情報処理学会論文誌 Vol.54 No.2 807–819 (Feb. 2013)

機械学習を用いた類似度に基づく有用なコードクローンの自動特定手法

楊 嘉晨[1,a)] 堀田 圭佑[1,b)] 肥後 芳樹[1,c)] 井垣 宏[1,d)] 楠本 真二[1,e)]

受付日 2012年5月14日, 採録日 2012年11月2日

概要：これまでにソースコード中からコードクローン（互いに類似するコード片）を自動的に検出するツールが多数開発されている．これらの検出ツールはコードクローンを多数検出するが，それらすべてがソフトウェア保守に有用であるとは限らない．さらに，あるコードクローンが有用であるか否かの判別基準は，個々の開発者により，あるいはコードクローン情報の使用目的などにより異なる可能性が高い．そこで本稿では，機械学習を用いた有用なコードクローンの自動特定手法を提案する．提案手法は，ツールによって検出されたコードクローンの一部を利用者に有用であるか否かに分類してもらい，それらを学習データとして残りのコードクローンからその利用者にとって有用なコードクローンを自動的に特定する．また，提案手法を実装し，被験者を用いて評価実験を行った．その結果，有用であるか否かの判断が利用者によって異なることを確認するとともに，提案手法が全体の 20%程度のデータを学習させることで，およそ 70%以上，最良の場合約 90.1%の精度で予測ができることを確認した．

キーワード：コードクローン，フィルタリング，機械学習，ソフトウェア保守

A Method for Identifying Useful Code Clones with Machine Learning Techniques based on Similarity between Clones

Jiachen Yang[1,a)] Keisuke Hotta[1,b)] Yoshiki Higo[1,c)]
Hiroshi Igaki[1,d)] Shinji Kusumoto[1,e)]

Received: May 14, 2012, Accepted: November 2, 2012

Abstract: Currently, there are a variety of code clone detection tools. However, results from clone detectors contain plentiful useless code clones. Thus, users of clone detectors need to identify useful code clones from a flood of code clones, which requires much efforts. In addition, judging whether a code clone is useful or not varies from user to user and from purpose to purpose. Therefore, it is difficult to define what code clones are useful for software maintenance. This paper proposes a filtering method for code clones based on machine learning techniques. The proposed method has its users mark on some of code clones whether they are useful or unuseful. The proposed method learns the judgement of the individual user, and predicts unmarked code clones whether they are useful or not. We have implemented the proposed method as a web-based system, and conducted a case study with subjects. As a result, we confirmed that the judgement of whether a code clone is userful or not varies from user to user. Moreover, we confirmed that the proposed method can predict the judgement of code clones by users with over 70% accuracy, in some cases it can predict the judgement with over 90% accuracy.

Keywords: code clone, filtering, machine learning, software maintenance

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University, Suita, Osaka 565–0871, Japan
[a)] jc-yang@ist.osaka-u.ac.jp
[b)] k-hotta@ist.osaka-u.ac.jp
[c)] higo@ist.osaka-u.ac.jp
[d)] igaki@ist.osaka-u.ac.jp
[e)] kusumoto@ist.osaka-u.ac.jp

© 2013 Information Processing Society of Japan

1. まえがき

近年，ソフトウェア工学における研究対象としてコードクローンが注目されている．コードクローンとは，ソースコード中の同一，あるいは類似するコード片のことを指す．一般的にコードクローンの存在はソフトウェアの保守に悪影響を与えると考えられており，これまでにコードクローンの検出技術，およびその除去技術がさかんに研究されている．また，コードクローンをソースコード中から自動的に検出するツールも多数開発，公開されている [1], [2], [3], [4].

コードクローン検出ツールは主にソフトウェア開発における保守フェーズにおいて利用される．開発者はコードクローンに起因する不具合の特定やライセンス違反の検知など，様々な用途においてコードクローン検出ツールを利用する [2], [5].

コードクローン検出ツールは対象となるソフトウェア中から特定したコードクローンをすべて提示するため，提示されるコードクローンの量は膨大になる．しかし，これらのコードクローンすべてがソフトウェア保守にとって有用であるとは限らない．このため，開発者は検出ツールによって提示された膨大なコードクローンの中から検出目的に即したコードクローンを特定する必要があるが，この作業は非常に大きな作業量を要する．さらに，あるコードクローンが有用であるか否かを判別する基準は，個々の開発者によって，あるいはコードクローンの検出目的によって異なると考えられる．したがって，どのようなコードクローンがソフトウェア保守に有用かを一般化することは難しい．

これまでにメトリクスなどを用いてコードクローンを分類する，あるいは有用でないものを除外する手法が提案されている [6], [7]．しかし，利用者ごと，あるいは利用目的ごとの違いを考慮したコードクローンのフィルタリング手法はこれまでに提案されていない．

本稿では，開発者がソフトウェア保守に有用なコードクローンを特定する作業を支援するために，機械学習を用いたコードクローンの自動フィルタリング手法を提案する．提案する手法では，まず利用者に一部のコードクローンを “有用” か “有用でない” に分類してもらう．その後，それらを学習データとして，残りのコードクローンを “有用” または “有用でない” に自動的に分類し，利用者に提示する．提案手法を用いることで，利用者は一部のコードクローンに対して自身が有用と思うか否かを入力するのみで，膨大な検出結果から自身にとって有用なコードクローンのみを抽出することが可能となる．これにより，有用なコードクローンの特定に要するコストを低減させることができる．さらに，利用者ごとに学習データを作成するため，利用者や利用状況による有用か否かの判別基準の違いにも対応できる．

2. 提案手法

本章では，提案するコードクローンのフィルタリング手法について述べる．

2.1 入出力

提案手法が入力として必要とするものは以下のとおり．

(1) 対象ソフトウェアのソースコード
(2) 対象ソフトウェア中に存在する全クローンセットの位置情報
(3) 学習データ

ここでクローンセットとは，互いにコードクローン関係にあるコード片の集合を指す．また (3) の学習データとは，対象ソフトウェアから検出されたクローンセットの一部であり，利用者によって “有用” あるいは “有用でない” のいずれかの分類が付与されている．

これらの入力をもとに，提案手法は与えられた利用者によって分類されていないクローンセット，すなわち学習データに含まれないすべてのクローンセットについて，それらが “有用” か “有用でない” かを自動的に判別し，提示する．

2.2 処理の流れ

図 1 に提案手法を用いたフィルタリング処理の流れを示す．提案手法を利用する際，事前準備として以下の作業が必要となる．

(1) 利用者が対象システムのソースコードを検出ツールに入力する．
(2) 検出ツールは入力されたソースコードからクローンセットを検出する．

これらの準備作業によって，提案手法が入力として求めるクローンセットの位置情報が特定される．同一プロジェクトに対し複数人の利用者が存在する場合，この準備作業で特定したクローンセットの位置情報は再利用することが可能である．すなわち，代表者が事前準備を行うことで，

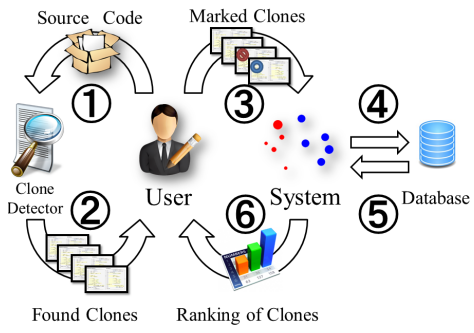


図 1 提案手法を用いたフィルタリング処理の流れ
Fig. 1 Overall workflow of filtering with the proposed method.

© 2013 Information Processing Society of Japan

```
struct blob *lookup_blob(struct object *obj)
{
   if (!obj->type)
      obj->type = OBJ_BLOB;
   if (obj->type != OBJ_BLOB)
      return NULL;
   return (struct blob *) obj;
}
```

(a) ソースコード

```
STRUCT ID TIMES ID LPAREN STRUCT ID TIMES ID RPAREN
LBRACE
  IF LPAREN LNOT ID RPAREN
    ID ARROW ID EQUALS ID SEMI
  IF LPAREN ID ARROW ID NE ID RPAREN
    RETURN ID SEMI
  RETURN LPAREN STRUCT ID TIMES RPAREN ID SEMI
RBRACE
```

(b) トークン型列

図 2 トークン型列の例

Fig. 2 An example of token types seqneuce.

他の利用者は自ら検出処理を行うことなく提案手法を利用することができる．

個々の利用者が提案手法を利用する手順は以下のとおり．

(3) 検出ツールにより検出されたクローンセットの中のいくつかを，利用者がその有用性に応じて “有用” か “有用でない” かに分類し，登録する．

(4) 提案システムは利用者によって分類されたクローンセット（学習データ）を利用者ごとに，あるいは利用目的ごとに別々に構築されたデータベースに格納する．

(5) 提案システムはデータベースに格納されている学習データを分析し，機械学習を用いてそれらの特徴を抽出する．

(6) 提案システムは (5) で抽出した特徴をもとに，学習データに含まれていないクローンセットすべてに対し，それらの有用性を判別し，出力する．

2.3 機械学習手法

提案手法における機械学習手法は，クローンセット間の類似度の算出，ならびにそれを用いた有用性の判別の 2 つのステップからなる．以降でそれぞれのステップについて述べる．

2.3.1 クローンセット間の類似度の算出

提案手法では，各クローンセットを構成するコード片のトークン型列（トークンの種類をアルファベットとする文字列）をもとにクローンセット間の類似度を算出する．これは，検出ツールがユーザ定義名の違いなどを吸収してクローンセットを検出する場合，同じクローンセットの中に含まれるコード片の間で文字列が一致しない可能性があるためである [4]．コード片のトークン型列を用いることで，細かな文字列の違いを吸収することが可能となり，算出するクローンセット間の類似度，および最終的な予測精度の向上が期待できる．**図 2** にトークン型列の例を示す．

提案手法では，類似度の算出に TF-IDF 値（Term Frequency - Inverse Document Frequency）[8] を用いる．以降の説明では，1 つのクローンセットを d，クローンセットの集合を D と表記する．

TF-IDF 値の算出に際し，まず各クローンセット d を構成するコード片のトークン型列から N-Gram を抽出する．たとえば，図 2 (a) のソースコードから得られるトークン型列（図 2 (b)）に対して $N = 3$ とすると，以下のような N-Gram が得られる．

```
STRUCT   ID   TIMES
         ID   TIMES   ID
              TIMES   ID   LPAREN
                      ID   LPAREN   STRUCT
                           LPAREN   STRUCT   ID
                                    ...
```

このようにして得られた N-Gram のうちの 1 つを t と表記する．次に，得られたすべての N-Gram に対して `term frequency` $tf(t,d)$ を式 (1) を用いて算出する．

$$tf(t,d) = \frac{|t : t \in d|}{|d|} \tag{1}$$

$tf(t,d)$ はあるクローンセット d 中に N-Gram t がどの程度の割合で出現するのかを表している．図 2 の場合，すべての N-Gram t に対して $tf(t,d)$ を算出すると以下のようになる．

```
STRUCT ID TIMES      :  0.06818181818181818
ID TIMES ID          :  0.045454545454545456
TIMES ID LPAREN      :  0.022727272727272728
ID LPAREN STRUCT     :  0.022727272727272728
...
```

続いて，`inverse document frequency` $idf(t,D)$ を式 (2) を用いて算出する．$idf(t,D)$ は t がどれだけ多くのクローンセット内に出現するのかを表す指標であり，多くのクローンセット内に出現している場合にその値が小さくなる．

$$idf(t,D) = \log \frac{|D|}{1 + |d \in D : t \in d|} \tag{2}$$

次に，式 (1) および式 (2) を用いて，$tf\,idf$ を算出する（式 (3)，式 (4)）．

$$tf\,idf(t,d,D) = tf(t,d) \times idf(t,D) \tag{3}$$

$$\overrightarrow{tf\,idf(d,D)} = [tf\,idf(t,d,D) \quad \forall t \in d] \tag{4}$$

$tf\,idf(t,d,D)$ は個々の N-Gram に対する TF-IDF 値を表しており，$\overrightarrow{tf\,idf(d,D)}$ は $tf\,idf(t,d,D)$ を連結して作成したベクトルである．

この $\overrightarrow{tf\,idf(d,D)}$ を用いて，2 つのクローンセット a，b 間の類似度 $nsim(a,b,D)$ を以下のように算出する．

$$sim(a,b,D) = \overrightarrow{tf\,idf(a,D)} \cdot \overrightarrow{tf\,idf(b,D)} \tag{5}$$

$$nsim(a,b,D) = \begin{cases} 0, & sim(a,b,D) = 0 \\ \dfrac{sim(a,b,D)}{|\overrightarrow{tf\,idf(a,D)}| \cdot |\overrightarrow{tf\,idf(b,D)}|}, & \text{otherwise} \end{cases} \tag{6}$$

© 2013 Information Processing Society of Japan

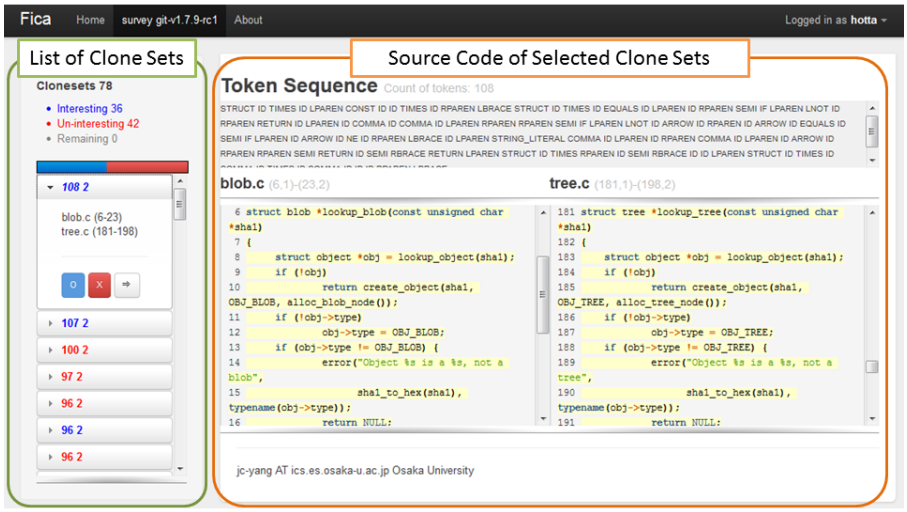


図 3 クローンセットの表示
Fig. 3 Fica showing clonesets.

2.3.2 有用性の判別

本項では，学習データとして与えられた分類済みのクローンセット集合を用いて，未分類のクローンセットが“有用”か否かを判別する方法について述べる．

学習データとなるクローンセットの集合を M，判別の対象となる未分類のクローンセットを t（$t \notin M$）と表記する．また，M に含まれるクローンセットのうち，“有用”と分類されているクローンセットの集合を M_i，“有用でない”と分類されているクローンセットの集合を M_{ni} とする．仮定より，$M_i \cap M_{ni} = \emptyset$，$M_i \cup M_{ni} = M$ である．このとき，M_i，M_{ni} のそれぞれに対し，式 (7) で定義する $poss(t, M_x)$（$M_x = M_i$ or M_{ni}）を算出する．

$$poss(t, M_x) = \begin{cases} 1, & |M_x| = 0 \\ \frac{\sum_{\forall m \in M_x} nsim(t, m, M_x)}{|M_x|}, & \text{otherwise} \end{cases} \quad (7)$$

この $poss(t, M_x)$ は，クローンセット t とクローンセット集合 M_x 中のクローンセットの類似度を表しており，この値が高いほど t が M_x に属する確率が高いといえる．提案手法では，$poss(t, M_i)$ と $poss(t, M_{ni})$ の値を比較し，$poss(t, M_i) \geq poss(t, M_{ni})$ の場合に t は“有用”と判定し，それ以外の場合は t は“有用でない”と判定する．

3. 実装

本章では，提案手法を実装したシステム Fica について述べる．Fica は提案するフィルタリング機能を有すると同時に，コードクローンを検出する機能をその内部に持つ．Fica 自体にコードクローン検出機能が備わっているが，利用者は Fica が内蔵する検出ツール以外の検出ツールを利用することもできる．

Fica は Web システムとして実装されており，利用者は一般的な Web サイトと同じようにログイン処理を行う．このログイン処理によって Fica は個々の利用者を判別する．Fica を利用する際，まず代表者が対象システムのソースコードをアップロードする．Fica はアップロードされたソースコードを解析し，コードクローンの検出を行う．なお，コードクローンの検出には `generalized suffix tree` [9] を用いている．

コードクローンの検出が終了すると，Fica は検出したクローンセットを利用者に提示する．クローンセットの表示例を**図 3** に示す．画面左側に検出されたクローンセットが一覧表示されており，選択されたクローンセットのソースコードが右側に表示される．さらに，画面中の青い ◯ あるいは赤い × を選択することで，そのクローンセットが利用者にとって“有用”か“有用でない”かを指定することができる．利用者がクローンセットを分類するたびに，Fica は学習データを更新し，再度予測を行って利用者にその結果を提示する．予測の結果はクローンセット一覧に表示される各クローンセットの文字の色に対応しており，青が“有用”なクローンセットを，赤が“有用でない”クローンセットをそれぞれ表している．これにより，利用者は Fica による予測結果をリアルタイムに知ることが可能である．

また，提案手法ではクローンセット間の類似度算出を行う際，N-Gram の N の値を指定する必要がある．Fica は $N = 5$ を規定値としているが，この値は自由に変更できる．

4. 評価実験の概要

提案手法の有効性を評価するために，オープンソースソフトウェアを対象とした評価実験を行った．本実験における評価項目は以下のとおり．

項目 A：利用者による“有用”か“有用でない”かの判別基準の違い

項目 B：提案手法による予測の精度

以降本章では，実験の方法，ならびに実験で用いる評価

© 2013 Information Processing Society of Japan

表 1 実験対象ソフトウェア
Table 1 Open source projects used in experiments.

Project & Version	CSs	Len	LOC	Tokens	Files
git v1.7.9-rc1	78	48	153,388	829,930	315
xz 5.0.3	36	48	25,873	113,894	113
bash 4.2	105	48	133,547	494,248	248
e2fsprogs 1.41.14	62	64	99,129	442,978	274
Total	281		411,937	1,881,050	950

指標ついて述べる．実験の結果，ならびにその考察については，それぞれ 5 章ならびに 6 章で述べる．

4.1 実験方法

実験対象のソフトウェアを**表 1** に示す．これらのプロジェクトに含まれるファイルのうち，C 言語のソースファイルのみを実験の対象とした．また，表中の CSs はそのプロジェクトから検出されたクローンセットの総数を，Len はコードクローン検出時の最小一致トークン数をそれぞれ表している．ここで最小一致トークン数とは，コードクローンとして検出されるコード片の最小の大きさ（トークン数）である．他のコードクローン検出ツールの多くが 5 行程度以上の長さを検出するコードクローンの大きさとして採用していることから，本実験においても 5 行程度の大きさのコード片がコードクローンとして検出されるように Len の値を規定した．いくつかの値を試した結果，本実験では Len の値として 48 を採用した．また本実験では `e2fsprogs` に対してのみ異なる最小一致トークン数を用いているが，これは最小一致トークン数 48 でコードクローンの検出を行った際に多くのクローンセットが検出され，実験に過度な負担がかかると判断したためである．さらに，本実験では N-Gram の N の値として，Fica の規定値である $N = 5$ を用いた．また本実験では，ソフトウェア開発やコードクローン検出技術について異なる知識，経験を持つ 8 人の被験者（被験者 A〜被験者 H）に協力を依頼した．

実験を行うにあたり，あらかじめ 4 つの対象ソフトウェアのソースコードを Fica にアップロードし，コードクローンを検出した．その後，8 人の被験者がそれぞれ，検出された全 281 のクローンセットを “有用” もしくは “有用でない” のいずれかに分類した．また，被験者は Fica が機械学習を用いてクローンセットを自動的にフィルタリングするシステムであることは知らされているが，その内部の具体的な処理内容については知らされていない．すなわち，被験者は Fica がトークン型列の比較に基づいてフィルタリングを行うことを認識していない．

4.2 評価指標

本実験では，被験者によって分類されたクローンセットのうち一定数をランダムに抽出し，それらを学習データとして残りのクローンセットの分類を Fica に予測させ，その精度を評価する．以降，学習データに含まれないクローンセットの集合を**評価対象データ**と表記する．

はじめに，予測精度の評価方法を述べるうえで必要となる諸用語を定義する．

それぞれのプロジェクトについて，CS をそのプロジェクトから検出されたすべてのクローンセットからなる集合，CS_t を学習データとして抽出されたクローンセットからなる集合，CS_e を評価対象データを構成するクローンセットの集合と定義する．このとき，定義より式 (8) および式 (9) がそれぞれ成立する．

$$CS_t \cap CS_e = \emptyset \tag{8}$$

$$CS_t \cup CS_e = CS \tag{9}$$

また，$judge_{sub}(d)$ を被験者 sub がクローンセット d を “有用” と判別しているときのみ真となる論理式とし，$judge_{fica}(d)$ を Fica がクローンセット d を “有用” と予測したときのみ真となる論理式とする．このとき，クローンセットの集合 D，および被験者 sub に対して，$True_{sub}(D)$, $False_{sub}(D)$, $Positive_{sub}(D)$, $Negative_{sub}(D)$, $TruePositive_{sub}(D)$，ならびに $FalseNegative_{sub}(D)$ をそれぞれ以下のように定義する．

$$True_{sub}(D) := \{\, d \in D \mid judge_{sub}(d)\} \tag{10}$$

$$False_{sub}(D) := \{\, d \in D \mid \neg judge_{sub}(d)\} \tag{11}$$

$$Positive_{sub}(D) := \{\, d \in D \mid judge_{fica}(d)\} \tag{12}$$

$$Negative_{sub}(D) := \{\, d \in D \mid \neg judge_{fica}(d)\} \tag{13}$$

$$TruePositive_{sub}(D) := \{\, d \mid d \in True_{sub}(D) \wedge d \in Positive_{sub}(D)\} \tag{14}$$

$$FalseNegative_{sub}(D) := \{\, d \mid d \in False_{sub}(D) \wedge d \in Negative_{sub}(D)\,\} \tag{15}$$

このとき，被験者 sub に対して，本実験で用いる評価指標，$Accuracy_{sub}$，$Recall_{tp_{sub}}$，$Precision_{tp_{sub}}$，$Recall_{fn_{sub}}$，$Precision_{fn_{sub}}$ をそれぞれ以下のように定義する．

$$Accuracy_{sub} := \frac{|TruePositive(CS_e)| + |FalseNegative(CS_e)|}{|CS_e|} \tag{16}$$

$$Recall_{tp_{sub}} := \frac{|TruePositive(CS_e)|}{|True(CS_e)|} \tag{17}$$

$$Precision_{tp_{sub}} := \frac{|TruePositive(CS_e)|}{|Positive(CS_e)|} \tag{18}$$

$$Recall_{fn_{sub}} := \frac{|FalseNegative(CS_e)|}{|False(CS_e)|} \tag{19}$$

$$Precision_{fn_{sub}} := \frac{|FalseNegative(CS_e)|}{|Negative(CS_e)|} \tag{20}$$

$Accuracy_{sub}$ は Fica による予測が被験者 sub による分類とどの程度一致しているのかを表す指標であり，その一致の

© 2013 Information Processing Society of Japan

度合いが大きければ $Accuracy_{sub}$ は高い値となる．*Recall* は被験者 *sub* が “有用”/“有用でない” と分類したすべてのクローンセットのうち，Fica が正しく予測できたものがどの程度存在するのかを表す指標（再現率）であり，$Recall_{tp_{sub}}$ は “有用” なものを正解としたとき，$Recall_{fn_{sub}}$ は “有用でない” ものを正解としたときの値である．*Precision* は Fica が “有用”/“有用でない” と判断したもののうち，被験者 *sub* による分類と一致したものがどの程度存在するのかを表す指標（適合率）であり，$Precision_{tp_{sub}}$，$Precision_{fn_{sub}}$ はそれぞれ “有用” なもの，“有用でない” ものを正解としたときの値である．

4.3 計測手順

評価指標の計測手順は以下のとおり．

(1) 被験者によって分類されたクローンセットのうち，一定数をランダムに抽出し，学習データとして Fica に登録する．
(2) それぞれの被験者について，学習データとして登録されていないクローンセット（以降，評価対象データ）の分類を Fica に予測させる．
(3) Fica の予測結果と，実際に被験者が分類を行った結果を比較し，評価指標値を算出する．

以上の (1)〜(3) を 256 回繰り返し行う．

Accuracy については，学習データとして抽出するクローンセット数を変化させながら計測を行う．一方，*Recall*，*Precision* については，全クローンセット数の 20%を学習データとして使用する．

5. 実験結果

5.1 項目 A：利用者による判別基準の違いについて

各被験者がそれぞれのプロジェクトに対して “有用” と判断したクローンセット数，ならびに “有用でない” と判断したクローンセット数をそれぞれ**表 2** に示す．表中の ‘CSs’ は各プロジェクトのクローンセット数を表しており，‘xx/yy’ は xx 個のクローンセットが “有用” と判別され，yy 個のクローンセットが “有用でない” と判別されたことを表している．表 2 より，被験者によって “有用” と判別したクローンセット数に大きな差があることが分かる．特に被験者 A は他の被験者と比べて “有用” と判別したクローンセット数が少ないことが分かる．

次に，**図 4** にクローンセット間の類似関係を force-directed アルゴリズム [10] を用いてグラフ化したものを示す．グラフ中の各頂点が 1 つのクローンセットを表しており，青に塗られた頂点は被験者によって “有用” と分類されたクローンセットを，赤に塗られた頂点は “有用でない” と分類されたクローンセットをそれぞれ表している．また，2 つの色がともに塗られている頂点は被験者によって分類結果が異なるクローンセットを表しており，各色の面積比がそれぞれの分類を行った被験者の比率に対応している．なお，中心部に近い色が過半数を占めた分類を表している．たとえば，中心部が赤で，外側に一部青が存在するクローンセットの場合，多くの被験者がそのクローンセットを “有用でない” と分類しているが，一部の被験者がそのクローンセットを “有用” と分類していることを意味する．

クローンセット間の類似度（式 (6)）が一定値以上の場合に，クローンセット間に辺が存在する．辺が存在する基準値として，今回は 0.028 を設定している．類似度の高さがクローンセット間の結び付きの強さに対応しており，類似するクローンセットほどそれらの距離が短くなっている．

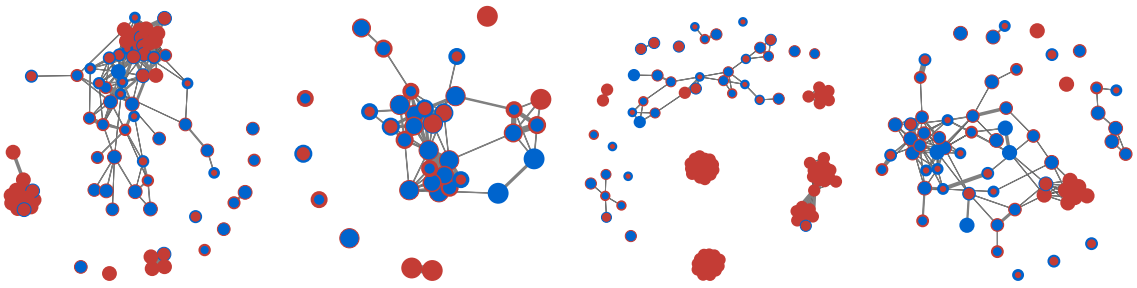
図 4 より，“有用でない” クローンセットは “有用” なクローンセットと比較して互いに類似度が高く，1 つの群として密集していることが分かる．これは，“有用でない” コードクローンはそのトークン型列の特徴によって，`switch-case` 文の集合や代入文の集合などのいくつかのカテゴリに分類できる可能性を示唆している．一方，“有用” と分類されたコードクローンは密集しておらず，カテゴリに分類することが難しい．

さらに図 4 より，被験者によって “有用” か “有用でない” かの判別が分かれたクローンセットが多数存在することが分かる．また “有用でない” と判断されたクローンセットについては，それらが密集する群において，すべての被験者が “有用でない” と判断したクローンセットが多数を占めている．一方で，密集していないものや “有用” と判断したクローンセットについては判断が分かれやすい傾向にあることが分かる．この結果から，多くの被験者から “有用でない” と判断されたクローンセットは多数存在し，かつそれらの間には高い類似性がある一方で，多くの被験者から “有用” と判断されたクローンセットは少ないといえる．すなわち，すべての利用者に対し普遍的に “有用” であるコードクローンはほとんど存在しないといえる．このことは “有用” なコードクローンの特定を行う際に利用者ごとの判断基準の違いを反映させることの必要性を示唆している．

表 2 被験者ごとの分類結果
Table 2 The number of clone sets judged as useful/unuseful.

	git	xz	bash	e2fprogs	total
CSs	78	36	105	62	281
A	9/69	7/29	5/100	15/47	36/245
B	33/45	24/12	27/78	33/29	117/164
C	16/62	18/18	23/82	33/29	90/191
D	40/38	28/8	23/82	36/26	127/154
E	39/39	30/6	25/80	48/14	142/139
F	19/59	15/21	21/84	29/33	84/197
G	29/49	17/19	24/81	38/24	108/173
H	36/42	27/9	31/74	51/11	145/136

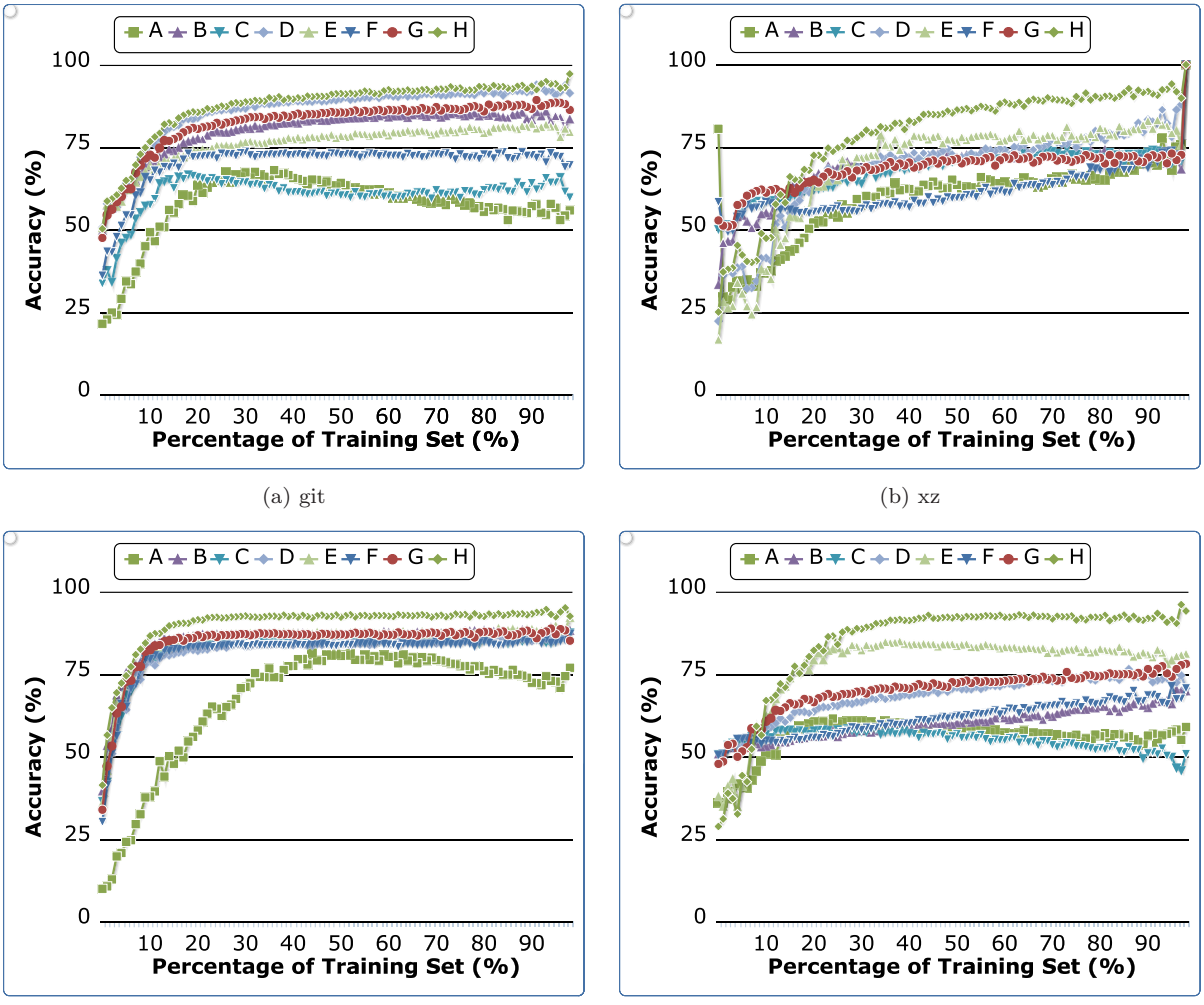
© 2013 Information Processing Society of Japan



(a) git (b) xz (c) bash (d) e2fsprogs

図 4 クローンセット間の類似度を表すグラフ

Fig. 4 Force-directed graphs showing code clone similarity.



(a) git (b) xz

(c) bash (d) e2fsprogs

図 5 予測精度（プロジェクトごと）

Fig. 5 Result of machine learning (for each project).

5.2 項目 B：提案手法による予測の精度について

5.2.1 *Accuracy*

図 5 にそれぞれのプロジェクトにおける *Accuracy* の計測結果を示す．グラフの X 軸が全クローンセットのうち学習データとして使用したクローンセットの割合を表しており，Y 軸が *Accuracy* の平均値を表している．また，図中の A〜H はそれぞれの被験者を表している．図 5 より，Fica の予測精度は被験者により，またプロジェクトにより大きく異なっていることが分かる．すべてのプロジェクトに共通してみられる傾向として，グラフの右側ほど，すなわち学習データとして与えるクローンセットの割合を増やすほど，予測精度が向上している．

4 つのプロジェクトのうち，`bash` に対する予測精度が最も高く，被験者 A を除くすべての被験者に対して 15%以上のクローンセットを学習データとして与えることで 80%以上の精度で予測を行えた．`bash` から検出されたクローン

© 2013 Information Processing Society of Japan

セット数は 105 であるため，16 のクローンセットに対して分類を行うことで 80%以上の精度で予測を行えるという結果となった．

`git` および `e2fsprogs` に対する実験では，予測精度は被験者によって大きく異なるという結果となった．被験者 H に対しては 20%ほどの学習データを与えることで 80%以上の精度を得た，被験者 A および C に対する予測では 60%程度の精度であった．また，`xz` に対する実験では，80%以上の予測精度を得るために，おおむね 90%以上のクローンセットを学習データとして与える必要があるという結果となった．

さらに，各プロジェクトから検出された計 281 のクローンセットをすべて同時に用い，4.3 節で述べた方法で評価を行った．その結果を**図 6** に示す．図 5 および図 6 より，`bash` を除く 3 つのプロジェクトについては，プロジェクトごとに機械学習を適用するよりも高い精度で予測することができた，という結果となった．

5.2.2 *Recall*，*Precision*

すべての被験者に対する *Recall* および *Precision* の計測結果をプロジェクトごとにまとめて箱ひげ図として表したものを**図 7** に示す．図 7 より，`git` および `bash` については，“有用でない” クローンセットに対する再現率，適合率が “有用” なクローンセットよりも高くなる傾向がみられた．一方，`xz` および `e2fsprogs` については “有用” なクローンセットに対する再現率，適合率と “有用でない” クローンセットに対する再現率，適合率の間に大きな差異はないという結果となった．

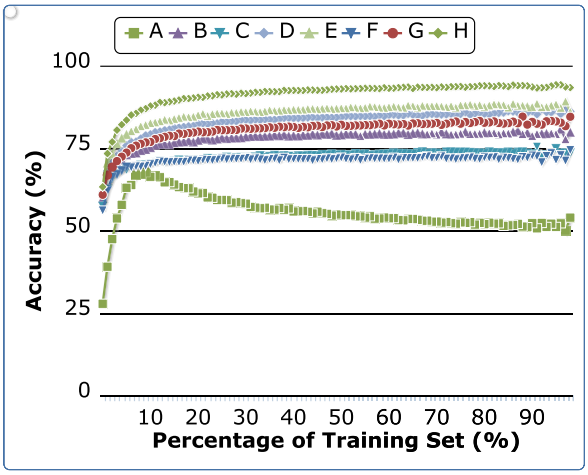
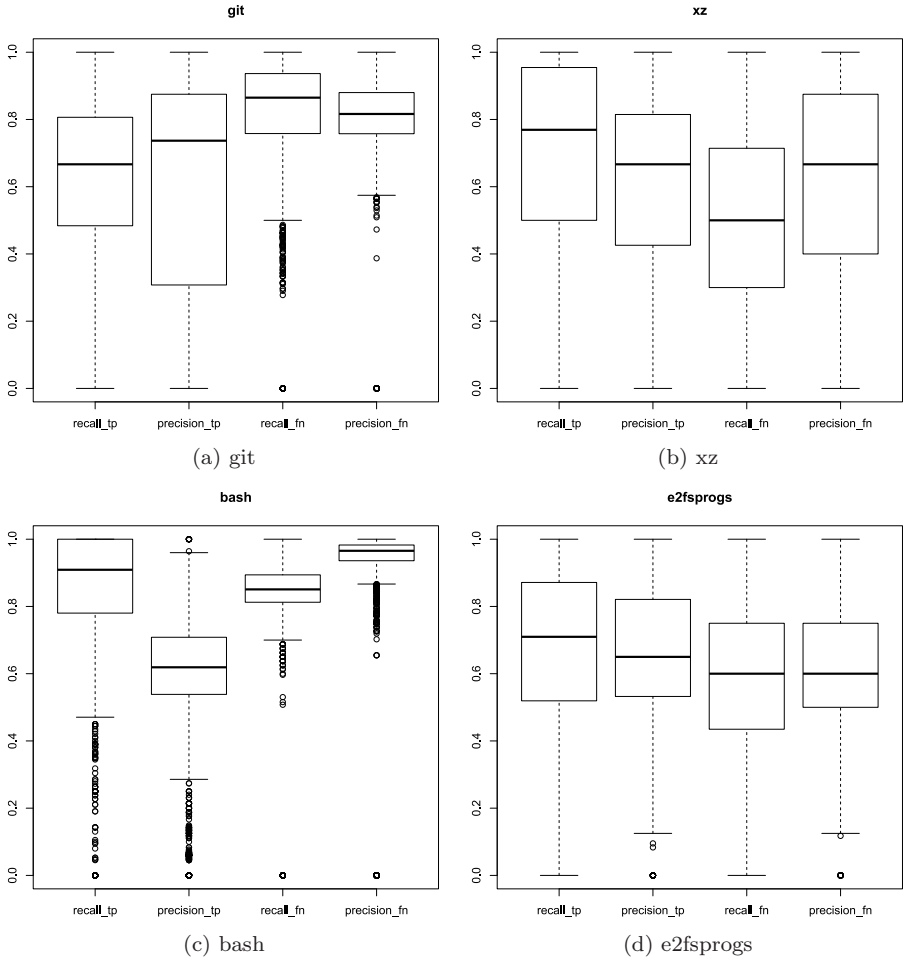


図 6 予測結果（全プロジェクト）
Fig. 6 Merged result of all projects.



(a) git (b) xz

(c) bash (d) e2fsprogs

図 7 再現率および適合率の計測結果
Fig. 7 Box plots of recall and precision.

© 2013 Information Processing Society of Japan

```
130     }
131     return LZMA_PROG_ERROR;
132 }
133 static void
134 block_encorder_end(lzma_coder *coder, ... )
135 {
136     lzma_next_end(&coder->next, allocator);
137     lzma_free(coder, allocator);
138     return;
139 }
140 static lzma_ret
141 block_encorder_update(lzma_coder *coder, ... ,
142             const lzma_filter *filters,
143             const lzma_filter * reversed_filters)
144 {
145     if (coder->sequence != SEQ_CODE)
```

(a) src/liblzma/common/block_encoder.c

```
61     }
62     return LZMA_OK;
63 }
64 static void
65 alone_encoder_end(lzma_coder *coder, ... )
66 {
67     lzma_next_end(&coder->next, allocator);
68     lzma_free(coder, allocator);
69     return;
70 }
71 static lzma_ret
72 alone_encoder_init(lzma_next_coder *next, ... ,
73             const lzma_options_lzma *options)
```

(b) src/liblzma/common/alone_encoder.c

```
474     else
475             lzma_free(coder->lz.coder, allocator);
476     lzma_free(coder, allocator);
477     return;
478 }
479 static lzma_ret
480 lz_encoder_update(lzma_coder *coder, ... ,
481         const lzma_filter * filters_null,
482         const lzma_filter * reversed_filters)
483 {
484     if (coder->lz.options_update == NULL)
```

(c) src/liblzma/lz/lz_encoder.c

図 8 ソースコード（xz）

Fig. 8 Example of source code (xz).

```
48     return 0;
49 }
50 static int parse_block(const char *request,
51     const char *desc, const char *str, blk_t *blk)
52 {
53     char *tmp;
54     *blk = strtoul(str, &tmp, 0);
55     if (*tmp) {
56             com_err(request, 0, "Bad %s - %s#, desc, str);
57             return 1;
58     }
59     return 0;
60 }
61 static int check_brel(char *request)
```

(a) test/progs/test_rel.c

```
44     return 1;
45 }
46 static int parse_inode(const char *request,
47     const char *desc, const char *str, ext2_ino_t *ino)
48 {
49     char *tmp;
50     *ino = strtoul(str, &tmp, 0);
51     if (*tmp) {
52             com_err(request, 0, "Bad %s - %s", desc, str);
53             return 1;
54     }
55     return 0;
56 }
57 void do_create_icount(int argc, char **argv)
```

(b) test/progs/test_icount.c

```
68 static errcode_t test_write_blk64(io_channel channel,
69   unsigned long block, int count, const void *data);
70 static errcode_t test_flush(io_channel channel);
71 static errcode_t test_write_byte(io_channel channel,
72   unsigned long offset, int count, const void *buf);
73 static errcode_t test_set_option(io_channel channel,
74   const char *option, const char *arg);
75 static errcode_t test_get_stats(io_channel channel, ...);
76 static struct struct_io_manager struct_test_managet = {
```

(c) lib/ext2fs/test_io.c

```
102 static errcode_t unix_write_blk(io_channel channel,
103   unsigned long block, int count, const void *data);
104 static errcode_t unix_flush(io_channel channel);
105 static errcode_t unix_write_byte(io_channel channel,
106   unsigned long offset, int size, const void *data);
107 static errcode_t unix_set_option(io_channel channel,
108   const char *option, const char *arg);
109 static errcode_t unix_get_stats(io_channel channel, ... );
110 static void reuse_cache(io_channel channel, ...
```

(d) lib/ext2fs/unix_io.c

図 9 ソースコード（e2fsprogs）

Fig. 9 Example of source code (e2fsprogs).

6. 考察

6.1 予測に失敗したクローンセットについて

評価実験の結果，Fica はおおむね 70%～90%の精度で予測できることが確認できた．この結果は，適切に予測できなかった事例がいくつか存在することを意味している．

図 8 にそのような事例の 1 つを示す．この例は `xz` から得られた例である．この 3 つのコード片から，Fica 内部のコードクローン検出器は 2 つのクローンセットを検出した．

α：(a) 130-142 & (b) 61-72

β：(a) 136-145 & (c) 475-484

クローンセット α は 1 つの関数の全体を含むが，クローンセット β は関数の一部分のみを含む．コードクローンを除去を目的としたリファクタリングを適用を考えた場合，関数全体を含むクローンセットは抽出が容易であるため，α のように関数の全体を含むクローンセットの方が除去しやすいといえる．したがって，‘コードクローンの除去を目的としたリファクタリング’ を考えた場合，α の方が “有用” と見なされる可能性が高い．事実，8 人の被験者のうち 7 人が，α を “有用” と分類している．他方，β は 8 人中 6 人が “有用でない” と分類している．しかしながら，α と β の間の類似度を式 (5) を用いて算出すると，43%という値が得られる．この値は他のクローンセットどうしの組合せから得られる類似度と比較して高い値である．このため，Fica は α が “有用” であるという学習データが存在した場合，β も同様に “有用” と判別する可能性が高い．これが Fica がこの例に対して適切に予測ができない要因である．

図 9 に Fica が適切に予測ができなかった別の例を示す．この例には (a) と (b) から構成されるクローンセット（クローンセット γ），および (c) と (d) から構成されるクローンセット（クローンセット δ）の 2 つのクローンセットが存在する．γ は関数の本文を含むクローンセットであるのに対し，δ は関数宣言の並びのみから構成されるクローンセットである．したがって，人間の目にはこの 2 つのクローンセットは大きく異なるものとして認識される．

しかし，この 2 つのクローンセット間には，共通の N-Gram（e.g. `STATIC ID ID LPAREN CONST, CONST ID TIMES ID COMMA`）が多数存在するため，この 2 つのクローンセットに対して類似度を算出すると高い値が得られる．すなわち，Fica はこれら 2 つのクローンセットを類似するものとして

© 2013 Information Processing Society of Japan

扱っている．このように，人間の目には大きく異なるものと認識されるクローンセットを Fica が類似したものであると認識していることが，Fica が誤った予測を引き起こす要因となっていると考えられる．

これらの問題点は，ともに Fica が類似度の算出をトークン型列の比較に基づいて行っていることに起因する．トークン型列の比較に加え，他のメトリクスを組み合わせて類似度を算出することによって，予測精度の向上が見込まれる．

6.2 被験者ごとの判断基準の違いについて

今回の実験における “有用” か否かの分類の方針について，被験者に対して聞き取り調査を行った．その結果，多くの被験者から『“有用” なものを選択する』，よりも，『“有用でない” ものを除外して，残ったものを “有用” とする』という方針で分類を行っていたという意見が得られた．“有用でない” クローンセットの分類基準として以下のような意見が得られた．

- クローンセットを構成するコード片が，ある関数の途中から別の関数の途中までというように，関数をまたぐコード片によって構成されている．
- クローンセットを構成するコード片が変数宣言や代入文などの単純な処理の繰返しによって構成されている．
- クローンセットを構成するコード片内で行われている処理が本質的な処理ではない（エラーチェック，標準出力への出力など）．

このように，“有用でない” クローンセットの分類は被験者ごとに明確な基準の下で行われていたことが，“有用でない” クローンセットが互いに類似した群を形成していた要因であると考えられる．その一方で，“有用” なクローンセットは “有用でない” と分類されなかったクローンセットであるため，“有用” なクローンセットであると分類する明確な基準が存在せず，“有用” なクローンセット間の類似度が低くなったといえる．

6.3 予測精度について

6.3.1 *Recall*，*Precision*

`git` および `bash` については “有用でない” クローンセットに関する再現率，適合率が高いという結果が得られた．その要因として以下の点があげられる．

- 図 4 より，“有用でない” クローンセットは “有用” なものと比較してクローンセット間の類似度が高いことが分かる．よって，“有用でない” クローンセットについてはそのクローンセットと類似する “有用でない” クローンセットが存在する可能性が高い．したがって，“有用” なクローンセットに対する予測精度よりも “有用でない” クローンセットに対する予測精度の方が高くなる傾向にあると考えられる．
- これらのプロジェクトでは，“有用” と分類されたクローンセットよりも “有用でない” と分類されたクローンセットの方が多い傾向にある．そのため，“有用でない” クローンセットが学習データに多数含まれる可能性が高い．

一方，`xz` および `e2fsprogs` については，“有用” なクローンセットに対する計測結果と “有用でない” クローンセットに対する計測結果の間に大きな差異は存在しない．これは，これらのプロジェクトでは “有用” と判断されたクローンセットが多いため，学習データにも “有用” なクローンセットが多く含まれていた可能性が高いこと，および “有用” なクローンセットは他の “有用” なクローンセットとの類似度が低い傾向にあることに起因すると考えられる．

6.3.2 *Accuracy*

実験結果より，`bash` を除く 3 つのプロジェクトについては，それぞれのプロジェクトごとに予測を行った場合より，4 つのプロジェクトから検出されたクローンセットをすべて用いて予測を行った方が予測精度が高いという結果が得られた．

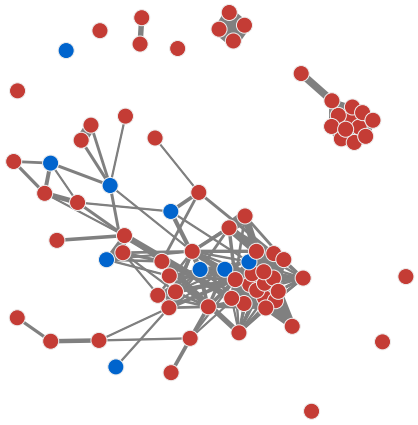
その要因として，被験者が “有用でない” と判断したクローンセット数の違いがあげられる．`bash` 以外の 3 つのプロジェクトは `bash` と比べて “有用でない” と分類されたクローンセット数が少なくなっている．これらのプロジェクトについては，他のプロジェクトから得られたクローンセットも用いて予測を行うことで，学習データ中に含まれる “有用でない” クローンセットの量が増加する可能性が高い．6.3.1 項で述べたように，“有用でない” クローンセットに対する予測精度の方が “有用” なクローンセットに対する予測精度より高くなる傾向にある．そのため，学習データ中の “有用でない” クローンセットの数が増加したことが，`bash` 以外のプロジェクトにおいて他のプロジェクトのクローンセットをあわせて使用した場合の方が予測精度が向上した要因の 1 つであると考えられる．

また，実験結果より，Fica による予測の精度は被験者によって大きく異なるという結果が得られた．特に，被験者 A に対する予測精度は他の被験者に対する予測精度と比較して著しく低いという結果となった．

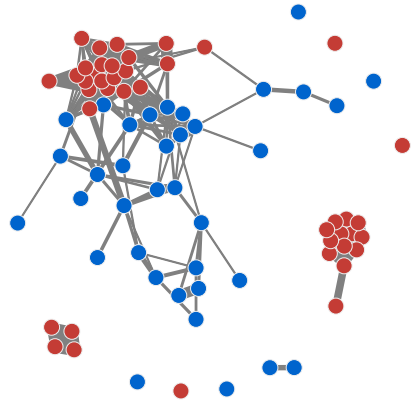
その要因として，被験者 A の行った分類が類似度の高いクローンセット間で一貫していないことがあげられる．**図 10** に `git` における被験者 A と被験者 H の分類結果を示す．図 10 より，高い予測精度が得られた被験者 H の分類結果は類似度の高いものどうしの間でほぼ一貫性が保たれている一方，被験者 A の分類結果は一貫性が保たれておらず，“有用でない” と分類されたクローンセットが密集している群の中に “有用” と分類されたクローンセットが含まれていることが分かる．

このように被験者 A の分類において一貫性が保たれて

© 2013 Information Processing Society of Japan



(a) 被験者 A



(b) 被験者 H

図 10 git における Force Directed Graph
Fig. 10 Force directed graphs on git.

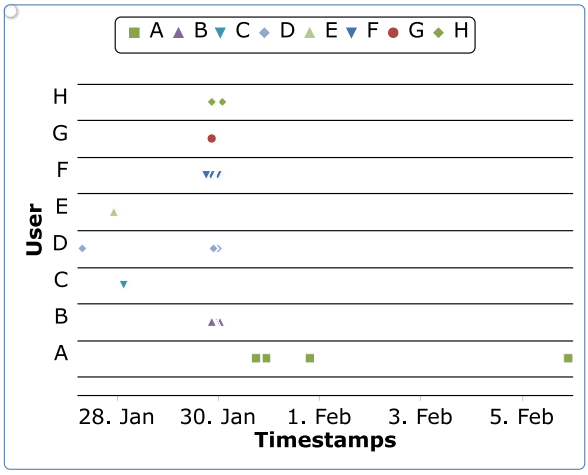


図 11 実験に要した時間
Fig. 11 Time required for the experiment.

いない理由の 1 つとして，被験者 A が実験に要した時間の長さがあげられる．図 11 にそれぞれの被験者が実験を行う際に要した時間を示す．図の X 軸は日時を表しており，Y 軸はそれぞれの被験者を表している．また，それぞれの点が被験者がある 1 つのクローンセットに対する分類を入力した時刻を表している．図 11 より，多くの被験者は数時間程度で実験を終えている一方，被験者 A は実験に約 1 週間を要していることが分かる．また，被験者 A は実験を複数回に分けて行っており，それぞれの実験の間に時間が空いていることが分かる．このような理由から，被験者 A が “有用” か否かを判別する基準の一貫性を保つことができず，結果として被験者 A が行った分類が他の被験者と比較して一貫性が低くなった可能性がある．

7. 関連研究

Higo らは複数のメトリクスを用いてリファクタリングしやすいコードクローンを特定する手法を提案し，またそれを実装したツール Aries を開発した [6]．また，Koschke はいくつかのメトリクス値と機械学習技術をコードクローン検出に応用することで，ライセンスに違反して流用されているソースコードの特定を支援する手法を提案している [11]．Higo らの手法や Koschke の手法は特定の目的に特化した手法であるため，それらの手法が目的としている領域においては提案手法と比較して高い精度で予測を行うことが可能である．これに対し，提案手法は様々な使用用途に対応することが可能であるため，汎用性においてこれらの手法に勝るといえる．

Balazinska らはコードクローン関係にあるメソッドをそれらの構造などによって 18 のカテゴリに分類する手法を提案している [12]．また，Bellon らはコードクローンをその一致の度合いによって分類した 3 つのカテゴリを定義している [3]．このほかに，機械学習やテキストマイニングの技術をコードクローン検出に応用することでコードクローンを分類する手法がいくつか提案されている [7], [13], [14]．これらの分類手法は一般的な分類を求めているのに対し，提案手法は個々の利用者ごとに異なる分類を行っているという点が異なる．

Jiang らは CCFinder によって得られた検出結果に対してデータマイニング手法を適用することでコードクローンをフィルタリングする手法を提案している [15]．Jiang らの手法はコードクローンに対してフィルタリングを行うという点は提案手法と共通しているが，利用者による手動の分類が必要でないという点において異なる．提案手法は利用者による手動の分類が必要なため Jiang らの手法と比べて利用者の負担が大きくなるものの，利用者ごとの判断基準の違いを繊細に反映させることができる．

8. あとがき

本稿では，機械学習を用いたコードクローンの自動フィルタリング手法を提案した．提案手法はその利用者にいくつかのコードクローンを “有用” か否かに分類させ，そのデータを学習することで未分類のコードクローンが “有用” か否かを自動的に判別する．提案手法を Web ベースのシステム Fica として実装し，評価実験を行った．その結果，提案手法の予測精度はおおむね 70%を超え，被験者やプロ

© 2013 Information Processing Society of Japan

ジェクトによっては90%以上の精度で予測ができることを確認した．さらに，あるコードクローンが “有用” か “有用でない” かの判断基準が利用者によって異なるということを確認した．

今後の課題として，類似度以外のメトリクス値を組み合わせることで予測精度を向上させることがあげられる．また，現在は “有用” あるいは “有用でない” の二択による学習データ作成を行っているが，これを拡張して，“有用さ” の度合いによる順位付けが可能な分類方式や，タグを用いた分類方式などに対応することを考えている．さらに，Ficaを公開し，多くの研究者，開発者に利用してもらうことで，さらなる評価実験を行いたいと考えている．

参考文献

[1] 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，電子情報通信学会論文誌，Vol.J91-D, No.6, pp.1465–1481 (2008).

[2] 神谷年洋，肥後芳樹，吉田則裕：コードクローン検出技術の展開，コンピュータソフトウェア，Vol.28, No.3, pp.28–42 (2011).

[3] Bellon, S., Koschke, R., Antniol, G., Krinke, J. and Merlo, E.: Comparison and Evaluation of Clone Detection Tools, *IEEE Trans. Softw. Eng.*, Vol.31, No.10, pp.804–818 (2007).

[4] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: A multi-linguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol.28, No.7, pp.654–670 (2002).

[5] 肥後芳樹，吉田則裕：コードクローンを対象としたリファクタリング，コンピュータソフトウェア，Vol.28, No.4, pp.42–56 (2011).

[6] Higo, Y., Kusumoto, S. and Inoue, K.: A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system, *Journal of Software Maintenance and Evolution: Research and Practice*, Vol.20, No.6, pp.435–461 (2008).

[7] Tairas, R. and Gray, J.: An information retrieval process to aid in the analysis of code clones, *Empirical Software Engineering*, Vol.14, pp.33–56 (2009).

[8] Jones, K.: A statistical interpretation of term specificity and its application in retrieval, *Journal of Documentation*, Vol.28, No.1, pp.11–21 (1972).

[9] Ukkonen, E.: On-Line Construction of Suffix Trees, *Algorithmica*, Vol.14, No.3, pp.249–260 (1995).

[10] Fruchterman, T. and Reingold, E.: Graph drawing by force-directed placement, *Software: Practice and experience*, Vol.21, No.11, pp.1129–1164 (1991).

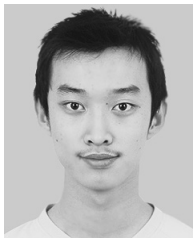
[11] Koschke, R.: Large-Scale Inter-System Clone Detection Using Suffix Trees, *Proc. 16th European Conference on Software Maintenance and Reengineering*, pp.309–318 (2012).

[12] Balazinska, M., Merlo, E., Dagenais, M., Lague, B. and Kontogiannis, K.: Measuring clone based reengineering opportunities, *Proc. 6th International Software Metrics Symposium*, pp.292–303 (1999).

[13] Marcus, A. and Maletic, J.: Identification of high-level concept clones in source code, *Proc. 16th Annual International Conference on Automated Software Engineering*, pp.107–114 (2001).

[14] Kuhn, A., Ducasse, S. and Gírba, T.: Semantic clustering: Identifying topics in source code, *Information and Software Technology*, Vol.49, No.3, pp.230–243 (2007).

[15] Jiang, Z. and Hassan, A.: A framework for studying clones in large software systems, *Proc. 7th International Working Conference on Source Code Analysis and Manipulation*, pp.203–212 (2007).



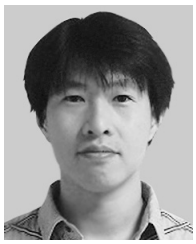
楊 嘉晨

平成23年中国上海交通大学軟件学院軟件工程専業卒業．現在，大阪大学大学院情報科学研究科博士前期課程在学中．コードクローン分析に関する研究に従事．



堀田 圭佑

平成22年大阪大学基礎工学部情報科学科卒業．平成24年同大学大学院情報科学研究科博士前期課程修了．現在，同研究科博士後期課程在学中．コードクローン分析に関する研究に従事．IEEE会員．



肥後 芳樹 （正会員）

平成14年大阪大学基礎工学部情報科学科中退．平成18年同大学大学院情報科学研究科博士後期課程修了．日本学術振興会特別研究員を経て，平成19年大阪大学大学院情報科学研究科コンピュータサイエンス専攻助教．博士（情報科学）．コードクローン分析，リファクタリングに関する研究に従事．IEEE会員．

© 2013 Information Processing Society of Japan



井垣 宏 （正会員）

平成 12 年神戸大学工学部電気電子工学科卒業．平成 14 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．平成 17 年同研究科博士後期課程修了．同年同研究科特任助手．平成 18 年南山大学数理情報研究科講師．平成 19 年神戸大学工学部工学研究科情報知能学専攻特命助教．平成 22 年東京工科大学コンピュータサイエンス学部助教．平成 23 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻特任准教授．博士（工学）．ソフトウェア工学教育，サービス指向アーキテクチャ，ホームネットワークシステム，Web サービス，ソフトウェアプロセス等の研究に従事．IEEE，ACM，IEICE 各会員．



楠本 真二 （正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学科助手．平成 8 年同学科講師．平成 11 年同学科助教授．平成 14 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻助教授．平成 17 年同専攻教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価，プロジェクト管理に関する研究に従事．IEEE，JFPUG 各会員．

© 2013 Information Processing Society of Japan