクローンの検出が考えられるため，ソフトウェア間コードクローンは必ずしもプログラム盗用とは限らない．ただし，そのようなクローンは長さが短いと考えられる．

そこで，本稿ではソフトウェア間コードクローンの長さに着目し，プログラム盗用確率を算出することを目指す．本稿では，プログラム盗用の判断基準を明確にするためのケーススタディとして，プログラム盗用のないソフトウェア間で，ある長さのソフトウェア間コードクローンが偶然に検出される確率を実験的に求める．本稿では，この確率のことをコードクローン検出確率と呼ぶこととする．この確率が小さいと盗用の疑いが大きいことになる．

4. 実験

本実験では，プログラム盗用以外の理由により生じるソフトウェア間コードクローンのコードクローン検出確率を算出することを目的とし，ソフトウェア間コードクローンの長さとコードクローン検出確率の関係を調査する．以降では実験の詳細について述べる．

4.1. 実験環境

本稿では，広く利用されている CCFinderX[1]を用いてコードクローンの検出を行った．コードクローンの検出方式には，行単位の検出，トークン単位の検出などが存在する[3]．CCFinderX は，コードクローンをトークン単位で検出できる．

4.1.1. コードクローン検出手法

ここでは，CCFinderX が用いるコードクローン検出手法の概要を示す．CCFinderX はソースコード集合を入力とし,コードクローンについての情報を出力する．CCFinderX が用いるコードクローン検出手法はトークン解析，トークン変換，マッチングからなる．

A) トークン解析

プログラミング言語の字句規則に従い，入力として与えられたソフトウェアに含まれるすべてのソースコードをトークンに分割する．ソースコード中の空白，コメントは無視する．

B) トークン変換

型，変数，定数の各トークンを同一トークンに置き換える．同一トークンに置き換えることで，変数名が異なるコード列の組をコードクローンとして検出することができる．

C) マッチング

トークン変換後のトークン列から同一部分列の組を探してコードクローンとして検出する．

本稿では，コードクローンを構成するトークンの数をコードクローン長と呼ぶこととする．

4.2. 実験対象

本実験は，C または C++で書かれたオープンソースソフトウェア 100 件のうち，プログラム盗用（流用）がないと確認できた 49 件を実験対象とした.プログラム盗用の有無の確認は，以下の手順にて行った．

(ア) CCFinderX を用いてソフトウェア間コードクローンの検出を行う．

(イ) コードクローン長の大きなコードクローンを目視で確認し，プログラム盗用のあったソフトウェアを実験対象から除外する．（ただし，コードクローン長 80 未満のものについては数が多すぎたため確認していない）

実験対象とした 49 件のソフトウェアは，楽譜作成ツールやダウンロード支援ツールなど様々なドメイン

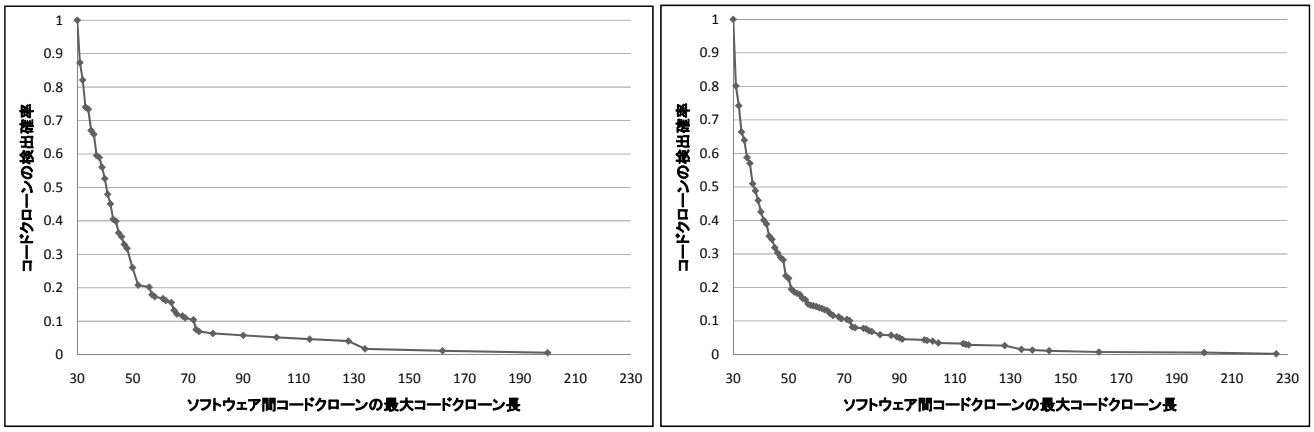

図 1. ソフトウェア数 25 件(左図)とソフトウェア数 49 件(右図)の実験結果

からなり，ソフトウェアの規模もそれぞれ異なるものとなっている．

また，本実験では，異なるソフトウェア数でコードクローン検出確率の傾向が変化するかを調べるために，49 件のうち，ランダムに 25 件選択したソフトウェア集合も作成した．25 件の結果と 49 件の結果を比較することによって，実験結果がソフトウェア数に依存するか否かを確認できる．

4.3. 実験手順

本実験は以下の手順にて行った．

1. ソフトウェア集合に含まれる 2 つのソフトウェアからなる組に対し，ソフトウェア間コードクローンを検出する．本実験では，コードクローンとして検出される最小のコードクローン長を 30 に設定している．
2. 各ソフトウェアの組で検出されたソフトウェア間コードクローンから，各組の最大コードクローン長を算出する．
3. 算出した最大コードクローン長ごとに，ソフトウェアの組数を数え，コードクローン検出確率を求める．最小のコードクローン長を min，ソフトウェア間の最大コードクローン長を l，コードクローンが検出されたソフトウェアの組合せ数を m_l とするとき，長さ l のコードクローン検出確率 P_l は以下の式(1)で算出する．

$$\begin{cases} P_l = 1 - \dfrac{m_l}{{}_n\mathrm{C}_2} & (l = \min) \\ P_l = P_{l-1} - \dfrac{m_l}{{}_n\mathrm{C}_2} & (l \neq \min) \end{cases} \qquad (1)$$

4.4. 実験結果と考察

ソフトウェア間コードクローンの最大コードクローン長とコードクローン検出確率の関係を図 1 に示す．

図 1 の各結果から累乗近似曲線を算出した．算出した累乗近似曲線はそれぞれ以下の式で表わされる．

$$P_l = 7188.8 \cdot l^{-2.606} \qquad (2)$$
$$P_l = 9830.7 \cdot l^{-2.718} \qquad (3)$$

(2)式がソフトウェア数 25 件の累乗近似曲線，(3)式がソフトウェア数 49 件の累乗近似曲線である．近似曲線の精度を検討するために(2), (3)式を用いてソフトウェア間コードクローンの検出確率の近似値を算出し，残差の平方平均を求めた．その結果，(2)式の残差の平方平均は 0.00054，(3)式の残差の平方平均は 0.00036 となった．これらは十分に小さい値のため，累乗近似曲線の精度は高いといえる．

(2), (3)式から，ソフトウェア間コードクローンの最大コードクローン長が増加するにつれて，コードクローン検出確率が減少することが確認できた．また，(2), (3)式は $l \geqq 30$ ではほぼ同じ値を取ることを確認した．これらのことから，実験結果より得られた傾向はソフトウェア数に依存しないと考えられる．また，25 件，49 件の中にはそれぞれ多様なドメインのソフトウェアを含むため，この結果はドメインに関わらず有用と考えられる．

5. おわりに

本稿は，プログラム盗用の判断基準を明確にすることを目的とし，盗用のない多数のソフトウェアに対して，ソフトウェア間コードクローンの最大コードクローン長とコードクローン検出確率の関係を分析した．実験により，ソフトウェア間コードクローンの最大コードクローン長とコードクローン検出確率の関係を累

乗近似により定式化し，減少傾向であることを明確にした．また，実験結果がソフトウェア数に影響されないことを確かめた．

本稿で行った調査は，プログラム盗用のないソフトウェアのみを用いているため，プログラム盗用を現実に検出できたことを示すものではない．今後の予定としては，実際に盗用のあるソフトウェアを用いて調査を行っていく予定である．

謝辞

本稿の一部は，文部科学省 「次世代ＩＴ基盤構築のための研究開発」の委託に基づいて行われた．また，本稿の一部は，文部科学省科学研究費補助金（基盤Ｃ：課題番号 19500056）による助成を受けた．

文　献

[1] CCFinderX: http://www.ccfinder.net/ccfinderx-j.html

[2] 林晃一郎, 楓基靖, 真野芳久, “特徴抽出と抽象化による動的バースマークの構成とその検証,” 情処学論, vol.48, no.4 pp.1799-1808, Apr. 2007.

[3] 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,” 信学論(D), vol.91-D, no.6, pp.1465-1481, Jun. 2008.

[4] インプレスジャパン, “Sigma Designs、RMP4 に XVID からのコード流用を認める,” (online), available from <http://av.watch.impress.co.jp/docs/20020828/sigma.htm>, (accessed 2008-11-24).

[5] 森山修, 古江岳大, 遠山毅, 松本勉, “API 関数呼び出し履歴を用いた動的ソフトウェアバースマークに対する攻撃への対策,” 情処学論, vol.48, no.9, Sep. 2007.

[6] G. Myles, and C. Collberg, “Detecting Software Theft via Whole Program Path Birthmarks,'' In Proc. Information Security Conference, Sep. 2004.

[7] G. Myles and C. Collberg, “k-gram Based Software Birthmarks,” Proc. of the 2005 ACM symposium on Applied computing, pp.314-318, Santa Fe, U.S.A, Mar. 2005.

[8] オージス総研, “Palamida,” (online), available from <http://www.ogis-ri.co.jp/pickup/palamida/>, (accessed 2008-11-24).

[9] 岡本圭司, 玉田春昭, 中村匡秀, 門田暁人, 松本健一, “API 呼び出しを用いた動的バースマーク,” 信学論(D), vol.J89-D, no.8, pp.1751-1763, Aug. 2006.

[10] L. Prechelt, G. Malpohl, and M.Philippsen, “Finding Plagiarisms among a Set of Programs with JPlag,” J. Universal Computer Science, vol.8, no.11, pp.1016-1038, Nov. 2002.

[11] Slashdot, "Epson Pulls Linux Software Following GPL Violations," (online), available from <http://slashdot.org/article.pl?sid=02/09/11/2225212>, (accessed 2008-11-24).

[12] スラッシュドットジャパン, "プロジーの DivX コンバータに GPL 違反の疑い," (online), available from <http://slashdot.jp/articles/03/02/05/1738259.shtml>, (accessed 2008-11-24).

[13] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto, “Java Birthmarks -- Detecting the Software Theft,” IEICE Trans. Inf. & Syst., vol.E88-D, no.9, pp.2148-2158, Sep. 2005.