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
SS2005-31 (2005-08)

版管理システムを用いたコードクローン履歴分析

川口 真司† 松下 誠† 井上 克郎†

† 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: †{s-kawagt,matusita,inoue}@ist.osaka-u.ac.jp

あらまし 本発表ではコードクローンの履歴を抽出する手法を提案する．従来のコードクローン分析手法は，その時点でのソースコードからコードクローンを検出するものであるが，このクローン分析を過去の時点に遡って順次適用することでコードクローンの履歴を抽出することを考える．クローンの履歴を用いることで，かつてクローン関係にあったコード片の抽出や，クローンを発生時期に基づいて大量のクローンを分類するなどさまざまな活用方法が考えられる．また PostgreSQL に対して提案手法を適用し，クローン履歴の活用方法について考察する．

キーワード コードクローン，履歴，ソフトウェアリポジトリ

Code Clone Origin Analysis Using Version Control System

Shinji KAWAGUCHI†, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University Machikaneyama-cho 1–3, Toyonaka-shi, 560–8531 Japan
E-mail: †{s-kawagt,matusita,inoue}@ist.osaka-u.ac.jp

Abstract We propose a method retrieving histories of code clones. Many code clone detection methods are proposed, but few researches forcused on histories of code clones. Histories of code clone is useful for retrieving somewhile clone relationship and grouping many code clones by their appearance time. We applied our method for PostgreSQL and considered usage of code clone histories.

Key words Code Clone, Revesion, Software Repository

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，コードクローン) が挙げられる．もしコードクローンの一つに不具合が見つかった場合には，すべてのコードクローンを調査し，それぞれに対して同様の修正を施さなければならない．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な作業となる．

この問題に対処するべく，これまでにコード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 [1]．このようなコード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コードクローンを発見することを可能にする．また発見されたコードクローンを適切な方法で抽象化することでコードクローンの解消に役立てられている．

しかし，最新のバージョンを分析するだけでは抽出できないコードクローン関係も存在する．例えば初期の実装ではコードクローン関係にあり，開発の進展と伴なってクローン関係でなくなった場合である．このような開発初期にコードクローンであった部分は，そうでなくなった場合にも強い関連があると考えられる．このような関連を抽出するには過去にさかのぼってクローン分析を行う必要がある．

本研究ではコードクローンを分析するための手法として，コードクローン履歴分析を提案する．コードクローン履歴分析では，現時点のコードクローンが過去のバージョンのどのコードクローンに対応するのか，すなわち今あるコード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．

コードクローンの履歴を解析することで，現在クローンである部分に加えて，過去にクローン関係にある部分も抽出することができる．その他にも広範囲に散らばっているクローンを調査するときにクローンの発生時期によって分類することなどにクローン履歴を活用できるのではないかと考えている．

また，時系列的に連続したコードクローンの分析を行うことで，ソフトウェアに含まれるコードクローンの量や割合の変化を連続的に調査・閲覧することができる．これらの情報はソフ

トウェアの開発工程や開発者の技量を評価する上での重要な指標となりうる．

2. クローン分析

クローン分析手法には大きくわけてソースコードの字面比較に基づく手法と，特徴メトリクスに基づく手法に分けられる．

ソースコードの字面比較に基づく手法では，基本的にソースコード中で同一の文字列を検索することでコードクローンの検出を行う．ただし，完全一致だけではなく，ある程度のあいまいさも含んだコードクローンも抽出できるようにしている．それに対して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．一般には字面ベースの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，字面比較ベースの検出ツールのひとつであるCCFinder [2] を利用してコードクローン履歴の分析を行う．CCFinder は字面比較を用いた手法のなかでも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う．また，実際にさまざまな大規模ソフトウェアに対して適用されており，その有用性が示されている [3].

本研究では字面比較ベースの検出方法を利用しているので，必然的に我々の手法が抽出するクローンも字面ベースのものとなる．本研究ではファイル名，開始行，終了行の 3 つの属性でクローンの位置を指定するものとする．

2.1 CCFinder

CCFinder は文字列比較に基づくクローン検出を行う．ただし CCFinder では変数名や関数名をすべて単一の識別子として認識する．たとえばプログラム中に変数 a，変数 b が存在した場合，これらの文字列は全く同一のものとして扱われる．このように識別子の名前をあえて無視することで，CCFinder は変数名のみが改変されたコードクローンも抽出することが可能である．

また抽出を行うクローンの規模を最小トークン数という形で指定することができる．この最小トークン数を大きな値に設定することで小さなコードクローンを無視して，より大きなクローンのみを抽出する．本パラメータを適切に設定することで非常に大規模なソフトウェアに対しても現実的な速度で解析できる．

3. 分析手法

3.1 分析対象

図 1 は時間 $t, t-1, \ldots$ におけるコードクローンの状態を表したものである．各 $V_t, V_{t-1}, \ldots$ は各時間におけるバージョンを表し，その上に点線で囲まれているのがそれぞれの時点におけるコードクローンである．本手法で抽出するコードクローン履歴とは，あるバージョン V_t に存在するクローンについて，V_{t-1} のクローンから対応するものを探しだすことである．あるクローン A について，過去のバージョンに対応するクローン B が

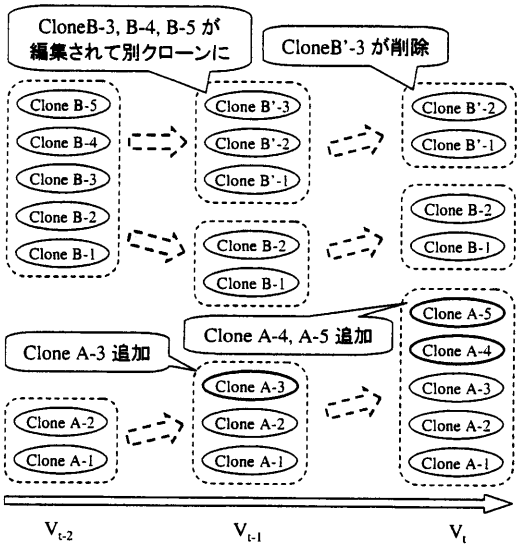

図 1 クローン履歴分析
Fig. 1 Clone Origin Analysis

ある場合，A と B の間の関係をクローン履歴対応関係と定義する．図 1 の例の場合，V_t の Clone A-1 と V_{t-1} の Clone A-1 などにはクローン履歴対応関係が成立する．

このクローン履歴対応関係を利用することで，例えば最新バージョンである V_t の時点では Clone B と Clone B' は無関係であるが，それらはかつて同一のクローンに属していたこと，すなわち同じコードから発展したコードであり関連性の高いコードであることがわかる．逆に Clone A に含まれるクローンコードは 5 つあるが，最初からある Clone A-1, A-2 と最新バージョンで追加された Clone A-4, A-5 とでは性質を異にする可能性も考えられる．

クローン履歴分析は隣りあう二つのバージョン間の分析に限定する．そのため，本分析では一度削除されたクローンがまた復活した場合には無力である．しかし，過去の全てのバージョンに対して分析を行うことは計算量コストの観点から見ても現実的ではない．

なおコードクローン解析の対象は与えられたプロダクト一式である．すなわち，ソースコードだけでなく，それに付随するドキュメントや各種ツール等すべて含む．

3.2 クローン履歴対応関係抽出手法の概要

クローン履歴対応関係抽出手法は大きくわけると (1) 時間区切りごとに通常のクローン分析を行って各バージョンに含まれるコードクローンの抽出を行い，(2) 隣りあうバージョンに含まれるクローン同士のクローン履歴対応関係を分析する，という 2 つのステップで構成される．(2) においては隣りあうバージョン間の差分情報を利用して，隣りあうバージョン全体を解析するよりも少ない計算コストでクローン履歴分析を行う．

新旧バージョン間のクローン履歴対応関係を分析するために，まずバージョン V_t とバージョン V_{t-1} の間の差分を取る．ここでいう差分とは純粋に文字情報上での差分情報である．そして旧バージョンと差分文字列の間でクローン解析を行い追加された行に含まれるクローンを発見する．

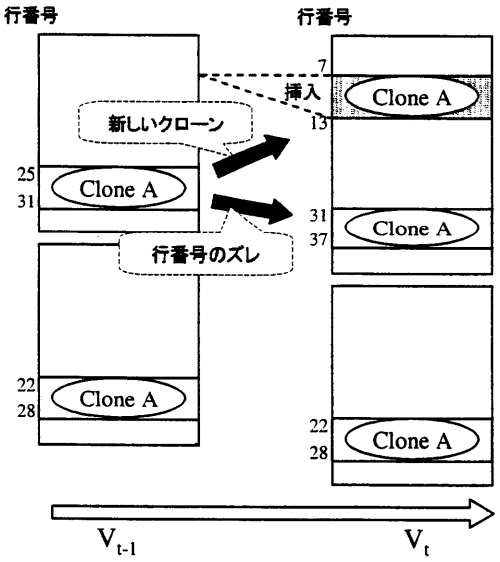

図 2　クローンが追加された状態
Fig. 2　Evolution of Clone

また図 2 で示したように，クローンそのものが何ら変更されていない場合でも編集操作によって行番号が前後する場合がある．このような行番号の変化にも追随するために，バージョン間に存在する編集操作を勘案して対応関係を保存する．

3.3　クローン履歴対応関係抽出アルゴリズム

図 3 にクローン履歴対応関係抽出アルゴリズムの概要を示す．V_t は新バージョンのソースコード集合を，V_{t-1} は一つ前のバージョンのソースコードを表す．ここでは V_{t-1} までの解析がすでに終わっている状態で最新バージョンの V_t が到着したときのアルゴリズムを示す．

（1）　V_t からコードクローンを抽出

まず最初に CCFinder を用いて V_t そのものについてクローン分析を行う．ただし CCFinder はファイル形式をあらかじめ指定する必要がある．そこで拡張子を元にファイルを Java ファイル，C ファイル，テキストファイル，バイナリファイルの 4 つに分類する．そしてバイナリファイルを除く 3 形式について，それぞれに対応するモードで CCFinder を起動する．また，Java - テキストファイル間などの異種ファイル間については plaintext モードで分析を行う．

この段階で各ファイル・ディレクトリについて各種メトリクスも計算し，記録する．現時点で記録している項目は更新日時，ファイル種別，行数，クローンを除いた行数である．ディレクトリの行数とは，ここではそのディレクトリ以下に含まれる全ファイルの行数の合計である．クローンを除いた行数も同様とする．

（2）　編集されていないコードクローンを追跡

diff に含まれるクローン履歴対応関係分析に先立って，V_t に含まれるクローンについて，V_{t-1} の対応する行にクローンが存在するかどうかを分析する．この分析により V_{t-1}，V_t 間で変更が加えられていないクローンを追跡する．

コードクローン自体に編集操作が加えられていない場合でも，クローンより上の部分で編集操作が加えられていた場合には行番号にずれが生じる．ここでは，そのようなずれを計算に入れたうえで対応関係の抽出を行う．

（3）　追加されたコードクローンを追跡

最後に V_{t-1} と V_{t-1}, V_t 間の差分との間でクローン分析を行う．そして V_t に追加された行にクローンが発見された場合，それは V_t に新しいクローンが存在することを意味する．これまでの工程で各行にどのクローンが存在するかは判別済みなので，差分クローンが存在する部分のクローン間についてクローン履歴関係を設定する．

なお，各バージョンに含まれるクローン，およびクローン間の対応関係も行数などと同様に保持する．

4.　実　　験

本実験では，PostgreSQL を対象として抽出したクローン履歴をどのように活用できるかを示す．その結果として，コードクローン履歴を見ることでそのコードクローンを分析する上において実際にどのような知見が得られるか，またコードクローン量の変遷を追うことがどのような効果があるかについて論ずる．

ここでは，あるコードクローンに着目したときのクローン履歴情報の分析と，ソースコード全体に含まれるクローン量の変遷を図示という 2 種類の実験を行った．

4.1　クローン履歴の抽出

ここでは PostgreSQL に含まれるクローンの中から，履歴を辿ることで得られるクローン履歴関係の例をとりあげる．

2004/07/01 の時点のソースコードを解析した結果，src/backend/commands/aggregatecmds.c には src/backend/commanc 等の 7 つのファイルとの間にクローンが含まれていた．図 5 の太字になっている部分が，これらのクローンの具体例である．これら 7 つのクローンはすべて互いが互いをクローン関係とする形で検出された．

そして約一ケ月後の 2004/08/05，これらのクローンのうち 3 ケ所に変更が加えられ，その結果，さきほどのクローン集合は変更されたコード片の集合 (Clone 1-2) と変更されていないコード片の集合の二つ分割された (図 4)．図 6 に Clone 1-2 の具体例を示す．

最新版に対しクローン分析を行った場合には，Clone 1 と Clone 1-2 は無関係なクローンの集合であるが，実際のコードサンプルを見ればわかるとおり，Clone 1，Clone 1-2 ともに同様の処理を行う部分であり関連性が高いと言える．つまり，これらの部分に変更を加えるときには Clone 1 と Clone 1-2 の両方に注意した上で変更する必要がある．このように「かつてクローンだったことがある」という関係も考慮に入れることもクローン分析において重要である．

4.2　クローン量変遷グラフ

次に PostgreSQL の開発工程においてコードクローンの量が全体的にどのように変遷したかを分析した．分析は 1998 年 7 月から 2005 年 6 月までの 7 年分のデータを一月ごとに区切って行った．図 7 が PostgreSQL のプロダクトの行数とその中に含まれるクローンの行数を表したグラフである．グラフは PostgreSQL

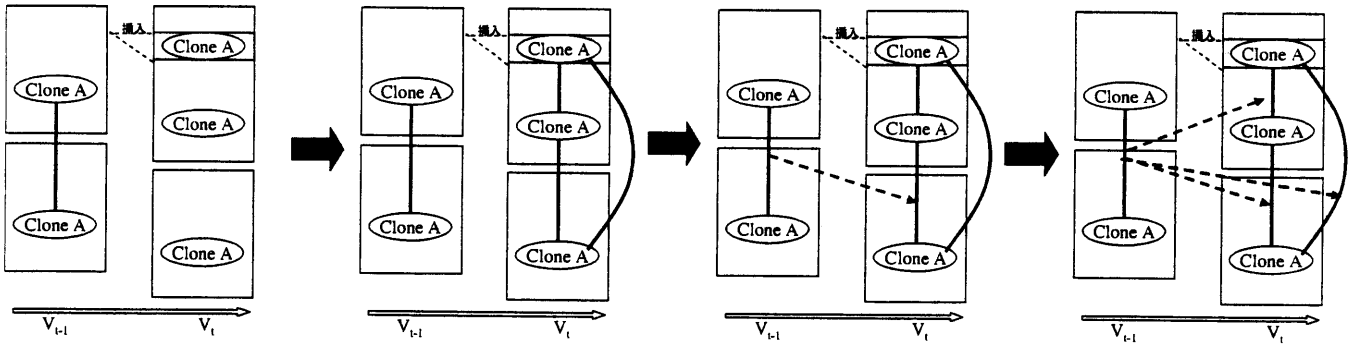

図 3　抽出アルゴリズム

Fig. 3　Retrieving Algorithm

```
pgsql/src/backend/commands/aggregatecmds.c 07/01
  291 /*
  292  * Change aggregate owner
  293  */
  294 void
  295 AlterAggregateOwner(List *name, TypeName *basetype, AclId newOwnerSysId)
  296 {
  297     Oid             basetypeOid;
  298     Oid             procOid;
  ...
  321     if (!HeapTupleIsValid(tup)) /* should not happen */
  322         elog(ERROR, "cache lookup failed for function %u", procOid);
  323     procForm = (Form_pg_proc) GETSTRUCT(tup);
  324
  325     /*
  326      * If the new owner is the same as the existing owner, consider the
  327      * command to have succeeded.  This is for dump restoration purposes.
  328      */
  329     if (procForm->proowner != newOwnerSysId)
  330     {
  331         /* Otherwise, must be superuser to change object ownership */
  332         if (!superuser())
  333             ereport(ERROR,
  334                     (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
  335                      errmsg("must be superuser to change owner")));
  336
  337         /* Modify the owner --- okay to scribble on tup because it's a copy */
  338         procForm->proowner = newOwnerSysId;
  339
  340         simple_heap_update(rel, &tup->t_self, tup);
  341         CatalogUpdateIndexes(rel, tup);
  342     }
  343
  344     heap_close(rel, NoLock);
  345     heap_freetuple(tup);
  346 }
```

```
pgsql/src/backend/commands/dbcommands.c 07/01
  765 /*
  766  * ALTER DATABASE name OWNER TO newowner
  767  */
  768 void
  769 AlterDatabaseOwner(const char *dbname, AclId newOwnerSysId)
  770 {
  771     HeapTuple       tuple,
  772                     newtuple;
  ...
  790
  791     newtuple = heap_copytuple(tuple);
  792     datForm = (Form_pg_database) GETSTRUCT(newtuple);
  793
  794     /*
  795      * If the new owner is the same as the existing owner, consider the
  796      * command to have succeeded.  This is to be consistent with other objects.
  797      */
  798     if (datForm->datdba != newOwnerSysId)
  799     {
  800         /* changing owner's database for someone else: must be superuser */
  801         /* note that the someone else need not have any permissions */
  802         if (!superuser())
  803             ereport(ERROR,
  804                     (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
  805                      errmsg("must be superuser to change owner")));
  806
  807         /* change owner */
  808         datForm->datdba = newOwnerSysId;
  809         simple_heap_update(rel, &newtuple->t_self, newtuple);
  810         CatalogUpdateIndexes(rel, newtuple);
  811     }
  812
  813     systable_endscan(scan);
  814     heap_close(rel, NoLock);
  815 }
```

図 5　Clone 1 の抜粋

Fig. 5　Excerption of Clone 1

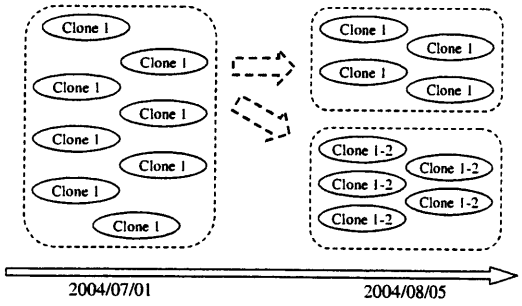

図 4　クローンが分裂した例

Fig. 4　An Example of Clone Branching

リポジトリを構成している主要なディレクトリについて，そのディレクトリの行数の総計をクローン部分とそうでない部分とに分けて表示している．また図中の実線は PostgreSQL リポジトリ全体でのクローン含有率の移りかわりを示す．

クローンの比率は初期には少しずつ増えてるものの，開発が進むにつれてクローンの占める割合が徐々に低下していることがわかる．これは既存のコードが正しく再利用されていること，すなわち開発されたソースコードの品質が高いことを示唆している．

PostgreSQL ではプロダクトの大部分を src ディレクトリが占めている．その src ディレクトリも複数のディレクトリから構成されているが，その中でも約 7 割を占める src/backend ディレクトリについても同様の分析を行った．図 8 に行数を，図 9 に，クローンが全体に占める割合の変化を示す．

これらのグラフからは 2000 年 10 月の時点で utils に大量のコードが追加されていること，追加されたコードの中にはコードクローンが含まれていないことなどが読みとれる．実際に差分を確認したところ，この間に文字コード変換機能が追加されており，utils 以下には文字コード間の対応表が追加されていたことが確認できた．

逆に SQL 命令を実際に処理する部分である src/backend/commands ディレクトリでは徐々にコードクローンの割合が増加している．特に上昇の著しい 2003 年 7 月と 2003 年 8 月の間の変更を調べたところ，エラーを処理するた

```
pgsql/src/backend/commands/dbcommands.c 08/04
 765 /*
 766  * ALTER DATABASE name OWNER TO newowner
 767  */
 768 void
 769 AlterDatabaseOwner(const char *dbname, AclId newOwnerSysId)
 770 {
 771     HeapTuple    tuple;
 772     Relation     rel;
 ...
 792     /*
 793      * If the new owner is the same as the existing owner, consider the
 794      * command to have succeeded.  This is to be consistent with other objects.
 795      */
 796     if (datForm->datdba != newOwnerSysId)
 797     {
 798         Datum       repl_val[Natts_pg_database];
 799         char        repl_null[Natts_pg_database];
 800         char        repl_repl[Natts_pg_database];
 801         Acl         *newAcl;
 802         Datum       aclDatum;
 803         bool        isNull;
 804         HeapTuple   newtuple;
 805
 806         /* changing owner's database for someone else: must be superuser */
 807         /* note that the someone else need not have any permissions */
 808         if (!superuser())
 809             ereport(ERROR,
 810                     (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
 811                      errmsg("must be superuser to change owner")));
 812
 813         memset(repl_null, ' ', sizeof(repl_null));
 814         memset(repl_repl, ' ', sizeof(repl_repl));
 815
 816         repl_repl[Anum_pg_database_datdba - 1] = 'r';
 817         repl_val[Anum_pg_database_datdba - 1] = Int32GetDatum(newOwnerSysId);
 818
 819         /*
 820          * Determine the modified ACL for the new owner.  This is only
 821          * necessary when the ACL is non-null.
 822          */
 823         aclDatum = heap_getattr(tuple,
 824                                 Anum_pg_database_datacl,
 825                                 RelationGetDescr(rel),
```

```
pgsql/src/backend/commands/tablecmds.c 08/04
 5100 /*
 5101  * ALTER TABLE OWNER
 5102  */
 5103 static void
 5104 ATExecChangeOwner(Oid relationOid, int32 newOwnerSysId)
 5105 {
 5106     Relation    target_rel;
 5107     Relation    class_rel;
 ...
 5142     /*
 5143      * If the new owner is the same as the existing owner, consider the
 5144      * command to have succeeded.  This is for dump restoration purposes.
 5145      */
 5146     if (tuple_class->relowner != newOwnerSysId)
 5147     {
 5148         Datum       repl_val[Natts_pg_class];
 5149         char        repl_null[Natts_pg_class];
 5150         char        repl_repl[Natts_pg_class];
 5151         Acl         *newAcl;
 5152         Datum       aclDatum;
 5153         bool        isNull;
 5154         HeapTuple   newtuple;
 5155
 5156         /* Otherwise, check that we are the superuser */
 5157         if (!superuser())
 5158             ereport(ERROR,
 5159                     (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
 5160                      errmsg("must be superuser to change owner")));
 5161
 5162         memset(repl_null, ' ', sizeof(repl_null));
 5163         memset(repl_repl, ' ', sizeof(repl_repl));
 5164
 5165         repl_repl[Anum_pg_class_relowner - 1] = 'r';
 5166         repl_val[Anum_pg_class_relowner - 1] = Int32GetDatum(newOwnerSysId);
 5167
 5168         /*
 5169          * Determine the modified ACL for the new owner.  This is only
 5170          * necessary when the ACL is non-null.
 5171          */
 5172         aclDatum = SysCacheGetAttr(RELOID, tuple,
 5173                                    Anum_pg_class_relacl,
 5174                                    &isNull);
```

図 6　Clone 1-2 の抜粋

Fig. 6　Excerption of Clone 1-2

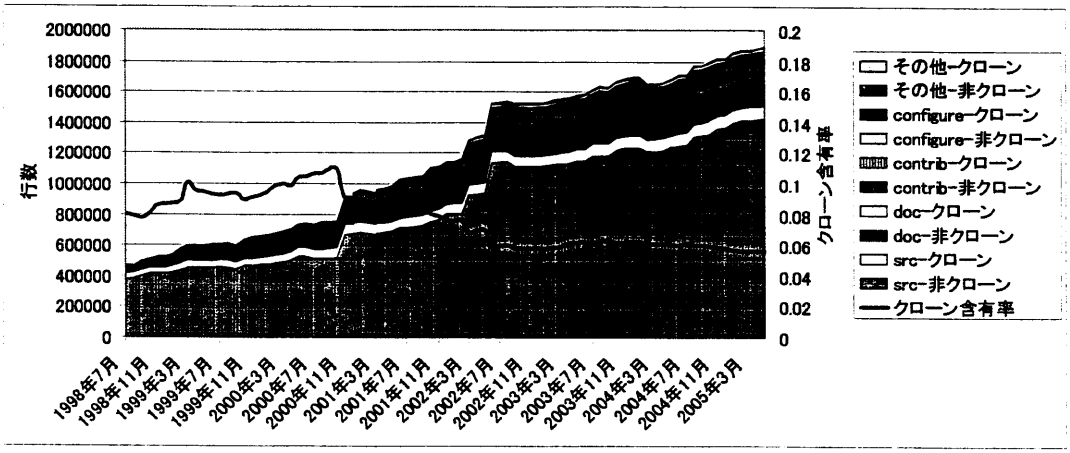

図 7　PostgreSQL の各サブディレクトリの LOC およびクローン無しの LOC の推移

Fig. 7　LOC and LOC without Clone

めの変更として同様のコードが追加されており，それらがコードクローンとして判定されていることがわかった．

このように，クローン量の変化を調査することで開発工程において不適切にコードクローンが増えていないかを監視したり，モジュールごとの傾向を調査したりすることができる．PostgreSQL を対象とした調査では開発が適正に進んでいるという推測ができるが，たとえばある時点において急激にコードクローンが上昇していたりしていれば，これを調査することができる．

5. 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 [4] 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いる．しかし，クローン間の履歴を抽出するために，一つ前のバージョンと今のバージョン全体を解析している．そのため非常に計算コストが高く大規模なソフトウェアに適用するためには，抽出するクローン粒度を非常に荒くしなければならない．

また関数を単位とした履歴追跡手法もいくつか提案されている [5,6]．Godfrey ら [5] 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．

6. まとめ

本研究では，クローン履歴解析手法を提案した．また Post-

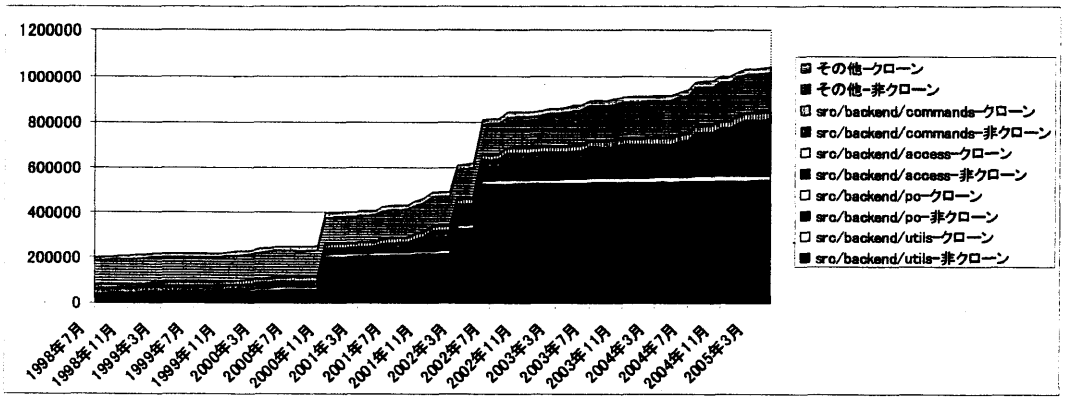

図 8　src/backend の LOC およびクローン無しの LOC の推移

Fig. 8　LOC and LOC without Clone

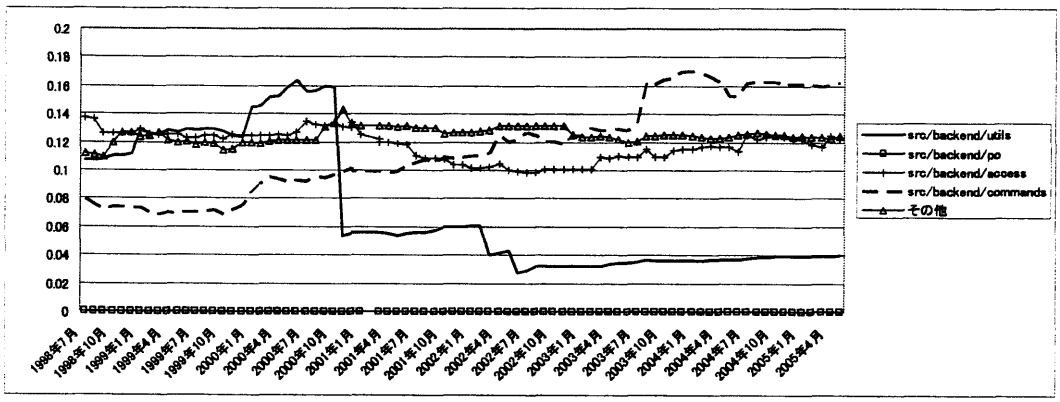

図 9　src/backedn 以下のサブディレクトリのコードクローンの割合

Fig. 9　LOC and LOC without Clone

greSQL を対象に適用実験を行い，かつてクローンだったコード片の抽出や，クローン量の変遷といった有用なデータが得られることを示した．

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．コードクローンの履歴は開発プロセス全体の把握やコードクローン発生源の特定，コード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，一度消えたクローンがまた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ので，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する必要があるであろう．

謝辞　本研究の一部は文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた．また，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] E. Burd and J. Bailey: "Evaluating clone detection tools for use during preventative maintenance", Proc. 2nd IEEE Int' l Workshop on Source Code Analysis and Manipulation (SCAM), pp. 36–43 (2002).

[2] 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, **28**, 7, pp. 654–670 (2002).

[3] 門田, 佐藤, 神谷, 松本："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, **44**, 8, pp. 2178–2188 (2003).

[4] M. Kim and D. Notkin: "Using a clone genealogy extractor for understanding and supporting evolution of code clones", MSR 2005, Saint Louis, Missouri, pp. 17–21 (2005).

[5] M. W. Godfrey and L. Zou: "Using origin analysis to detect merging and splitting of source code entities", IEEE Trans. Software Engineering, **31**, 2, pp. 166–181 (2005).

[6] N. Gold and A. Mohan: "A framework for understanding conceptual changes in evolving source code", ICSM '03, Amsterdam, pp. 22–26 (2003).