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

ソースコード流用のコードクローンメトリクスに基づく検出手法

岡原 聖† 真鍋 雄貴‡ 山内 寛己† 門田 暁人† 松本 健一†

†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@ist.osaka-u.ac.jp

あらまし 近年，OSS のソースコードの一部を再利用した開発がよく行われている．ただし，OSS のライセンスに沿わない再利用（ソースコード流用）が指摘されるケースが報告されており，ソフトウェア開発企業と開発委託元にとって新たなリスクとなっている．本研究では，ソースコード流用のコードクローンメトリクスに基づく検出手法を提案する．本稿では，ソフトウェア間の最大コードクローン長と部分類似度に着目し，どの程度の値であればソースコード流用ありといえるか，その判断基準を実験的に導出する．50 件の OSS を用いて実験した結果，ソフトウェア間で検出される最大コードクローン長，部分類似度を用いたソースコード流用の判断基準を導出した．導出した判断基準を用いることで，作成した正解集合の約 84%を false-positive なしでソースコード流用ありと判断できた．

キーワード ソフトウェアメトリクス，ソフトウェアの部分類似度，コードクローンの長さ

Detection of Source Code Reuse based on Code Clone Metrics

Satoshi OKAHARA†, Yuki MANABE‡, Hiroki YAMAUCHI†

Akito MONDEN†, and Ken-ichi MATSUMOTO†

† 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@ist.osaka-u.ac.jp

Abstract In recent software development, programmers often reuse source code of Open Source Software(OSS) as a part of commercial product. While reusing OSS increase productivity, it raises a new risks of violating OSS license to both software venders and purchasers. This paper proposes a method to detect reuse of source code based on code clone metrics. For two code clone metrics, maximum length of code clone and partial similarity of software, we experimentally derived criteria to identify the reuse, by using 50 OSS packages. By using derived criteria, we could correctly identify 84% of products containing reused code without having false-positive.

Keyword Software Metrics, Partial Similarity of Software, Length of Code Clone

1. はじめに

近年，ソフトウェア開発において，オープンソースソフトウェア（OSS）のソースコードを流用して，開発を行うことがある．しかし，オフショア開発など，開発の外注により，開発委託元が意図しないソースコード流用が発生し，ライセンス違反を犯してしまうケースが報告されている[1][9][10]．そのため，開発委託元には，ソフトウェアの出荷前にソースコード流用が存在しないことを確認したい，もしくは，ソースコード流用が存在する場合には，どの部分が流用なのかを確認したいというニーズがあり，ソースコード流用検出の技術が開発されている[5][7]．

ソースコード流用検出の従来研究として，ファイル間に含まれる一致または類似したコード片がファイルに占める割合（コードクローン含有率）を用いた既存手法がある[8]．しかし，従来手法では，ソースファイルの一部に流用があるような部分的なソースコード流用の場合，コードクローン含有率が小さくなるため検出が難しい．

そこで，本稿では，コードクローンの量に基づくソースコード流用検出手法を提案する．コードクローンの量を測る尺度として,「ソフトウェア間の最大コードクローン長」と，最大コードクローンを含むファイルのコードクローン含有率を表す「ソフトウェア間の部

分類似度」を用いる．ソフトウェア間の最大コードクローン長は，事前調査から，部分的なソースコード流用検出に役立つことが分かっている[6]．一方，ソフトウェア間の部分類似度は，本稿で新たに提案する尺度であり，ソースコード間の最大コードクローン長が小さい場合であっても，部分的なソースコード流用検出が期待できる．

本稿では，上記の 2 つの尺度についてソースコード流用があるかどうかを判断する為の判断基準（閾値）を，50 件の OSS を用いて実験的に求める．

以降，2 章でソースコード流用検出に求められる要件について整理する．3 章で提案手法の詳細について述べる．そして，4 章にて提案手法の有効性を検討するための実験と，その結果と考察を述べたあと，5 章でまとめと今後の課題について述べる．

2. ソースコード流用

本章では，ソースコード流用の定義を行い，また，ソースコード流用検出に必要な要件を整理する．

2.1. ソースコード流用の定義

2 つのソースコード片 *p*,*q* について，以下の A)，B)または C)を満たす場合，*p*,*q* はソースコード流用の関係にあると定義し，本稿における検出対象とする．

A) *q* は *p* の完全なコピーである．
B) *q* は *p* のソースコード中に現れるすべて（または一部）のシンボル名を変更したものである．
C) *q* は *p* のソースコードからコメント行をすべて（または一部）を追加・削除したものである．

2.2. ソースコード流用の検出

[要件 1] 改変のあるソースコード流用の検出

ソフトウェア開発において，2.1 節 A)のような完全なコピーが行われるとは限らない．B)，C)のようにシンボル名の変更やコメントの追加・削除が行われる場合がある．そのため，ソースコード流用検出を行うには，完全なコピーの検出だけではなく，シンボル名の変更やコメントの追加・削除が行われたコピーにも対応する必要がある．

ただし，本稿では，ソースコード流用が検出されないように難読化などの細工を行った場合については取り扱わない．このような細工の存在するソースコード流用検出は非常に難しいため，今後の課題とする．

[要件 2] 部分的なソースコード流用の検出

ソースコード流用には，ライブラリや一部のアルゴリズムの使用など，ソースコード全体を流用しないソースコード流用（部分的なソースコード流用）が存在する．また，流用したソースコードに機能を追加，変更を加える場合もある．そのため，流用検出を行うには，部分的な流用に対応する必要がある．

2.3. ソースコード流用の判断基準

ソースコード流用検出には，流用の有無を判断する為の判断基準（閾値）が必要である．ただし，明確にソースコード流用あり（ソースコード流用なし）と分かる事例ならばよいが，判断の難しい事例が考えられる.そのため,判断の確からしさを表す為の適合率と，再現率を求める必要がある．

3. 提案手法

本章では，コードクロ―ンに基づく部分的なソースコード流用検出手法を提案する．まず，コードクローン検出手法の説明を行い，着目したコードクローンメトリクスについて説明する．次に，本稿で提案するソースコード流用検出手法について説明する．

3.1. コードクローン

コードクローンとは，ソースコード中の類似または一致したコード列のことである．コードクローンは，コピーアンドペーストや定型処理，偶然の一致などの理由で発生する[3].

ソースコード流用は，コードクローンと同様にコピーアンドペーストなどの理由で発生するため，ライセンスの異なるソフトウェア間でコードクローンが検出されるとソースコード流用の疑いがある．ただし，コードクローンの発生理由に偶然の一致や定型処理が存在するため，コードクローンは必ずしもソースコード流用によるものとは限らない．そのため，2.3 節で述べたとおり，検出されたコードクローンがソースコード流用によるものかどうかの判断基準が必要となる．

3.2. コードクローン検出方法

コードクローンは，ソースコードまたはソースコード群を入力として，与えられた閾値以上連続して一致する行や字句の部分列がコードクローンとして検出される．本稿では，字句単位でのコードクローン検出方法を用いる[4]．字句単位のコードクローン検出方法はトークン解析，トークン変換，マッチングの 3 段階を経てコードクローンの検出を行う．以下，各段階について説明する．

(1) 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ースコードに含まれる全てのソースコードをトークンに分割する．このとき，ソースコード中の空白やコメントは無視する．

(2) 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出来る．

(3) マッチング

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

コードクローンは，シンボルを同一トークンに置き換えて検出するため， 2.1 節で述べた要件 1 を満たすことが出来る．

3.3. 流用検出のためのコードクローンメトリクス

本稿では，ソフトウェア間の最大コードクローン長と，ソフトウェア間の部分類似度を用いる．以降，それぞれの定義とソースコード流用検出における狙いを述べる．

3.3.1. ソフトウェア間の最大コードクローン長

2 つのソフトウェア間のコードクローンの中で最大のトークン数を持つものを最大コードクローン,また，そのトークン数を最大コードクローン長と呼ぶ．

ソフトウェア間の最大コードクローン長とソースコード流用との間には，次の二つの関係がある[5].

(1) 最大コードクローン長が小さいとき，
ソースコード流用の可能性は低い

(2) 最大コードクローン長が大きいとき，
ソースコード流用の可能性は高い

全く独立に開発したソフトウェアであっても，ソースコードの一部が偶然に一致する場合がある．そのような一致は短いコードクローンとして検出されると考えられる（上記の(1)）．一方，長いコードクローンは偶然には発生しにくいと考えられる（上記の(2)）．

3.3.2. ソフトウェア間の部分類似度

最大コードクローン長を含むファイルのコードクローン含有率をソフトウェア間の部分類似度と呼び，次式で与えられる．

$$PSim(A,B)=\frac{2\cdot MLCC(a,b)}{|a|+|b|}$$

ソフトウェア A,B 間の部分類似度 $PSim(A,B)$は，a,b をソフトウェア A,B 間で最大コードクローンを含むファイル，$|a|,|b|$:ファイル a,b の長さ，a,b 間の最大コードクローン長を $MLCC(a,b)$とする．

ソフトウェア間の部分類似度は，2 つのソフトウェア間で類似または一致しているソースファイルの組がある場合に高くなる．そのため，部分的なソースコード流用の検出に役立つと期待される．

3.4. ソースコード流用の有無の判断方法

本節では，3.3 節の各メトリクスを用いたソースコード流用の判断方法について説明する．

本稿では，図 1，2 に示すソースコード流用の判別フローチャートを用いてソースコード流用の有無を判別する．まず，ソースコード流用ありと判断する閾値 T_R と，ソースコード流用なしと判断する閾値 T_{NR} を決定する．なお，閾値 T_R と T_{NR} は $MLCC$ と $PSim$ に対してそれぞれ決定する（最大コードクローン長の閾値：$T_{R\text{-}MLCC}$, $T_{NR\text{-}MLCC}$, 部分類似度の閾値: $T_{R\text{-}PSim}$, $T_{NR\text{-}PSim}$）．ソフトウェア間の最大コードクローン長の閾値 $T_{R\text{-}MLCC}$，または，ソフトウェア間の部分類似度の閾値 $T_{R\text{-}PSim}$ を上回るとソースコード流用ありと判断する．一方，ソフトウェア間の最大コードクローン長の閾値 $T_{NR\text{-}MLCC}$，または，ソフトウェア間の部分類似度の閾値 $T_{NR\text{-}PSim}$ を下回るとき，ソースコード流用なしと判断する．

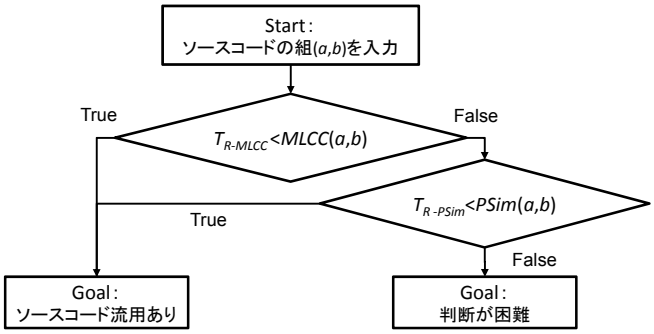

図 1．閾値 T_R を用いた流用ありフローチャート

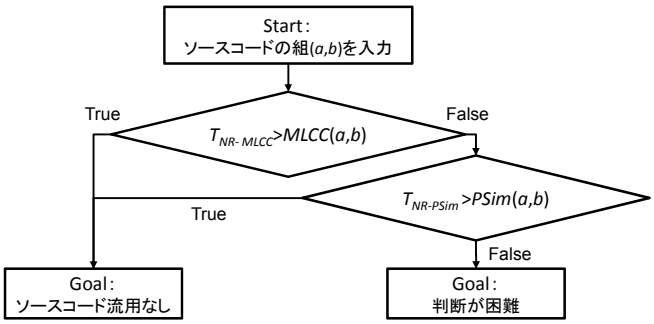

図 2．閾値 T_{NR} を用いた流用無しフローチャート

3.5. 閾値の決定方法

各閾値の評価指標に適合率，再現率を用いる．適合率，再現率は以下の通りである．

$$(\text{適合率})=\frac{\text{正検出数}}{\text{全検出数}},\quad (\text{再現率})=\frac{\text{正検出数}}{\text{正解集合の要素数}}$$

流用の有無を検出するにあたって，false-positive（流用がないにもかかわらず「流用あり」と判断する，もしくは，流用があるにもかかわらず「流用なし」と判断すること）を避ける為に，本稿では，特に，適合率に着目する．適合率とは「検出結果がどの程度正しいか」を示す評価指標であり，2.3 節の要件を満たす．本稿では，ソースコード流用を含む場合を正解としたときの適合率（ソースコード流用あり適合率）と，ソースコード流用を含んでいない場合を正解としたときの適合率（ソースコード流用なし適合率）の両方を求める．

適合率が 1 のときは，検出結果全てがソースコード

流用あり（ソースコード流用なし）であるため，適合率 1 のときの各メトリクスの値を閾値の候補とする．さらに，多数のソースコード流用を検出するために，再現率にも着目する．再現率は，検出結果中にどの程度正解集合が含まれているかを示すため，適合率 1，かつ，再現率が最大値をとる各メトリクスの値を閾値とする．

4. 実験

提案手法の有効性を確認するため，提案手法で用いる閾値を導出し，導出した閾値を用いてソースコード流用を判断できるかどうかを確認した．

コードクローン検出ツールには神谷年洋博士の開発した CCFinderX[2]を用いた．実験対象には，C 言語または C++で開発された 50 件の OSS を使用した．OSS は Games，Security，Audio など多数のドメインからなっており，ソフトウェアの規模も多様である．

4.1. 実験内容

本実験は以下の手順にて行った．

手順1. ソフトウェアの全組合せについて，ソースコード流用の有無を予め判断し，ソースコード流用ありの正解集合を作った．正解集合を作成する際，ソースコード流用の有無は，「ライセンス明記の有無」，「プログラム構造」，「検出したコードクローンの内容」から判断した．作成した正解集合を表 1 に示す．

表 1．作成した正解集合

コードクローンの分類	組合せ件数
ソースコード流用ありコードクローン	121
ソースコード流用なしコードクローン	648
実験対象外のコードクローン	456

手順2. 2.2 節で述べた，シンボル名の変化やコメント行の追加・削除の存在する，コードクローン長が 30 以上のコードクローンを検出した．コードクローン長が 30 未満のコードクローンは数行程度で，ソースコード流用の可能性が低いためである（図 3）．

```
if (string_space != NULL)        if(gitab!=NULL)
   free (string_space);              free(gitab);
if (map != NULL)                 if(gst1tab!=NULL)
   free (map);                       free(gst1tab);
```

図 3．コードクローン長が 30 の例

手順3. ソフトウェア間の最大コードクローン長，ソフトウェア間の部分類似度を算出する．

手順4. 表 1 の正解集合に基づいて，ソフトウェア間の最大コードクローン長，ソフトウェア間の部分類似度の閾値を変化させたときの，ソースコード流用ありの適合率，再現率を算出する．また同様に，ソースコード流用なしの適合率，再現率を算出する．

5. 結果と考察

5.1. 結果

ソフトウェア間の最大コードクローン長を閾値として変化させたときの適合率と再現率を図 4，ソフトウェア間の部分類似度を閾値として変化させたときの適合率と再現率を図 5 に示す．また，閾値，検出したソースコード流用あり（ソースコード流用なし）の件数，検出出来なかったソースコード流用あり（ソースコード流用なし）の件数の内訳を表 2 に示す．

図 4 から，ソフトウェア間の最大コードクローン長を用いた場合，閾値 $T_{R\text{-}MLCC}$ は 270，閾値 $T_{NR\text{-}MLCC}$ は 50 であった．図 5 から，ソフトウェア間の部分類似度を用いた場合，閾値 $T_{R\text{-}PSim}$ は 0.30 と分かった．閾値

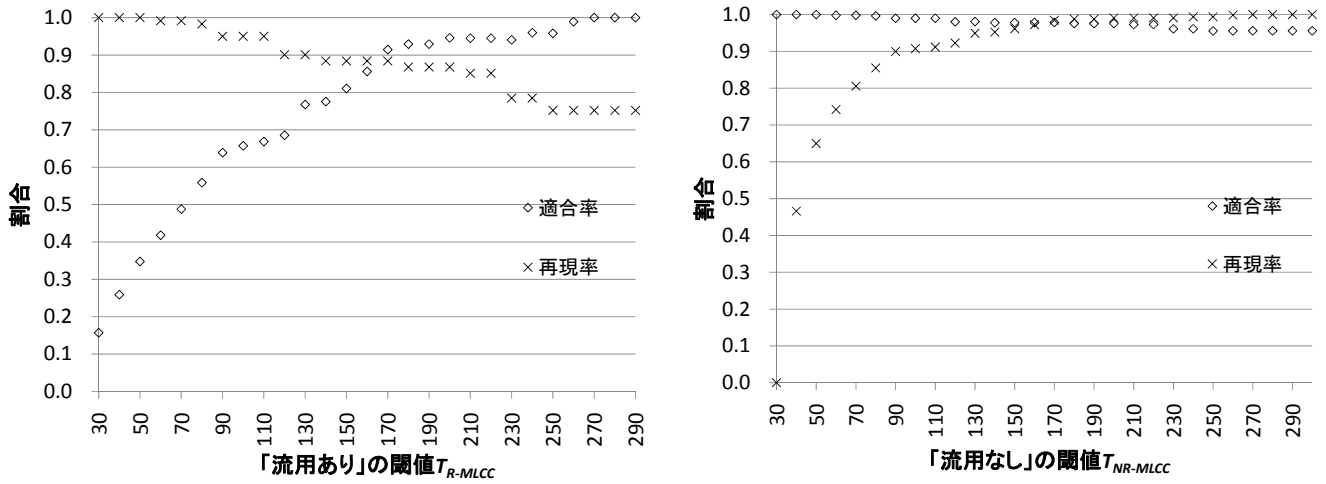

図 4．ソフトウェア間の最大コードクローン長を用いた実験結果

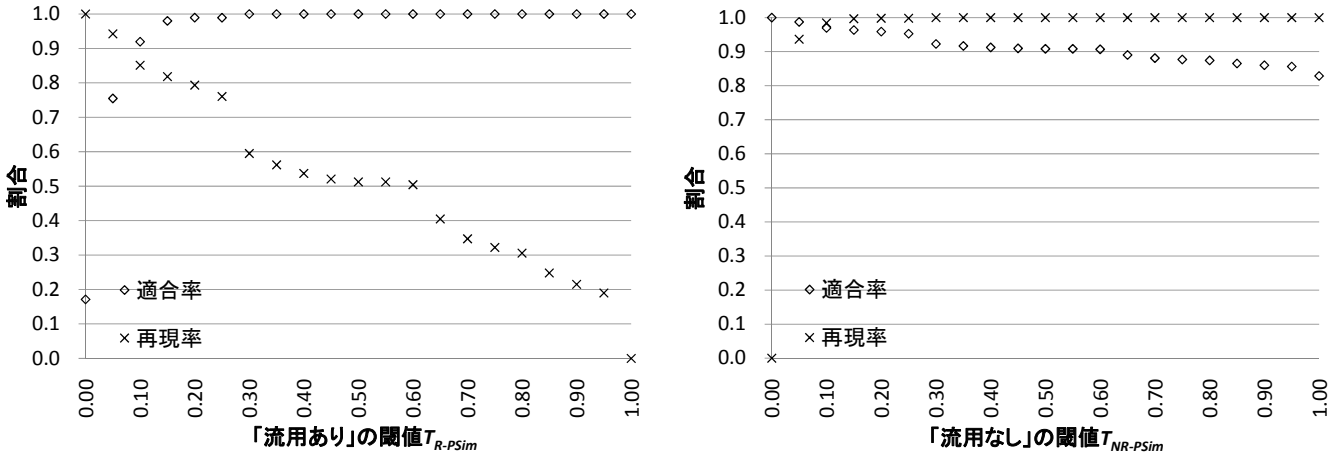

図 5. ソフトウェア間の部分類似度を用いた実験結果

表 2. 実験結果の内訳

	ソフトウェア間の最大クローン長		ソフトウェア間の部分類似度	
	ソースコード流用あり	ソースコード流用なし	ソースコード流用あり	ソースコード流用なし
閾値	270	50	0.30	-
検出した ソースコード流用あり(なし)の件数	91(30)※	421(227)※	72(49)※	-

※（）の値は検出できなかったソースコード流用あり（なし）の件数

```
/* Splice in "list" into "head" */
static inline void glame_list_splice(struct glame_list_head *list,
struct glame_list_head *head)
{
  struct glame_list_head *first = list->next;

  if (first != list) {
    struct glame_list_head *last = list->prev;
    struct glame_list_head *at = head->next;

    first->prev = head;
    head->next = first;

    last->next = at;
    at->prev = last;
  }
}
```

```
/*
 * Splice in "list" into "head"
 */
static INLINE void list_splice(struct list_head *list, struct list_head
*head)
{
  struct list_head *first = list->next;

  if (first != list) {
    struct list_head *last = list->prev;
    struct list_head *at = head->next;

    first->prev = head;
    head->next = first;

    last->next = at;
    at->prev = last;
  }
}
```

図 6. ソースコード流用ありの例（最大コードクローン長：59，部分類似度：0.32）

$T_{NR\text{-}PSim}$ は，ソフトウェア間の部分類似度が非常に小さい値のときにソースコード流用が検出されたため決定できなかった．

図 6 に示すコードクローンは，ソフトウェア間の最大コードクローン長の閾値 270 よりも小さいときにソースコード流用が存在したコードクローンである．ただし，ソフトウェア間の部分類似度が 0.32 を示し，ソフトウェア間の部分類似度の閾値 0.30 を超えた為，ソースコード流用として検出することができた．

5.2. 考察

図 4 の結果より，短いコードクローンは偶然の一致などで検出され，長いコードクローンはソースコード流用を含むことが多いと分かった．ただし，図 6 のように，ソースコード流用を含む短いコードクローンも検出されるため，最大コードクローン長だけでソースコード流用を判断することは困難だと分かった．

一方，図 5 の結果より，部分類似度は流用ありの検出には役立つが，流用なしの判断に用いることは出来ないと分かった．また，最大コードクローン長を用いた場合より検出精度が低いことが分かった．そこで，最大コードクローン長により流用の有無の判断が出来なかった 30 件について，部分類似度による流用あり判別を試みた．その結果，30 件のうち 11 件(約 36%）を流用ありとして新たに検出できた．全体として，流用を含む 121 件のプログラムの組に対し，102 件(約 84%)を false-positive なしで流用ありと検出出来たことになる．

このことから，最大コードクローン長と部分類似度を併用して流用検出を行うことが有用であると言える．

6. おわりに

本稿は，部分的なソースコード流用を検出することを目的として，コードクローンメトリクスに基づくソースコード流用検出手法の提案を行い，流用の有無を判定するための閾値を実験的に導出した．また，導出した閾値を用いて，ソースコード流用の判断の精度を確認した．実験結果から，50 件の OSS を用いて作成した正解集合のうち，流用を含む 121 件のプログラムの組に対し，102 件(約 84%)を false-positive なしで流用ありと検出できることが分かった．

今後は，他のメトリクスを併用することで検出精度の向上を目指す予定である．

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] @IT, “第 1 回　訴訟が増えている!? OSS ライセンス違反,” (online), available from <http://www.atmarkit.co.jp/flinux/rensai/osslc01/osslc01a.html>, (accessed 2009-10-12).

[2] CCFinderX, “CCFinder ホームページ,” (online), available from <http://www.ccfinder.net/ccfinderx-j.html>, (accessed 2009-10-12).

[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] T. Kamiya, S. Kusumoto, and K. 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, Jul. 2002.

[5] NEC, “ Protex,” (online), available from <http://www.nec.co.jp/oss/protexip/index.html>, (accessed 2009-10-12).

[6] 岡原 聖, 真鍋 雄貴, 山内 寛己, 門田 暁人, 松本 健一, 井上 克郎, “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,” 信学技報, No.SS2008-40, pp.7-11, Oct. 2008.

[7] オージス総研, “Palamida,” (online), available from <http://www.ogis-ri.co.jp/pickup/palamida/index.html>, (accessed 2009-10-12).

[8] 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.

[9] Slashdot, “PlayStation 2 Game ICO Violates the GPL,” (online), available from <http://news.slashdot.org/article.pl?sid=07/11/28/0328215>, (accessed 2009-10-12).

[10] Slashdot, “Epson Pulls Linux Software Following GPL Violations,” (online), available from <http://slashdot.org/article.pl?sid=02/09/11/2225212>, (accessed 2009-10-12).