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

信学技報
TECHNICAL REPORT OF IEICE.

ソフトウェア開発における コピーアンドペーストによって生じたコード片に対する調査

大田　崇史† 　井垣　　宏† 　堀田　圭祐† 　肥後　芳樹† 　楠本　真二†

† 大阪大学大学院情報科学研究科　〒 565-0871 大阪府吹田市山田丘 1-5
E-mail: †{t-ohta,igaki,k-hotta,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　ソフトウェア開発における生産性の向上等を目的として，コピーアンドペーストを伴うソースコードの再利用が行われている．コピーアンドペーストによるプログラミングには，生産性の向上といったメリットの他に，コピー元に含まれるバグがコピー先に伝搬してしまうといった問題点があることも知られている．そのため，ソースコード中の類似したコード片（コードクローン）をコードクローン検出ツールを用いて特定し，修正の必要なコード片の特定やリファクタリング候補の特定といった支援が行われている．一方で，コードクローン検出ツールはツールによる自動生成コードやテストケースといったコピーアンドペーストを伴わないコードクローンも数多く検出することで知られている．そこで本研究では，開発者によるコピーアンドペーストとコードクローン検出結果の間にどの程度乖離があるか，またその差異が具体的にどのようなものであるかを実際のソフトウェア開発プロジェクトを対象として分析した．結果として，既存の検出ツールはコピーアンドペーストの 74.6% を検出できず，CDSW の検出結果の 95.7%がコピーアンドペーストを伴わないコードクローンであった．
キーワード　コードクローン，コピーアンドペースト，コードクローン検出ツール

An Empirical Study on Copy and Paste of Code in Software Development.

Takafumi OHTA†, Hiroshi IGAKI†, Keisuke HOTTA†, Yoshiki HIGO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University　Yamadaoka 1-5, Suita-Shi, Osaka, 565-0871 Japan
E-mail: †{t-ohta,igaki,k-hotta,higo,kusumoto}@ist.osaka-u.ac.jp

Abstract　In order to improve productivity etc. copy and paste programming is performed. It is known that not only source code but bugs are also copied between file in copy and paste programming. Therefore, code clone detection tools which extract the identical or similar code fragments (called code clone) in a source code are proposed. On the other hand, it is also known that code clone detection tools detect lots of code clones which include the source code which is not generated by copy and paste. In this paper, we analyzed the difference between the code fragments generated by copy and paste, and the code clone detection result, through actual software development project. As a result, a conventional code clone detection tool did not detect 74.6% of the code fragments generated by copy and paste, and 95.7% of the result detected by the tool are not generated by copy and paste.
Key words　Code Clone, Copy and Paste, Code Clone Detection Tool

1. ま え が き

ソフトウェア開発において，ソースコードの再利用は開発効率や信頼性の向上に有用であると言われている．特に，ソースコードが公開されているオープンソース・ソフトウェアやソースコード検索システムの普及に伴い，開発者が既存のコード片をコピーし，開発対象に貼り付ける，コピーアンドペーストによるプログラミングがよく行われるようになってきている [1]．

一方で，ソースコードのコピーアンドペーストによる再利用には問題も存在する．ソースコードのコピーアンドペーストを行う際，コピー元のコード片がバグを含むと，コピー先のコード片もバグを含むことになる．そのため，バグ修正ではコピー

元のコード片に加えコピー先のコード片の修正が必要になる．バグを含むコード片が複数箇所にコピーされた場合，コピー元のコード片と全てのコピー先のコード片についてバグ修正を行う必要がある．しかし，全てのコピー元とコピー先を把握することは困難であるため，コピーアンドペーストは保守作業を困難にする要因の一つとされている [2]．

この種のコピーアンドペーストによる再利用を特定するために，コードクローン検出ツールが良く利用される [3] [4]．コードクローンとはソースコード中に存在する互いに類似または一致したコード片のことであり，コピーアンドペーストや同一処理の繰り返し記述で発生する．コードクローン検出ツールを利用することで，コードクローンを検出し，修正の必要なコード片の特定やリファクタリング候補の特定 [5] といった様々な支援が可能となる．

一方で，コードクローン検出ツールによって検出されるコードクローンには，ツールによる自動生成コード等の開発者によるコピーアンドペーストを伴わない類似コード片も数多く含まれる．しかしながら，開発者によるコピーアンドペーストを伴うコードクローンとそれ以外の理由によって類似しているコードクローンの違いを分析した既存研究は存在しない．

そこで本研究では，複数人による小規模ソフトウェア開発プロジェクトを対象とし，開発者によるすべてのコピーアンドペースト行動の特定を行った．さらに，その結果とコードクローン検出ツール「CDSW [6]」による結果を比較し，ツールによる結果と開発者によるコピーアンドペーストの違いを分析した．その結果，CDSW はコピーアンドペーストの 74.6% を検出できず，CDSW の検出結果の 95.7%がコピーアンドペーストを伴わないコードクローンであった．

2. コードクローン検出技術

本節ではコードクローンの分類を説明し，コードクローンの検出ツールを紹介する．また，本稿におけるコピーアンドペーストを定義し，コードクローン検出ツールにおける課題を示す．

2.1 コードクローンの分類

コードクローンはその類似性に基づき 3 つのタイプに分類できる [7] [8]．

Type-1　空白やタブの有無，括弧の位置などのコーディングスタイルに依存する箇所を除いて，完全に一致するコードクローン．

Type-2　変数名や関数名などのユーザ定義名，また変数の型など一部の予約語のみが異なるコードクローン．

Type-3　Type-2 における変更に加えて，文の挿入や削除，変更が行われたコードクローン．なお，類似するコード片間の非共通部分のことをギャップと呼ぶ．

タイプごとに対応するコードクローン検出ツールが存在する．コピーアンドペーストにおいては，コピーアンドペースト後に，コピー先の関数名や変数名などにあわせてペーストを行ったコードの変更が行われることが多い．そのため，コピーアンドペーストの特定においては，Type-2 以上，文の挿入等も考慮する場合は，Type-3 のコードクローンが検出できることが望ましい．

Type-3 のコードクローンを検出する手法として，抽象構文木を用いた検出手法，プログラム依存グラフを用いた検出手法，関数メトリクスを用いた検出手法，Smith-Waterman アルゴリズム [14] を用いた検出手法 [6] などが存在する．

ソースコードを対象としたコピーアンドペーストとは，あるソースファイルの一部あるいは全部のコード片を開発者が一時的にコピーし，コピー元のソースファイルとは別のファイルもしくは同一ファイルの別箇所に貼り付けることを指す．本稿では特に，コピー元及びコピー先のファイルが同一プロジェクトに含まれるようなコピーアンドペーストを対象とする．すなわち，異なるプロジェクトや Web 上に存在するソースコードがコピー元となる場合を対象としない．コピーアンドペーストによるソースコードの再利用を特定する目的で，コードクローン検出ツールが使われている [10]．

2.2 コードクローン検出の課題

コードクローン検出ツールによって検出されるコード片には，コピーアンドペーストによらないものが多く含まれる．例えば，ツールによる自動生成コードやテストケース，単純な Java オブジェクト等において，開発者がコピーアンドペーストしたわけでは無いにもかかわらず，コード片が類似しているために，コードクローンであると判断されることがある．

このようなツールによる自動生成コードやテストケースに起因するコードクローンは，コードクローン検出結果に悪影響を及ぼすことがあると考えられている．そのため，Goede ら [9] は，コードクローンに対する開発者の修正頻度やリスクを分析する際に，ツールによる自動生成コードをコメントから類推し，コードクローン検出対象から排除している．同様に，石原 [11] ら，Harder [12] らは自動生成コードを，[13] らはテストケースをコードクローン検出の対象から排除している．

一方で，コピーアンドペーストとコードクローン検出結果の間にどの程度乖離があるかは必ずしも分析されていない．そこで本研究では，コピーアンドペーストとコードクローン検出結果の差異がどの程度あり，その差異が何に起因するものであるかをケーススタディにもとづいて明らかにする．

3. ソフトウェア開発プロジェクトにおけるコピーアンドペーストの抽出

本節では，コードクローン検出ツールの検出結果とコピーアンドペーストを比較するために，実際のコピーアンドペーストを特定する仕組みを提案する．

3.1 キーアイデア

コピーアンドペーストを行う際，開発者は対象となるソースファイルからコード片を選択し，コピーアンドペーストを行うコード片をクリップボードに保存する．その後，コピー先ソースファイルを開き，貼り付けたい箇所を選択し，クリップボードに保存されたコード片を貼り付ける．そこで我々はクリップボードに着目し，コピーアンドペーストを特定する．コピーアンドペースト特定に利用する各種ログを以下に示す．

クリップボードログ　クリップボードに文書が保存された時刻

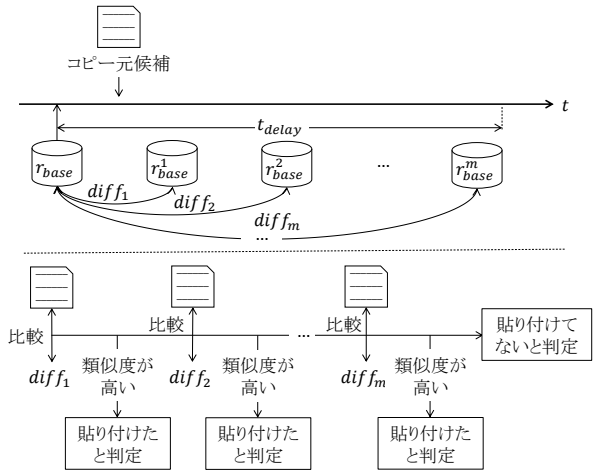

図 1　貼り付けられたかの判定

(コピーアンドペースト時刻)，保存した開発者及び文書 (コピーアンドペーストコード) の内容

アプリケーション実行履歴　どの開発者がどのアプリケーションを用いてどのファイルをいつからいつまで開いていたか

ソースファイル編集履歴　開発者ごとに，どのソースファイルがどのように編集されたかを保持する．ソースファイルは一定時刻あるいは開発者がファイルを保存したタイミングごとに版管理されており，特定のリビジョンが保存された時刻とファイル内容が保持されている．

クリップボードログに保存されたすべての文書を対象として，下記 Step にもとづいてコピーアンドペーストの特定を行う．

Step1:　アプリケーション実行履歴及びソースファイル編集履歴より，コピーアンドペースト元がどこであるかを特定する．

Step2:　Step 1 の結果とソースファイル編集履歴より，コピーアンドペースト先がどこであるかを特定する．

以降では，各 Step の詳細について述べる．

3.2　コピーアンドペーストの特定

クリップボードログには，開発者，時刻，コピーアンドペーストされたコード片が保存されている．すべてのコピーアンドペーストされたコード片を対象として以降の Step1,2 を行う．

Step1:コピーアンドペースト元の特定

コピーアンドペーストされた各コード片について，コピーアンドペースト元を特定する．ここでは，開発者がクリップボードに文書をコピーした時刻を利用し，その時刻に対象の開発者がどのアプリケーションを用いてどのファイルを開いていたかをアプリケーション実行履歴より特定する．ここでは，開発者が開いていたファイルがコピーアンドペーストの特定を目指すプロジェクト内のファイルであるかを調査する．Web ページや別のプロジェクトのファイルが開かれていた場合は，コピーアンドペーストコードを廃棄し，終了する．

対象プロジェクト内のファイルを開いていたことが確認された場合，コピーアンドペーストされた時刻時点での対象ファイルの内容をソースファイル編集履歴から取得し，コピーアンドペーストコードがどのファイルからコピーされたものであるかを特定する．

Step2:コピーアンドペースト先の特定

Step2 では，図 1 に示すようにコピーアンドペーストコードとソースファイル編集履歴を用いて，どのファイルに対してコピーアンドペーストが行われたか（コピーアンドペースト先）の特定を行う．

この Step では，まずコピーアンドペースト時刻より前のリビジョンのうち，最新のリビジョン r_{base} を抽出する．次に，r_{base} から一定時間 t $_{delay}$ 以内に記録されたリビジョン r^i_{base}（0 < i < m）についてリビジョンが古いものから順に r_{base} との差分 $diff_i$ を抽出する．抽出した差分とコピーアンドペーストコード間の類似度を比較することで，対象の差分を含むファイルがコピー先であるかを判定する．ここで，コピーアンドペーストコード $candidate$ と $diff_i$ の類似度 $sim(candidate, diff)$ を以下の式 1 で定義する．

$$sim(candidate, diff_i) = \frac{levenshtein(candidate, diff)}{length(candidate)} \quad (1)$$

$levenshtein$ は 2 つの文字列のレーベンシュタイン距離を表し，$length$ は文字列の長さを表す．つまり，式（1）は，コピーアンドペーストコードの文字列の何割を編集すればリビジョン間の差分と一致するかを表しており，小さければ小さいほどコピーアンドペーストコードを貼り付けた確率が高いということを表す．ここではある閾値 T_h を用いて，$sim(candidate, diff_i) < T_h$ のときコピーアンドペーストコードが貼り付けられた．つまり，コピーアンドペーストが行われたと判定する．本稿では T_h を 0.3 とした．なお，閾値 T_h を 0 としないのは，ソースファイル編集履歴に，コピーアンドペーストコードが貼り付けられた直後のソースファイルが必ずしも保存されているとは限らないためである．あるリビジョン r^i_{base} と r_{base} の差分からコピーアンドペーストコードが貼り付けられたと判定された場合，判定処理を打ち切る．つまり，r^{i+1}_{base} 以降のリビジョンについては，差分の抽出及び差分とコピー元候補の抽出を行わない．

以上の Step により，クリップボードログに保存されたすべての文書からコピーアンドペーストを特定する．

4.　ケーススタディ

本節では，大学院生向け講義 Cloud Spiral の講義中に開発を行ったプロジェクトについてコピーアンドペーストの抽出及び，コードクローンの検出を行った結果を述べる．

4.1　Cloud Spiral

Cloud Spiral とは大学院生向けの講義で 2013 年度は西日本の 9 大学から 49 人が受講した．Cloud Spiral では，5 又は 6 人で構成される班を 9 班作り，開発を行った．開発対象は全チーム共通で，2013 年度はチケットの購入を行う web アプリケーションを作成した．開発期間は連続する 4 日間の 10:30〜17:00 で，各プロジェクトの総行数は平均して約 5000 LOC であった．開発は統合開発環境 Eclipse 上で行い，版管理システムとして SVN を用いた．本稿では，ある 1 つの 6 人班に対して行ったコピーアンドペーストの抽出及び，コードクローンの検出結果について述べる．この班の開発したプロジェクトは

5028 LOC であった．

4.2 Cloud Spiral で収集したログ

本小節では，Cloud Spiral で収集したログについて述べる．Cloud Spiral では前節で述べたクリップボード，アプリケーションの実行履歴及びソースファイルの編集履歴を収集した．ソースコードの編集履歴の保存は Dropbox を用いて行った．Cloud Spiral のプロジェクトは SVN で管理していたが，それとは別にプロジェクト開発履歴収集のために Dropbox を用いた．本プロジェクトでは，すべての受講生は eclipse を利用しており，eclipse プラグインとして，編集中のソースファイルを 1 分ごとに保存する機能を追加した．結果として，すべてのソースファイルは Dropbox 上に 1 分毎に保存されることになった．

本稿の実験対象とする班について，1130 件のクリップボードのログ，11879 件のアプリケーション実行履歴及び 360967 リビジョンのソースファイル編集履歴を収集した．

4.3 抽出したコピーアンドペースト

前小節で示したログを元にコピーアンドペーストの抽出を行った．コピーアンドペーストの特定結果を表 1 に示す．1130 件のクリップボードログの内，288 件の文書が対象プロジェクト外から複製されたものであった．プロジェクト内からクリップボードに保存された文書 842 件の内，実際にコピーアンドペーストが行われたと特定されたものは 264 件であった．さらに，その中から java ファイルにおけるコピーアンドペーストの詳細を表 2 に示す．java ファイルにおけるコピーアンドペーストは 171 件で，コメントに関するコピーアンドペーストが 67 件，ソースコードのコピーアンドペーストが 104 件であった．コメントは今回利用したコードクローン検出ツールで検出されないため，比較対象から除外した．ソースコードのコピーアンドペーストの内 47 件はコピー元又はコピー先が開発の過程で消失し，55 件は最終成果物に存在した．ここで消失したかどうかの判定については，下記の Step で行った．

Step1: コピーアンドペーストされたコピー元及びコピー先のコピーアンドペースト時点のソースコードと SVN 上に保存された最終成果物の同一ファイルとを目視で比較を行い，最終成果物におけるコピーアンドペーストコードの開始行・終了行をコピー元ファイル，コピー先ファイルそれぞれについて特定する．

表 1 抽出したコピーアンドペーストの内訳

ファイルの種類	検出数［個］
java	171
html	34
txt	50
xml	9
合計	264

表 2 java ファイルについて抽出したコピーアンドペースト

	検出数［個］
開発過程で消失	49
最終成果物に存在	55
コメント	67
合計	171

```
IdentifyingEventForm form=new IdentifyingEventForm();
form.setEventId("abc");
boolean actual=form.validate();
```

コピー元

```
IdentifyingEventForm form=new IdentifyingEventForm();
form.setEventId("");
boolean actual=form.validate();
```

コピー先

図 2 コード片が小さく検出できないコピーアンドペースト

Step2: Step1 の結果にもとづいて，最終成果物における 2 つのコピーアンドペーストコードとコピーアンドペースト時にクリップボードログとして記録されたコピーアンドペーストコード間の類似度を式 1 と同様の方法で算出する．このとき，コピー元，コピー先いずれか一つでも類似度が閾値 0.5 を超えたときに，消失したと判定する．

コピーアンドペーストが存在する場合は，CDSW の結果と比較するためにコピー元及びコピー先の開始，終了行番号を記録する．

4.4 コードクローンの検出

コードクローン検出ツールとして CDSW [6] を用いて，プロジェクトの最終成果物についてコードクローンの検出を行った結果， 329 個のコードクローンが検出された．CDSW では最小クローン長と許容ギャップ数を変更できる．最小クローン長及び，許容ギャップ数はコードクローンとして認める最小のトークン数，許容するギャップの数を示す．本稿では，最小クローン長を 30 ，許容ギャップ数 3 とした．

5. コピーアンドペーストとコードクローンの比較

本節では，前節で収集したコピーアンドペーストとコードクローンについて比較を行った結果について述べる．最終成果物についてコードクローンの検出を行ったため，最終成果物に存在するコピーアンドペースト 55 個と，ツールによって検出されたコードクローン 329 個について比較を行った．CDSW はブロック単位でコードクローンの検出を行い，コードクローンをブロックの開始行，終了行及びファイルパスで表す．そこで本研究では同一のパスを持つコピー元ファイルとコピー先ファイルに含まれるコピーアンドペーストコードの開始行，終了行がツールによって検出されたコードクローンの開始行，終了行の間に含まれれば CDSW の検出結果とコピーアンドペーストが一致したものと判定する．CDSW の検出精度を測る評価指標として 適合率（*precision*） 及び 再現率（*recall*）を定義する．CP を最終成果物に存在するコピーアンドペーストの集合，*Clone* を検出したコードクローンの集合とする．このとき，

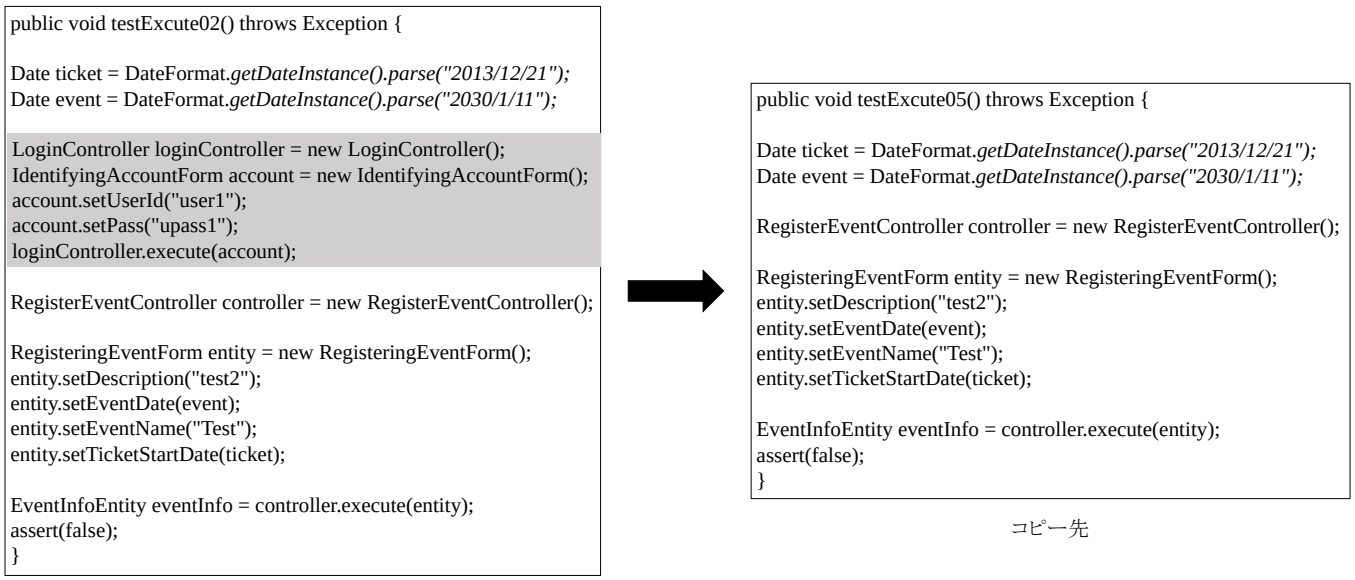

図 3　不一致部分が多く検出できないコピーアンドペースト

precision 及び *recall* を以下の式で定義する．

$$precision = \frac{|CP \cap Clone|}{|Clone|} \tag{2}$$

$$recall = \frac{|CP \cap Clone|}{|CP|} \tag{3}$$

最終成果物に存在するコピーアンドペーストと，検出したクローンについて比較を行った結果，$|CP \cap Clone| = 14$，$|CP| = 55$，$|Clone| = 329$ であった．よって，式 2，式 3 より $precision = 0.0425$，$recall = 0.2545$ となった．*precision* と *recall* が低い，つまり，ツールによるコードクローン検出結果とコピーアンドペーストは一致しないことが分かる．特に CDSW で検出したコードクローンのほとんどがコピーアンドペーストを含まないことが分かる．

5.1　検出漏れしたコピーアンドペーストの特徴

図 2 に CDSW で検出できなかったコピーアンドペーストの例を示す．図 2 の例では，コピーアンドペーストで作成したコード片が小さいこと，つまり，コピーしたコード片に含まれるトークン数が少ない事が検出漏れの原因だった．CDSW はコードクローンとして検出するものの最小のトークン数を設定できる．本実験では，最小トークン数を 30 として行ったが検出漏れとなったコピーアンドペーストの内，25 個のコピーアンドペーストのトークン数が 30 よりも少なかった．これらのコピーアンドペーストを除外して再度再現率を計算した結果 0.4242 となった．CDSW の設定を変更することでトークン数が少ないものについてもコードクローンとして検出することができるが，より小さなコード片もコードクローンとして検出するため，再現率が低くなる．

十分なトークン数を含むにも関わらず，CDSW で検出できなかったコピーアンドペーストを図 3 に示す．図 3 はコピー先においてコピー元の塗りつぶした部分が削除されている．その結果，不一致部分を多く含み CDSW の許容ギャップ数を越え，CDSW で検出できなくなった．許容ギャップ数を増やすことで不一致部分を多く含むコピーアンドペーストも検出できるが，誤検出率も高くなると考えられる．わずかではあるが，これらの例とは別に検出できないコピーアンドペーストとして複数のブロックにまたがるコードをコピーアンドペーストがあった．CDSW はブロック単位でのコードクローン検出を行うため，このようなコピーアンドペーストは検出できない．表 4 に検出漏れとなったコピーアンドペーストの内訳を示す．

5.2　コピーアンドペーストを伴わないコードクローンの特徴

CDSW が誤検出したコードクローンの例を図 4 に示す．図 4 ではコードクローンペアの両方がテストケースから検出されている．テストでは，あるメソッドを特定の引数で呼び出した結果が期待する動作をするか調べるというようにある程度処理

表 3　検出漏れとなったコピーアンドペーストの原因

原因	検出漏れ数［個］
トークン数が少ない	25
ギャップを含む	14
複数のブロックを含む	2

表 4　検出したコードクローンペアの分類

コードクローンペアの分類	検出数［個］
2 つのコード片をテスト対象から検出	11
テスト対象とテストケースから 1 つずつ検出	4
2 つのコード片をテストケースから検出	314
合計	329

表 5　抽出したコピーアンドペーストの内訳

	最終成果物に存在［個］	CDSW で検出［個］
テスト対象	10	3
テストケース	45	11
合計	55	14

```
public void testValidate02() throws Exception {
IdentifyingAccountForm idAccForm =
new IdentifyingAccountForm();
idAccForm.setUserId("0123456789AB");
idAccForm.setPass  ("abcdefg01234");
boolean actual = idAccForm.validate();
assertTrue(actual);
}
```

```
public void testRegisteringSeatCategoryForm1() throws Exception{
RegisteringSeatCategoryForm registeringSeatEntity =
new RegisteringSeatCategoryForm();
registeringSeatEntity.setCount(1);
registeringSeatEntity.setEventId("100");
registeringSeatEntity.setFee(0);
registeringSeatEntity.setSeatName("a");
boolean actual = registeringSeatEntity.validate();
assertTrue(actual);
}
```

図 4　コピーアンドペーストを伴わないコードクローン

の流れが決まっている．図 4 のコードクローンペアは両者とも，クラスのフィールド値を変化させ，そのフィールド値の検査が正しく行われるかをテストする．このように，テストケースは処理が似通う傾向にある．また，CDSW は変数名や関数名が異なるコードクローンを検出するために，ユーザ定義名をすべて特殊文字に置換する．例えば，a = b.getc(); という文と d = e.getf(); という文を同一とみなす．よって，処理が似通うコード片のペアをコードクローンとして検出する可能性がある．

テストケースが検出結果に与える影響を調べるために CDSW が検出したコードクローンペアを 1）2 つのコード片が共にテスト対象から検出されるもの．2）片方のコード片がテスト対象からもう一方が，テストケースから検出されるもの．3）2 つのコード片が共にテストケースから検出されるもの．に分類した結果，表 4 が示すように，ほぼ全てのコードクローンはテストケースから検出されていることがわかった．また，コピーアンドペーストではないと判断されたコードクローンの内 99 % がテストケースに関連するコードクローンであった．以上より，テストケースがコードクローン検出ツールの検出結果に与える影響は大きいと考えられる．

そこで，テストケースを除外した場合について CDSW の検出精度を調べた．表 5 はコピーアンドペーストにおける，テスト対象とテストケースの内訳を示す．テストケースを除外したときに $|CP \cap Clone| = 3$，$|CP| = 10$，$|Clone| = 11$ となり，CDSW の検出精度は $precision = 0.2727$，$recall = 0.3$ となった．

6. おわりに

本稿ではソフトウェア開発中で行われたコピーアンドペーストとコードクローン検出ツール CDSW の検出結果を比較した．比較に向けて，ソフトウェア開発中のログを利用してコピーアンドペーストを抽出する手法を提案した．比較の結果，CDSW がコピーアンドペーストを検出できない場合（検出漏れ）は 74.6 % で，CDSW がコピーアンドペースト以外を検出する場合（コピーアンドペーストを伴わないコードクローン）は 95.7% であった．検出漏れをしたもののほとんどが，コピーアンドペーストしたコード片のトークン数がコードクローンとして認める最小トークン数よりも少ない事が原因であった．また，（コピーアンドペーストを伴わないコードクローンの 99 % がテストケースに関するコードクローンであった．これは，テストの構造が類似することが原因だと考えられる．

今後は，より多くのデータを集めることでコピーアンドペーストに関する知見を得る予定である．本稿では，Cloud Spiral の開発における 1 つの班の開発データを用いて，コードクローンとコピーアンドペーストの比較を行った．別の班の開発データを用いた比較をすることで，より信頼度の高い結果を得られると考えられる．

文　献

[1] Zhang, G., Peng, X. and Zhao, Z. X. W.: Cloning Practices: Why Developers Clone and What can be Changed, Proceedings of the 28th International Conference on Software Maintenance, pp. 285-294 (2012).

[2] Zibran, M. F. and Roy, C. K.: The Road of Software Clone Management: A Survey, Technical Report, University of Saskatchewan (2012).

[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，電子情報通信学会論文誌，Vol. 91-D, No. 6, pp. 1465-1481 (2008).

[4] 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).

[5] 堀田圭佑, 肥後芳樹, 楠本真二, "プログラム依存グラフを用いたコードクローンに対するテンプレートメソッドパターン適用支援手法," 電子情報通信学会論文誌 D, volume J95-D, number 7, pages 1439-1453, 2012 年 7 月.

[6] Hiroaki Murakami, Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto, "Gapped Code Clone Detection with Lightweight Source Code Analysis," In Proc. of the 21st International Conference on Program Comprehension, pages 93-102, May 2013.

[7] S. Bellon. Detection of software clones. Technical Report, Institute for Soft- ware Technology, University of Stuttugart, 2003. available at http://www.bauhaus-stuttgart.de/clones/.

[8] S. Bellon, R. Koschke, G. Antniol, J. Krinke, and E. Merlo. Comparison and evalua- tion of clone detection tools. IEEE Trans. on Software Engineering, Vol. 31, No. 10, pp. 804-818, October 2007.

[9] N. Goede and R. Koschke, "Frequency and Risks of Changes to Clones", ICSE 2011.

[10] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, M. Irlbeck. On the Extent and Nature of Software Reuse in Open Source Java Projects. Top Productivity through Software Reuse, pp. 207222, 2011.

[11] 石原 知也, 堀田 圭佑, 肥後 芳樹, 井垣 宏, 楠本 真二 , "大規模なソフトウェア群を対象とするメソッド単位でのコードクローン検出," 情報処理学会論文誌, volume 54, number 2, pages 835-844, 2013 年 2 月.

[12] J. Harder and N. Goede, "Cloned code: stable code", Journal of Software Evolution and Process 2013.

[13] L. Barbour, F. Khomh, and Y. Zou, "Late Propagation in Software Clones", ICSM 2011.

[14] T. F. Smith and M. S. Waterman.: Identication of common molecular subsequences. Journal of Molecular Biology, Vol. 147, No. 1, pp. 195-197, Mar. 1981.

[15] "Eclipse". http://www.eclipse.org.