論　文

版管理システムを用いたクローン履歴分析手法の提案

川口　真司[†a)]　　松下　誠[††]　　井上　克郎[††]

Clone History Analysis Using Configuration Management System

Shinji KAWAGUCHI[†a)], Makoto MATSUSHITA[††], and Katsuro INOUE[††]

あらまし　ソフトウェアの保守工程における大きな問題の一つとしてクローンが指摘されている．コピーされて生成されたクローンは多少の編集が施されることが多く，そのような場合でも適切にクローンが検出できるようにするため様々なクローン検出手法が提案されている．これらの手法は最新時のソースコードに対して分析を行うが，より施された編集が少ない時点に遡ってクローン分析を適用することで，最新時の分析だけでは得られないクローンを得ることができる．このような過去にクローン関係にあったコードを抽出するためには過去のソースコードにおけるクローン解析結果に加えて，過去に存在したクローンが現在のソースコードのどこに対応するのかという情報，すなわちクローン履歴が必要である．本論文では，版管理システムに蓄積されたソースコードを対象としてクローンの履歴を抽出する手法を提案する．本手法ではまずクローン分析を過去の時点に遡って順次適用し，各時点間のクローンについてクローン履歴を抽出する．また，PostgreSQL に対して提案手法を適用し，抽出できるクローンの有用性について考察を行う．

キーワード　クローン，履歴，ソフトウェアリポジトリ

1. まえがき

近年，ソフトウェアの大規模化にともないソフトウェアの開発工程において保守工程の占める割合は年々増加の一途を辿っている．保守工程におけるさまざまな問題のなかでも非常に大きな問題の一つとして，ソースコード中に含まれる重複コード (以下，クローン) が挙げられる [1]．もしクローンの一つに不具合が見つかった場合，すべてのクローンを調査し，それぞれに対して同様の修正を施すことになる可能性が高い．この作業はソフトウェアが大規模であればあるほど困難となる．

この問題に対処するべく，これまでにクローンを検出するための様々な手法が提案されており，そのいくつかは実際に利用可能なシステムとして実用化されている [2]．このようなクローン抽出システムは，大規模なソフトウェアから自動的にクローンを発見することで，クローン解消作業の負担を減らすために利用されている．

しかし，実際には関連性が高いにも関わらず，最新のバージョンを分析するだけではクローンとして抽出できないコード片が存在する．図 1 にそのようなコード片の例を示す．図 1 は時刻 $t-1, t$ におけるソースコードに含まれるクローンを表している．時刻 $t-1$ において二つのクローンセット (互いにクローン関係にあるクローンの集合) が存在している．そして，時刻 t においてクローンセット B のうち一部のクローンに編集が加えられた結果，二つのクローンセット B', C' に分かれている．

このとき，もしクローンセット B' に不具合が見つかった場合には，クローンセット B' だけではなく，若干の編集を加えられたクローンセット C' もまた見過ごせないコピーの一部である (以後，クローンセット C' をクローンセット B' の分岐クローンセットと呼ぶ．逆も同様)．しかし，これまでに提案されているクローン分析では現時点での情報のみを用いて分析を行うため，時刻 t においてはクローンセット B' にとってクローンセット C' はクローンセット A' と同じく無関係なクローンセットでしかない．

このようなクローンセット同士の関係を抽出するに

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama-cho, Ikoma-shi, 630-0101 Japan
†† 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, 1-3 Machikaneyama-cho, Toyonaka-shi, 560-8531 Japan
a) E-mail: s-kawagt@is.naist.jp

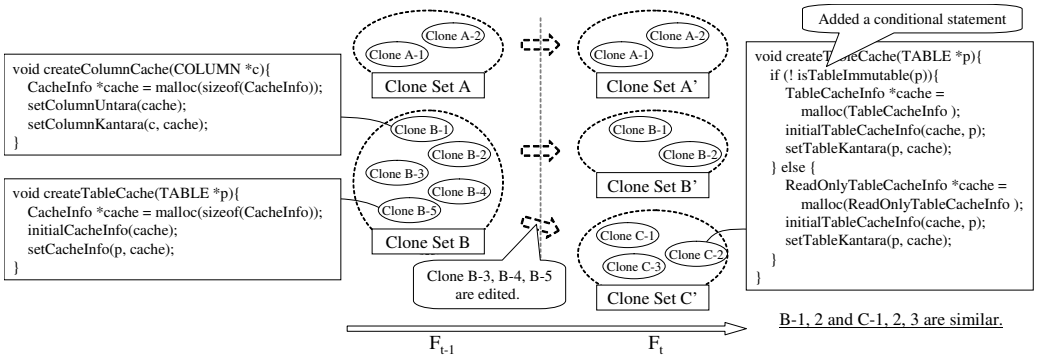

図 1　クローンの変更履歴
Fig. 1　Clone Evolution

は，過去のソースコードにおけるクローン解析結果に加えて，過去に存在したクローンが現在のソースコードのどこかに対応しているのか，対応しているとすればそれがどこなのかという情報が必要である．

そこで，本研究では過去と現在のクローン間の繋がりを分析するための手法として，クローン履歴分析を提案する．クローン履歴分析では，現時点のクローンが過去のバージョンのどのクローンに対応するのか，今あるクローンがどのような変遷を辿ってきたのかを特定する．

2. 前提とするシステム

2.1 クローン分析システム

クローン分析手法には大きくわけてソースコードの字句解析に基づく手法 [1,3,4] と，特徴メトリクスに基づく手法 [5,6] に分けられる．ソースコードの字句解析に基づく手法 では，ソースコード中で同一の文字列を検索することでクローンの検出を行う．特徴メトリクスに基づく手法では，例えばクラスや関数，ファイルのようなプログラム中のある種の単位ごとに特徴メトリクスを定義・算出し，それらのメトリクス値が類似したものをクローンとして抽出する手法である．一般には字句解析に基づく手法のほうがコストが増えるが，より細粒度なクローンを抽出できる．

本研究では，字句解析ベースの検出ツール CCFinder [1] を利用してクローン履歴の分析を行う．CCFinder は高いスケーラビリティを有しており，大規模なソフトウェアに対しても実用的な時間でクローンの抽出を行える．また，実際にさまざまな大規模ソフトウェアへ適用され，その有用性が確認されている [7].

クローン分析を行う際，CCFinder は空白や改行，コメント等を除去するとともに，入力テキスト中の変数名や関数名等を同一記号に縮退させる．その後に，しきい値以上の長さの共通字句列を探索し，全ての対のリストを出力する．コピー& ペーストによってクローンが作られた場合，変数名などが変更されないことは稀であり，何らかの形で変数名などがコピー先のコンテキストに合致するよう書きかえられることが多い．CCFinder では変数名，関数名を同一記号とみなすことによって適切にコピーを検出できる．また，長大な共通字句列が存在したときに，その部分字句列が併せて出力されるのを防ぐために，それぞれ互いに包含関係でないもののみをクローンとして出力する．

2.2 版管理システム

版管理システムとは CVS [8] や Subversion [9] のようにプロダクトの保管・管理に用いられるシステムである．版管理システムでは，管理下のプロダクトを任意の時点の状態に復元して取得する機能が提供されている．

冒頭でとりあげた分岐クローンセットを取得するためには，過去のソースコードにおけるクローン解析結果が必要であることはすでに述べた．このような情報を得るためには，版管理システムを用いて過去のソースコードを取得し，そのソースコードに対してクローン分析手法を適用すればよい．

3. 諸 定 義

本節では分析対象となるクローンについて定義を与える．その上で異なる時点におけるコード片の対応関係を表す写像の概要と，クローン履歴関係の定義について述べる．

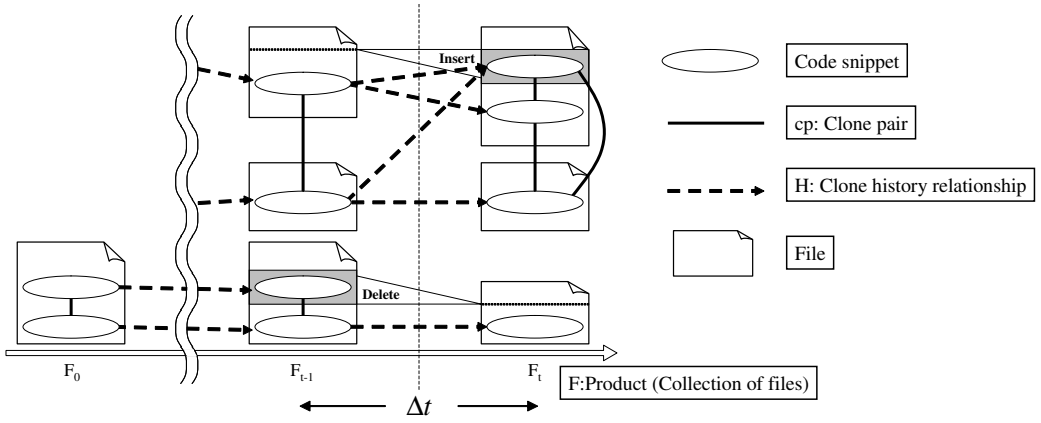

図 2　クローンとクローン履歴関係
Fig. 2　Clone and Clone History Relationship

3.1　クローン

クローン履歴分析のために定義したクローン履歴モデルを図 2 に示す．本モデルでは時間と共に変更されるソースコードを Δt ごとに区切って考える．分析の対象を版管理システムの管理下にあるソースコードファイルの集合とし，ある時刻 t における集合を**プロダクト (Product)** F_t と呼ぶ．

いま，各ファイルを文字列と考え，その連続する部分列を**コード片 (Code snippet)** と呼ぶ．そして，F_t に含まれるコード片のうち，共通字句列となるコード片の対 (a, b) を F_t に関する**クローンペア (Clone pair)**，a や b をそれぞれ F_t に関する **クローン (Clone)** と呼ぶ．このとき a と b は**クローン関係 (Clone relationship)** にあるという．また，クローン関係を同値関係と考え，その同値類を**クローンセット (Clone set)** と呼ぶ．

3.2　コード片の写像

次に時刻 t より Δt だけ過去の時点におけるプロダクト F_{t-1} に含まれるコード片に対して，F_t に含まれるコード片への写像を定義する．例えば F_{t-1} にあるコード片がもし編集されていなければ，F_t にも同じ内容のコード片が必ず存在する．また，時刻 t において編集されたテキストを含むコード片についても，図 4 のように編集操作を考慮して近似的に対応関係にあるコード片候補を求められる．このような両者の関係を写像として定義する．本写像によって F_{t-1} のコード片 r が空文字列でない F_t のコード片 s に写像される時，r を s の**親**，s を r の**子**とする．なお写像の詳細な定義は 4.3 節で述べる．

3.3　クローン履歴関係

F_{t-1} のコード片 a と F_t のコード片 b が以下のいずれかを満たす場合，a, b 間には **クローン履歴関係 (Clone history relationship)** があるといい，クローン履歴関係が成りたつコード片の組 (a, b) の集合を H_t と表す (図 3)．H_t は以下に述べる HC_t, HT_t, HA_t の和集合として定義される．

（1） HC_t: (a, a') が F_{t-1} に関するクローンペア，(b, b') が F_t に関するクローンペアで，b が a の，b' が a' の子となる a', b' が存在する．

（2） HA_t: a は F_{t-1} に関するクローン，b は F_t に関するクローンで，b と 1 行以上重複する位置にある $b' \in Change_{\Delta t}$ について (a, b') が $\{F_{t-1} \cup Change_{\Delta t}\}$ についてのクローンペアである．ただし，$Change_{\Delta t}$ は F_t のコード片の集合のうち F_{t-1} からの変更時に追加，編集された部分，$\{F_{t-1} \cup Change_{\Delta t}\}$ は F_{t-1} 全体に $Change_{\Delta t}$ のコード片を追加した部分とする．

（3） HT_t: a は F_{t-1} に関するクローン，もしくは F_{t-1} のコード片で $\exists x, (x, a) \in HT_{t-1}$．$b$ は a の子で a と b のテキスト類似度が高い (テキスト類似度の詳細は後述)．

HC_t は F_{t-1}, F_t の両方に存在する同一のクローンを表す．本定義は F_{t-1} でクローンペアを構成するクローン a, a' が，写像先の b, b' においてもクローンペアを構成している，ということを表す．

HA_t は F_t において追加されたクローンの履歴を表す．このようなコード片は，既にあるクローンセット

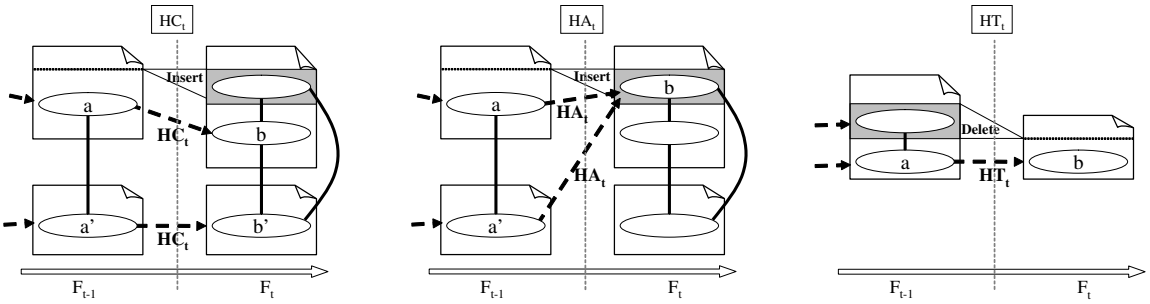

図 3　3 種類の履歴関係
Fig. 3　Three Types of Clone History Relationship

のどれとも等しくクローン履歴関係があると考える．これは，既存クローンのうちどれか一つを原本と決めることは難しく，またそうすることの意義も薄いからである．

このように本手法では HA_t を HC_t と分けて定義している．HC_t では親子関係が明示的に一意に定まるのに対して，HA_t では親が属するクローンセット内のクローンすべてとクローン履歴関係が存在することになる．そのため，HA_t と HC_t を分けることでクローン履歴関係がより正確なものとなる．

最後の HT_t は，かつてクローンを構成していたが最新版ではクローンペアであったコード片が削除，または編集されて別のクローンセットに属するようになったためにクローンでなくなった，いわば元クローンというべきコード片についてクローン履歴対応関係を抽出するための定義である．

ただし，クローンペアに一度でもなった部分全てを追いつづけるのは非効率である．また，何らかの変更が加わった場合には，その時点で過去のコードとの関連は希薄になる．そこで，テキストの変更度合いを表す尺度としてテキスト類似度を定義する．このテキスト類似度はコード片中に占める同一テキストの割合を表す．そして，テキストがほとんど変更されていない場合にのみクローン履歴対応関係があるものとする．

4. クローン履歴関係抽出手法

4.1 概　　要

以下にクローン履歴対応関係抽出手法の手順を示す．なお，クローン履歴解析を適用する期間の始まりを時刻 0，終わりを時刻 T で表し，C_t を F_t に関するクローンの集合，CT_t を HT_t を満たすコード片のうち F_t に属するコード片の集合とする．

（0） $H_0 = \emptyset, CT_0 = \emptyset$.

（1） $t = 0$ の F_0 を版管理システムから取得し，クローン解析手法を用いて C_0 を求める．

（2） for $t = 1, 2, ..., T$

(2-a) 版管理システムから F_t を取得し，クローン解析手法を用いて C_t を求める．

(2-b) F_{t-1} に関する全てのクローンとそれぞれの子について，HC_t を求める．

(2-c) $\{F_{t-1} \cup Change_{\Delta t}\}$ に対して CCFinder を適用して，HA_t を求める．

(2-d) C_{t-1} の残りと CT_{t-1} について HT_t を求める．

このように，本手法では解析を行う期間 $[0 \cdots T]$ を Δt ごとに区切って考え，クローン履歴対応関係 $H_1, H_2, ..., H_T$ を順次，抽出する．

以下に各手順の詳細，および HC_t, HT_t の抽出時に利用する親子関係を定義するための写像について述べる．

4.2 各手順の詳細

本節では F_t の解析における各手順について，その詳細を述べる．なお，前節のアルゴリズムにより，時刻 t についての解析をはじめる時点で $t-1$ までの各情報 $(F_{t-1}, C_{t-1}, H_{t-1}, CT_{t-1})$ は既知である．

(2-a) 版管理システムから F_t を取得し，クローン解析手法を用いて C_t を求める

入力: (版管理システム)

出力: F_t, C_t

まず当該時刻のプロダクト F_t を版管理システムから取得し，クローン解析を適用して C_t を得る．本研究においてはクローン解析手法として CCFinder を用い，その解析結果を C_t とする．

(2-b) F_{t-1} に関する全てのクローンについて HC_t を求める

入力: $F_{t-1}, F_t, C_{t-1}, C_t$

出力: HC_t

F_{t-1} に含まれる全てのクローンペアについて，それぞれのクローンの子が F_t に関するクローンペアとなっているかどうかを検証する．もしこの条件を満たしていれば，それぞれの親子ペアは HC_t に含まれるものとする．

(2-c) $\{F_{t-1} \cup Change_{\Delta t}\}$ に対して CCFinder を適用して，HA_t を求める

入力: $F_{t-1}, F_t, Change_{\Delta t}, C_{t-1}, C_t$

出力: HA_t

次に F_{t-1} と $Change_{\Delta t}$ の間にクローン解析を適用する．そして $a \in F_{t-1}, b' \in Change_{\Delta t}$ なるクローンペア (a, b') があれば，b' が指す位置にあるクローン $b \in C_t$ を考え (a, b) を HA_t に加える．

実際に適用する際には，編集された行とその前後 10 行を $Change_{\Delta t}$ とする．本研究では CCFinder が検出するクローンの最小トークン数を初期設定の 30 トークンで適用しているため，差分が 30 トークンよりも小さい場合にはクローンとして検出されなくなる．前後 10 行を含めることで，このような差分を含むクローンを適切に扱うことができる．

(2-d) C_{t-1} の残りと CT_{t-1} について HT_t を求める

入力: $F_{t-1}, F_t, C_{t-1}, HC_t, HA_t, CT_{t-1}$

出力: HT_t

最後に C_{t-1} に含まれるコード片のうち HC_t, HA_t 内のペアに含まれないコード片，および CT_{t-1} に含まれるコード片 a について，その子 b とのテキスト類似度を計測して HT_t の抽出を行う．テキスト類似度は以下の式で定義する．

$$TextSim(a, b) = \frac{2|a \cap b|}{|a| + |b|}.$$

ただし $|a| = a$ の行数，$|a \cap b|$ は GNU diff によって同一と判定された行数とする．そしてテキスト類似度が 0.7 以上のコード片 b が存在したとき，$(a, b) \in HT_t$ とする．なお，コード片 a に対して子は複数ありうるため，HT_t の条件を満たすコード片も複数存在し得る．このときには類似度が最大のコード片のみを HT_t の対象とする．

このように HT_t の判定条件は，ほぼ同一のテキストかどうかの判定となっている．もしクローン自身も何らかの変更があった場合には，この部分は $Change_{\Delta t}$ に含まれるため HA_t の解析対象となる．従って HT_t の解析を行う際には，そのようなケースを考慮する必要はない．

4.3 コード片の写像

本研究ではコード片 $a \in F_{t-1}$ からコード片 $b \in F_t$ への写像を，各ファイルの行番号の対応関係に基づいて定義する．CCFinder の出力には，コード片がどのファイルの何文字目から始まるか，という情報が含まれているため，行番号に基づく対応関係を適用することができる．

まず時刻 $t-1$ と時刻 t において，コード片 a が含まれるファイルに変更がなかった場合，a の写像先 b の開始行，終了行は a と全く同じとする．

次に，コード片 a が含まれるファイルに何らかの編集操作が行われていた場合を考える．もし a よりも前の部分に変更箇所があれば，その内容に応じて写像先 b の開始行，終了行を調整する．図 4 の Case 1 はコード片 a の前で編集操作が行われた場合の例である．このとき，a の前で行われた編集操作の全てを勘案して a の写像先 b の開始行，終了行は a のそれのそれぞれ 4 行後ろとする．また a に含まれる部分に変更が加えられていた場合には，その内容に応じて b の終了行を調整する．図 4 の Case 2 では a 内に 2 行新しい行が追加されているため，b の終了行は a のそれに対して 2 行追加した値とする．

最後に，クローン片の端の部分で書きかえ操作が発生した場合について述べる．図 4 の Case 3 ではコード片 a の開始行を跨がる形で書きかえされている．このような場合,「コード片 a の上に 2 行挿入があった」という解釈と,「コード片 a に 2 行の挿入がされた」という解釈，およびその中間 (1 行がコード片の上に，1 行はコード片 a 中に挿入された) が成りたつ．そこで，これら全てのケースをコード片 a の子とする．このように複数の子が生成されるケースは以下の 3 つである．

- 開始行，終了行が編集されている箇所を含む (図 4 Case 3)
- 開始行の一行前に挿入
- 終了行に挿入

本研究においては $F_{t-1} \rightarrow F_t$ への写像のみを扱うため，削除操作ではこのような複数候補を考える必要はない．

以上述べた 3 つの編集操作による影響をすべて足しあわせて写像先を決定する．

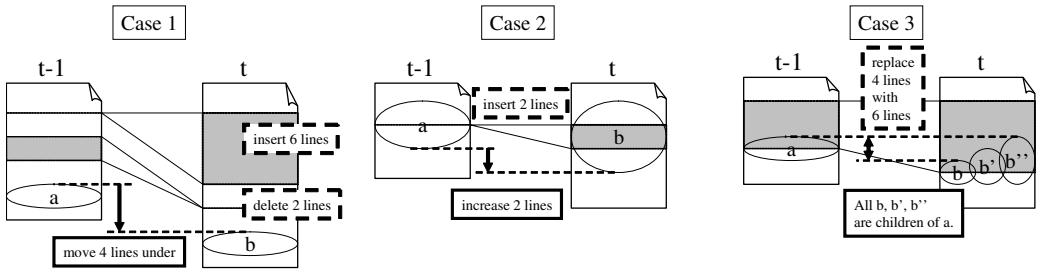

図 4　コード片の写像
Fig. 4　Mapping of Code Snippet

5. 実　　験

4. で述べた手法を用いることで図 1 におけるクローンセット B', C' 間のような分岐クローンセットを抽出することができる．本節では，PostgreSQL を対象として本手法を適用し，実際に分岐クローンセットをどの程度抽出できているかの検証を行う．

5.1 実 験 方 法

PostgreSQL リポジトリのなかから，ソースコードか格納されている src ディレクトリ以下の全てのファイルを対象とした．これらのファイルを 2005/01/01 〜2005/6/30 までの 6 ヶ月間について Δt を 1 週間として，計 24 時点についてクローン履歴関係の抽出を行った．

PostgreSQL を対象とした理由は，第一に提案手法で用いている CVS リポジトリが公開されていること，第二に PostgreSQL が非常に広く活用されており，開発も活発に行われているからである．

そして各バージョンに含まれるクローン c_t，および c_t とクローン履歴対応関係にある c_{t-1} について，c_t が属するクローンセットと c_{t-1} が属するクローンセットを考え，図 1 における Clone Set B と Clone Set B' のようにクローンセット中のクローン数が減少している箇所を " クローン減少箇所" として抽出した．

次に，それぞれのクローン減少箇所において，そこに含まれるクローンセットが分岐クローンセットであるかどうかの判定を人手により行い，もし分岐クローンセットが存在すればクローン履歴関係を正しく抽出できているかどうか検証した．

このように本実験ではクローンセット内のクローン数の減少が確認できたクローンセットのみに着目した．これは，実際に開発されているソフトウェアには数百から数万もの大量のコードクローンが存在し，さらに CCFinder では検出できない潜在的に存在するコードクローンもかなりの数に上るため，全てのクローンセットを検証することは現実的ではないからである．

5.2 実 験 結 果

Branched clone set	4
Branched clone set (miss)	0
Deleted clone set	5
Useless clone set	15

表 1　クローン減少箇所の調査結果
Table 1　Clone Decreasing Places

実験手法を適用した結果，クローン減少箇所は計 24 箇所あった (表 1)．そのうち 4 つのクローンセットに分岐クローンセットが存在し (Branched clone set)，それらのクローン履歴関係が適切に抽出できていることを確認した．そして，実際には分岐クローンセットであるにも関わらず，分岐クローンセット内のクローンについてクローン履歴関係が適切に抽出できていなかったケース (Branched clonse set (miss)) は存在しなかった．また，5 箇所がクローンセット内のクローンのいくつかが削除されたもの (Deleted clone set)，残り 15 箇所はそもそもクローンとして有用でないものであった (Useless clone set).

有用でないクローンとは，printf 文が連続している部分や数十，数百の定数定義が続いている部分などのように，単純な構文が 10 行以上連続している部分を指す．CCFinder は構文木として同形であればクローンとして検出を行うため，このような単純な構文が繰りかえされている箇所もクローンとして検出される．しかし，このような部分はそもそもクローンとして検出する必要がなく，実際にユーザにとっても中身を検討する価値がない．したがって，本実験においてはそのような箇所を評価から除外している．

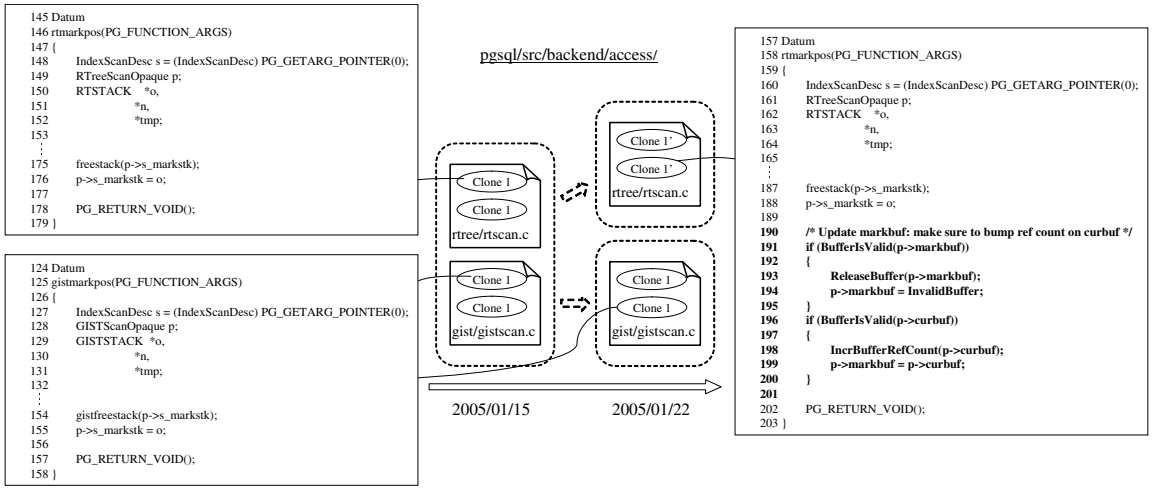

図 5　過去にクローンペアであったコード片の例
Fig. 5　Example of code snippets used to be clone pair

図 5 に分岐クローンセットとなったクローン減少箇所の例を示す．2005 年 1 月 15 日の時点では 4 つのコード片がクローンであった．一週間後，そのうちの 2 つに編集が加えられ，それぞれ独立したクローンセット Clone 1 と Clone 1’ になった．図 5 右側は新しい Clone 1’ の抜粋である．このように Clone1’ は太字で示した処理が追加されただけで，本質的には Clone1 同様のことを行っており，これらのコード片の間には依然として強い類似性がある．本手法により過去の履歴を辿ることで，これらのコード片の関係を抽出することができる．

6. 関 連 研 究

クローンの履歴を調査する研究としては，Kim らの研究 [10] が挙げられる．Kim らは我々の手法と同様に CCFinder を用いてクローンの履歴抽出を試みており，クローンセットの生存期間に着目して生存期間の分布がどのようになっているか，生存期間の違いによってクローンにどのような特徴があるかを調査している．Kim らの研究では，履歴抽出の対象となっているのはクローンセットであり，個々のクローンについての詳細な履歴は対象としていない．それに対して本研究では一つ一つのクローンを単位とした履歴を抽出しており，より具象的な解析を行っている．また，本研究の目的は分岐クローンセットの発見による要修正箇所の提示という開発者に焦点を当てたものであるのに対して，Kim らの研究はクローンを生存期間という観点から分類することで，クローンの性質を探求することを目的としている．

任意のコード片ではなく，関数を対象とする履歴追跡手法もいくつか提案されている [11, 12]．Godfrey ら [11] は関数のペアを対象とした 5 つの評価基準を定義している．これらは関数名の類似度，関数宣言部の類似度，LOC 等のメトリクスの類似度，呼び出し元や呼び出し先の類似度，および上記 4 つをユーザが組みあわせたものから構成され，ユーザが指定した計算尺度に基づいてバージョン間の関数同士の対応関係を求めている．また Antoniol ら [13] は，クラス単位での履歴追跡手法を提案している．

このほかの関連研究として版管理システムを利用して何らかの有益な情報を引きだそうとする研究が挙げられる．これまでに，版管理システムに存在する開発履歴から関連性の高い関数を提示する研究 [14, 15] や，バグ修正支援 [16]，行数の変化や開発者ごとの編集を行った行数などを図示する試み [17, 18] が行われている．

7. ま と め

本論文では，クローン履歴解析手法を提案した．また PostgreSQL を対象に適用実験を行い，クローン履歴関係を用いて，かつてクローンだったコード片が正しく抽出できること，抽出したコード片は派生元のコード片と強い関連を持っていることを確認した．

今後の課題としては，まず，より目的に沿った形で

のクローン履歴情報閲覧システムの構築が挙げられる．クローンの履歴は開発プロセス全体の把握やクローン発生源の特定，クローンの管理等さまざまな応用が考えられるが，抽出結果を活用するためには分析した結果を目的に沿う形で表示する必要がある．

また分析手法そのものについても評価，改善が必要である．クローンの履歴となりうるもののうち本手法で分析できる範囲，できない範囲を厳密に調査する必要がある．また既存の類似手法との結果比較も重要であると考える．

現時点で判明している取得できないクローン履歴としては，一度削除されたクローンが，その後の変更により復活した場合がある．前述したとおり全てのバージョン間においてクローン履歴分析をするのは現実的ではないため，たとえば削除されたクローンについても逐次記録していき，過去の削除されたクローンも差分解析の対象として加えることが考えられる．しかし，この手法は削除されたクローン数によっては膨大な計算量を要するため，既存の開発パターンを調査して削除されたクローン数の推移傾向を調査したい．

謝辞 本研究において多大なご協力を頂いた産業技術総合研究所 神谷年洋氏に深く感謝する．また，本論文の一部は文部科学省「e-Society 基盤ソフトウェアの総合開発」および日本学術振興会の科学研究費補助金基盤研究 (A)（課題番号：17200001）の助成を得た．

文　献

[1] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A Multi-Linguistic Token-based Code Clone Detection System for Large Scale Source Code,” IEEE Trans. Software Engineering, vol.28, no.7, pp.654-670, 2002.

[2] E. Burd, and J. Bailey, “Evaluating clone detection tools for use during preventative maintenance,” Proc. 2nd IEEE Int. Workshop on Source Code Analysis and Manipulation (SCAM 2002), pp.36-43, Montreal, Canada, Oct 2002.

[3] B.S. Baker, “A program for identifying duplicated code,” Computing Science and Statistics, vol.24, pp.49-57, 1992.

[4] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, “CP-Miner: A tool for finding copy-paste and related bugs in operating system code.,” Proc. Sixth Symposium on Operating System Design and Implementation (OSDI’04), pp.289–302, 2004.

[5] J.H. Johnson, “Identifying redundancy in source code using fingerprints,” Proc. Centre for Advanced Studies on Collaborative research (CASCON’93), pp.171-183, Toronto, Ontario, Canada, Oct 1993.

[6] E. Merlo, G. Antoniol, M.D. Penta, and V.F. Rollo, “Linear complexity object-oriented similarity for clone detection and software evolution analyses,” Proc. 20th IEEE Int. Conf. on Software Maintenance (ICSM’04), pp.412-416, Chicago, Illinois, USA, Sep 2004.

[7] 門田暁人，佐藤慎一，神谷年洋，松本健一，“コードクローンに基づくレガシーソフトウェアの品質の分析,” 情報処理学会論文誌，vol.44，no.8，pp.2178-2188，2003 年 8 月.

[8] CVS, http://www.cvshome.org/.

[9] Subversion, http://subversion.tigris.org/.

[10] M. Kim, and D. Notkin, “Using a clone genealogy extractor for understanding and supporting evolution of code clones,” Proc. 2nd Intl. Workshop on Mining Software Repositories (MSR 2005), pp.17-21, Saint Louis, Missouri, May 2005.

[11] M.W. Godfrey, and L. Zou, “Using origin analysis to detect merging and splitting of source code entities,” IEEE Trans. Software Engineering, vol.31, no.2, pp.166-181, Feb 2005.

[12] N. Gold, and A. Mohan, “A framework for understanding conceptual changes in evolving source code,” Proc. 19th Intl. Conf. on Software Maintenance (ICSM 2003), pp.22-26, Amsterdam, 2003.

[13] G. Antoniol, M.D. Penta, and E. Merlo, “An automatic approach to identify class evolution discontinuities,” Proc. 7th Int. Workshop on Principles of Software Evolution (IWPSE’04), pp.31-40, Kyoto, Japan, Sep 2004.

[14] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller, “Mining version histories to guide software changes,” IEEE Trans. Software Engineering, vol.31, no.6, pp.429-445, Jun 2005.

[15] D. Cubranic, G.C. Murphy, J. Singer, and K.S. Booth, “Hipikat: A project memory for software development,” IEEE Trans. Software Engineering, vol.31, no.6, pp.446-465, Jun 2005.

[16] C.C. Willams, and J.K. Hollingsworth, “Automatic mining of source code repositories to improve bug finding techniques,” IEEE Trans. Software Engineering, vol.31, no.6, pp.466-480, Jun 2005.

[17] S.G. Eick, T.L. Graves, A.F. Karr, A. Mockus, and P. Schuster, “Visualizing software changes,” IEEE Trans. Software Engineering, vol.28, no.4, pp.396-412, Apr 2002.

[18] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler, “A system for graph-based visualization of the evolution of software,” Proc. ACM symposium on Software visualization (SOFTVIS 2003), pp.77-86, San Diego, California, USA, Jun 2003.

（平成 xx 年 xx 月 xx 日受付）

川口　真司

平 13 阪大・基礎工・情報卒．平 18 同大大学院博士課程了．同年奈良先端大・情報・情報システム・助手．博士 (情報科学)．ソフトウェアプロセス，ソフトウェア開発支援の研究に従事．情報処理学会会員．

松下　誠

平 5 阪大・基礎工・情報卒．平 10 同大大学院博士課程了．同年同大・基礎工・情

報・助手．平 14 阪大・情報・コンピュータサイエンス・助手．平 17 阪大・情報・コンピュータサイエンス・助教授．博士 (工学)．ソフトウェアプロセス，オープンソースソフトウェア開発の研究に従事．情報処理学会，日本ソフトウェア科学会，ACM 各会員．

井上　克郎　（正員）

昭 54 阪大・基礎工・情報卒．昭 59 同大大学院博士課程了．同年同大・基礎工・情報・助手．昭 59〜61 ハワイ大マノア校・情報工学科・助教授．平元阪大・基礎工・情報・講師．平 3 同学科・助教授．平 7 同学科・教授．平 14 阪大・情報・コンピュータサイエンス・教授．博士 (工学) ソフトウェア工学の研究に従事．情報処理学会，日本ソフトウェア科学会，IEEE，ACM 各会員．

Abstract Since a code clone is one of the major problems for maintenance phase, several code clone detection methods have been proposed. We focus on another clone relation, branched clone set - clone sets used to be one clone set. Such clone sets are also highly related. To retrieve branched clone sets, we propose a method retrieving histories of code clones. We applied our method for PostgreSQL and evaluated whether we can get branched clone sets using code clone histories.

Key words Code Clone, Revision, Software Repository