

# 版管理システムを用いた クローン履歴分析手法

川口真司(奈良先端大), ○松下誠(阪大)

kawaguti@is.naist.jp,

matusita@ist.osaka-u.ac.jp

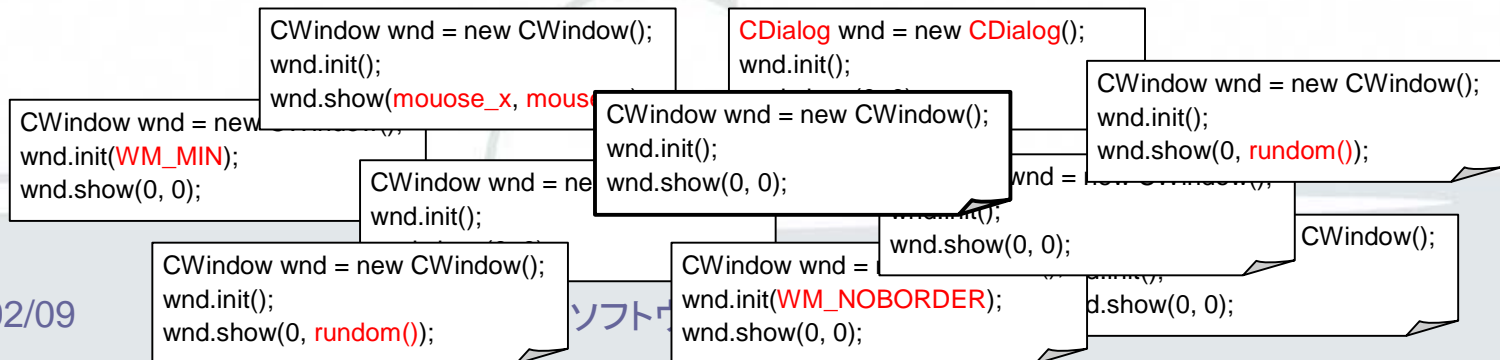
EASEプロジェクト

# レジュメ

- ✿ コードクローンとは?
- ✿ コードクローン履歴とは?
- ✿ コードクローン履歴の抽出方法
- ✿ PostgreSQL を対象にした分析
- ✿ 今後の展開

# コードクローンとは？

- ✧ プログラム中に何度も類似した箇所が現れるコード片
- ✧ 主に、安易なコピー & ペーストによって生まれる
- ✧ 大規模ソフトウェアの保守において問題となっている
  - ✧ ある一箇所のクローンにバグがあった場合、すべての類似箇所について同じ修正が必要かどうか、検討が必要
  - ✧ 若干の修正が加わっていることが多く、網羅的に把握することが困難



# コードクローン分析

## ✿ CCFinder: クローン自動抽出システム

- ✿ 大規模ソフトウェアに対して適用可能
- ✿ 文法エラーに対して頑健
- ✿ 実用的な時間で計算可能

## ✿ 解析結果から全体的な傾向は把握できる

## ✿ 実際にクローンを削除し、コード品質向上を図るのは難しい

- ✿ 膨大なクローン
- ✿ クローンの多様性
- ✿ 分析者の知識

# 問題点1: 膨大なクローン情報

- ✿ クローンが検出されないソフトウェアは稀
  - ✿ オープンソフトウェアでも1割程度はクローン
  - ✿ 100万行規模なら10万行程度は検出される
- ✿ 「どの部分から見ていけばいいのか・・・」
  - ✿ クローンが多数含まれているところから?

## 問題点2: クローンの多様性

- ✿ クローンが発生した理由はさまざま
  - ✿ コピー&ペーストされたコード
  - ✿ ソースファイルの編集権限の都合, コピーせざるをえなかったコード
  - ✿ ある種のイディオム
  - ✿ デザインパターンを構成しているコード
- ✿ 区別して対処しなくてはいけない
- ✿ 「なぜクローンができたのか?」の提示が必要

# 問題点3: 分析者の知識

- ✿ 個々人の分析者の知識は限られている
  - ✿ 開発者と分析者が別の人
  - ✿ 開発者であっても全領域を把握している人はいない (担当外のところは細かくわからない)
- ✿ 分析者は少数
- ✿ 「なぜクローンができたのか？」  
→「わからない」

# クローンのコンテキスト

- ✿ なに(What)がクローンか, は判明している
- ✿ クローンがなぜ(Why)生まれたか, を知るには..
  - ✿ クローンがいつ(When)作成されたのか
  - ✿ クローンをだれ(Who)が作成したのか
  - ✿ そのオリジナルはどこ(Where)にあったのか
  - ✿ クローンがどのように(How)発展したかを知ること
    - ✿ どの程度の頻度でコピーされているのか
    - ✿ 各クローンに対して編集操作はどの程度行われているのか

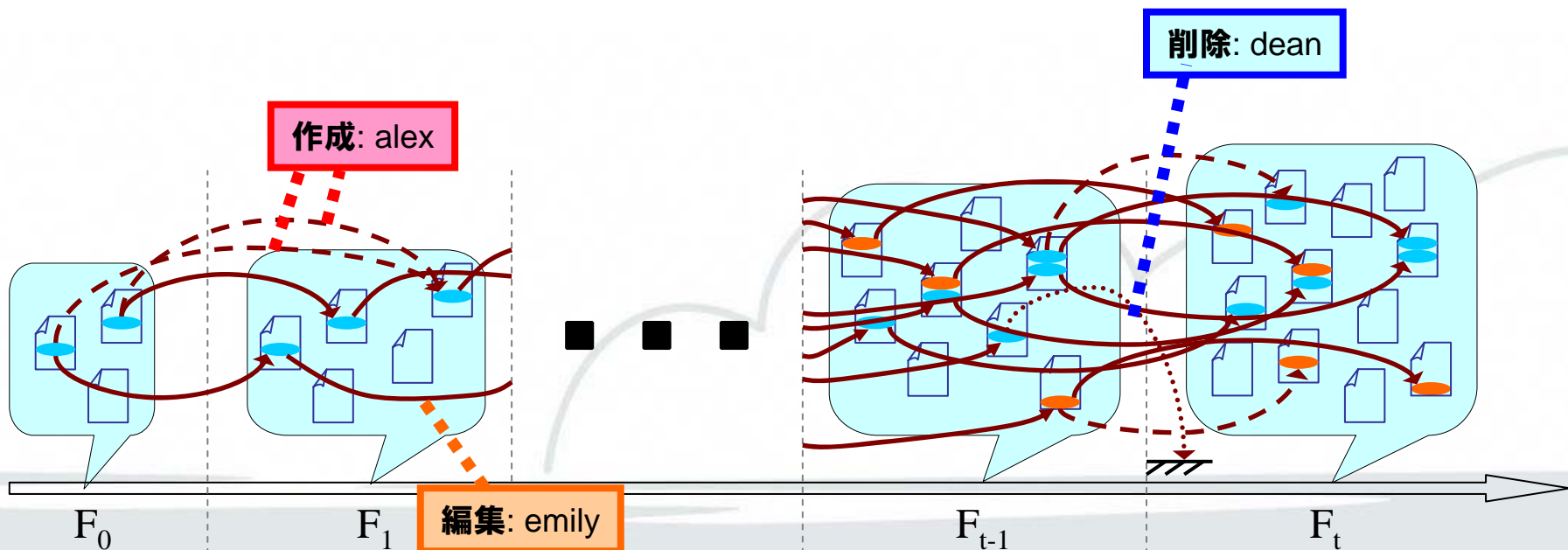


# コードクローン履歴とは？

❁ コードクローンの過去を提示するためのもの

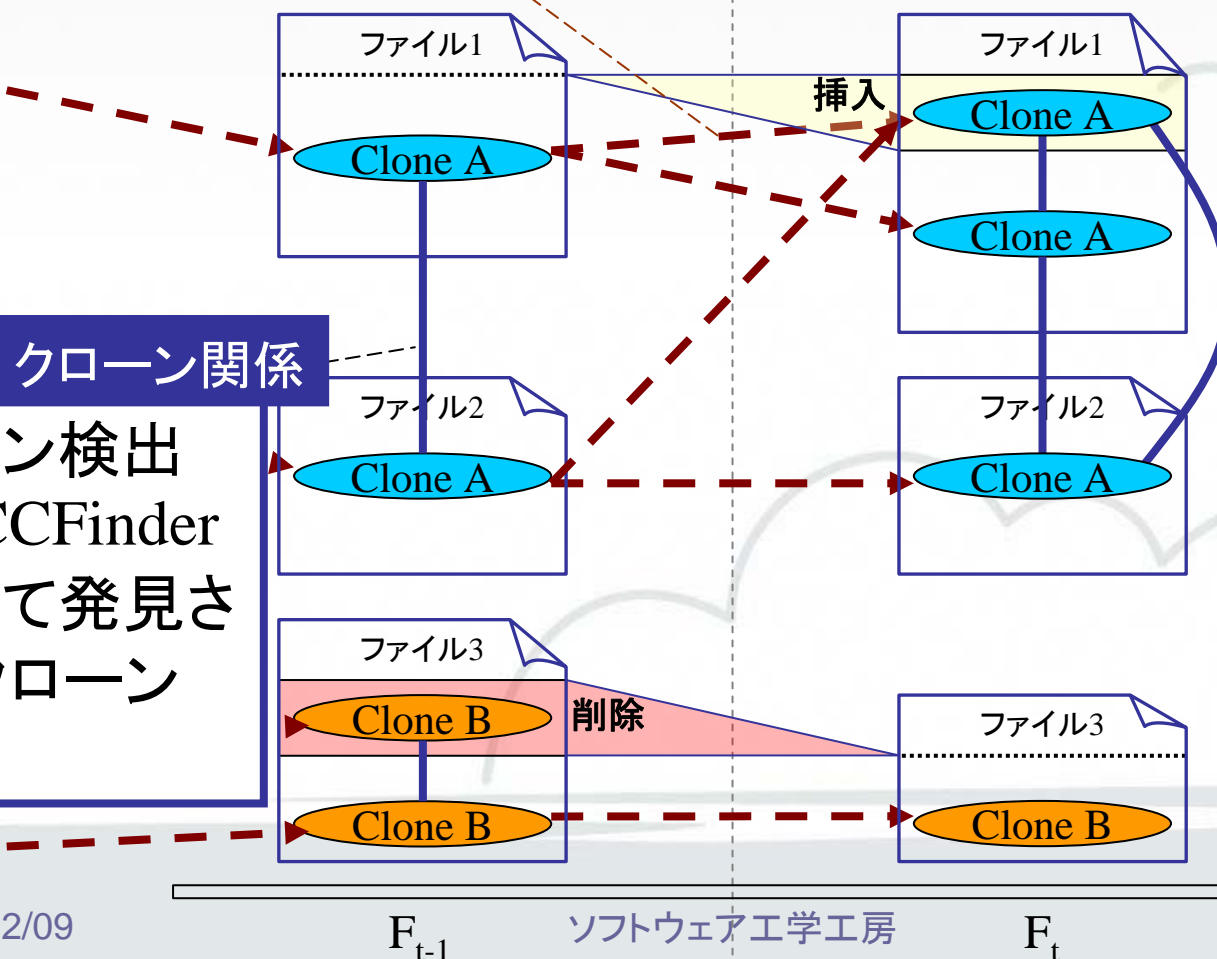
❁ いつコードクローンができたのか

❁ だれがコードクローンを作ったのか



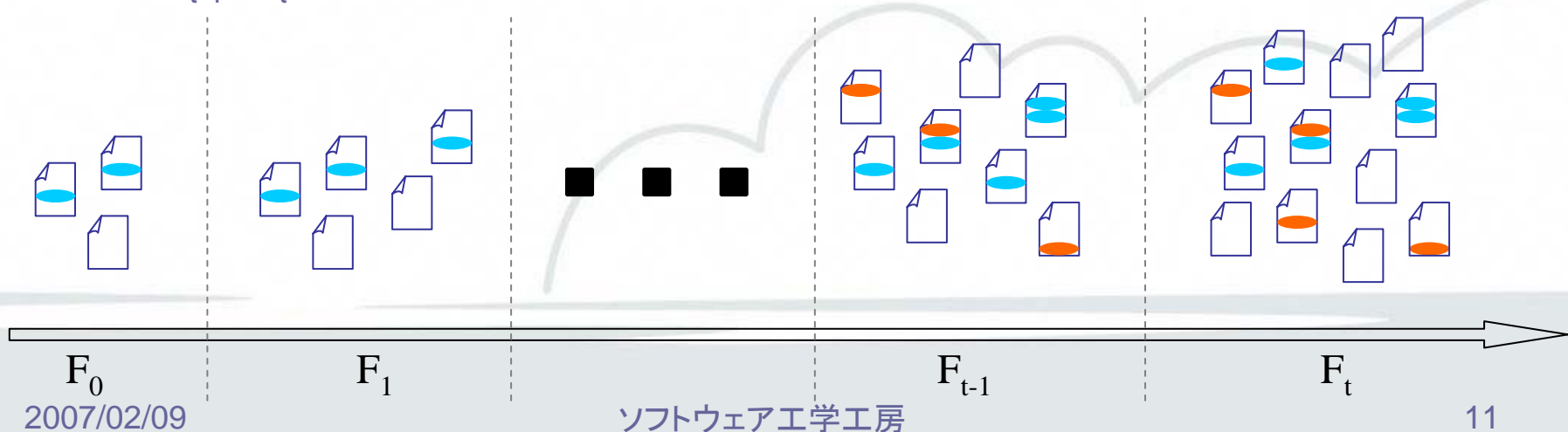
# クローン履歴関係の定義

クローン履歴関係 (過去のコード片, 現在のコード片) のペア



# クローン履歴抽出手法(1/2)

- ❧ 版管理システム(ex. CVS, Subversion, ...)を用いて過去の時点の製品を取得
- ❧ となりあう2時点間について逐次的に分析
  - ❧  $F_0, F_1$  をリポジトリから取得
  - ❧  $F_0, F_1$  間のクローン履歴関係を分析
  
- ❧  $F_t$  をリポジトリから取得
- ❧  $F_{t-1}, F_t$  間を分析



# クローン履歴抽出手法(2/2)

隣あう二時点のソースコード  $F_{t-1}$ ,  $F_t$  について

## 1. クローン分析

クローン分析には CCFinder を使用する  
クローンの行数、総行数に対する割合も計算

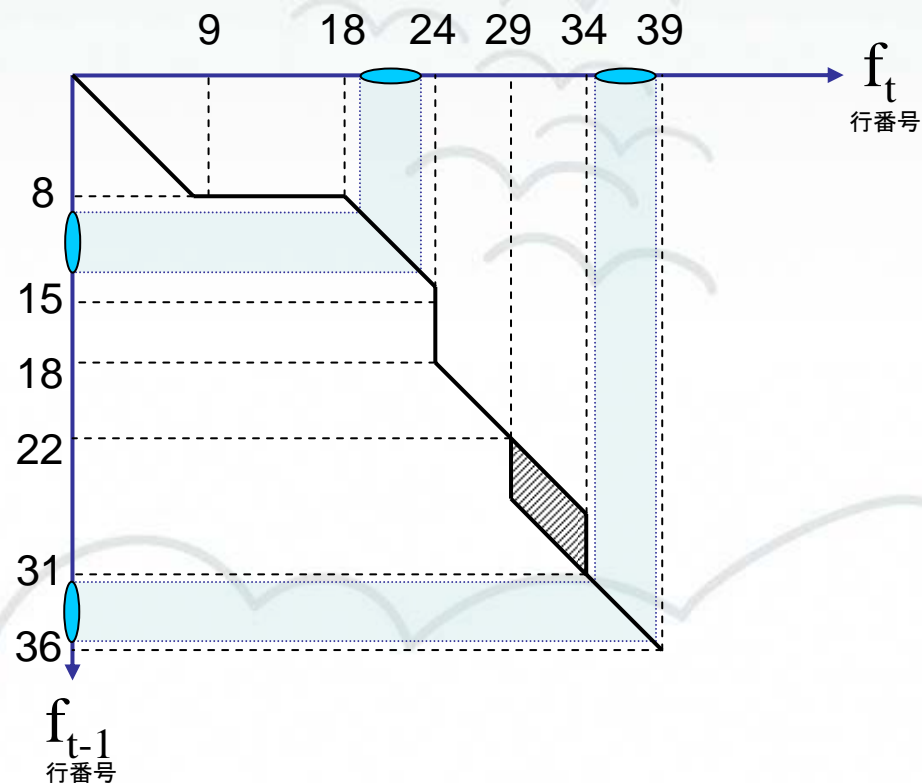
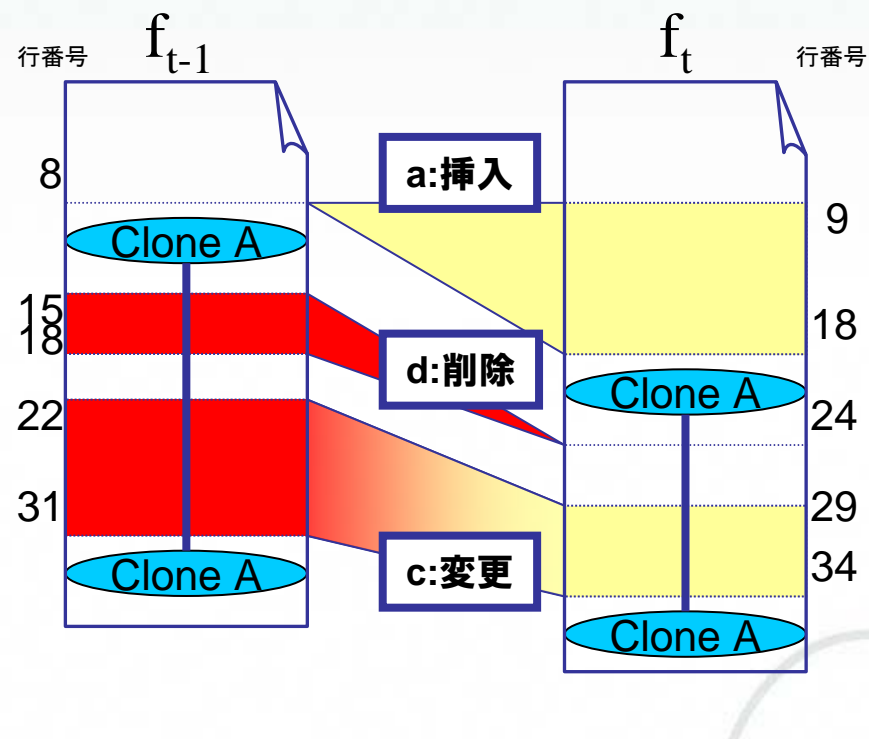
## 2. クローン履歴関係分析

1.  $HC_t$ : バージョン間でクローン関係に変化のない履歴関係
2.  $HA_t$ :  $F_t$  において新規に追加されたクローンの履歴関係
3.  $HT_t$ :  $F_t$  において片方が削除されたクローンの履歴関係

## 3. クローン履歴関係者分析

CVSのコミットログから, クローンを編集した開発者を特定する

# 行番号の調整



衝突時には対応行が一意でない

最小、最大の推定値を考慮

# クローン履歴

版管理システムについて

1.  $f_{t-1}$  に編集  
2.  $f_t$  に編集があれば、

**編集元の領域**

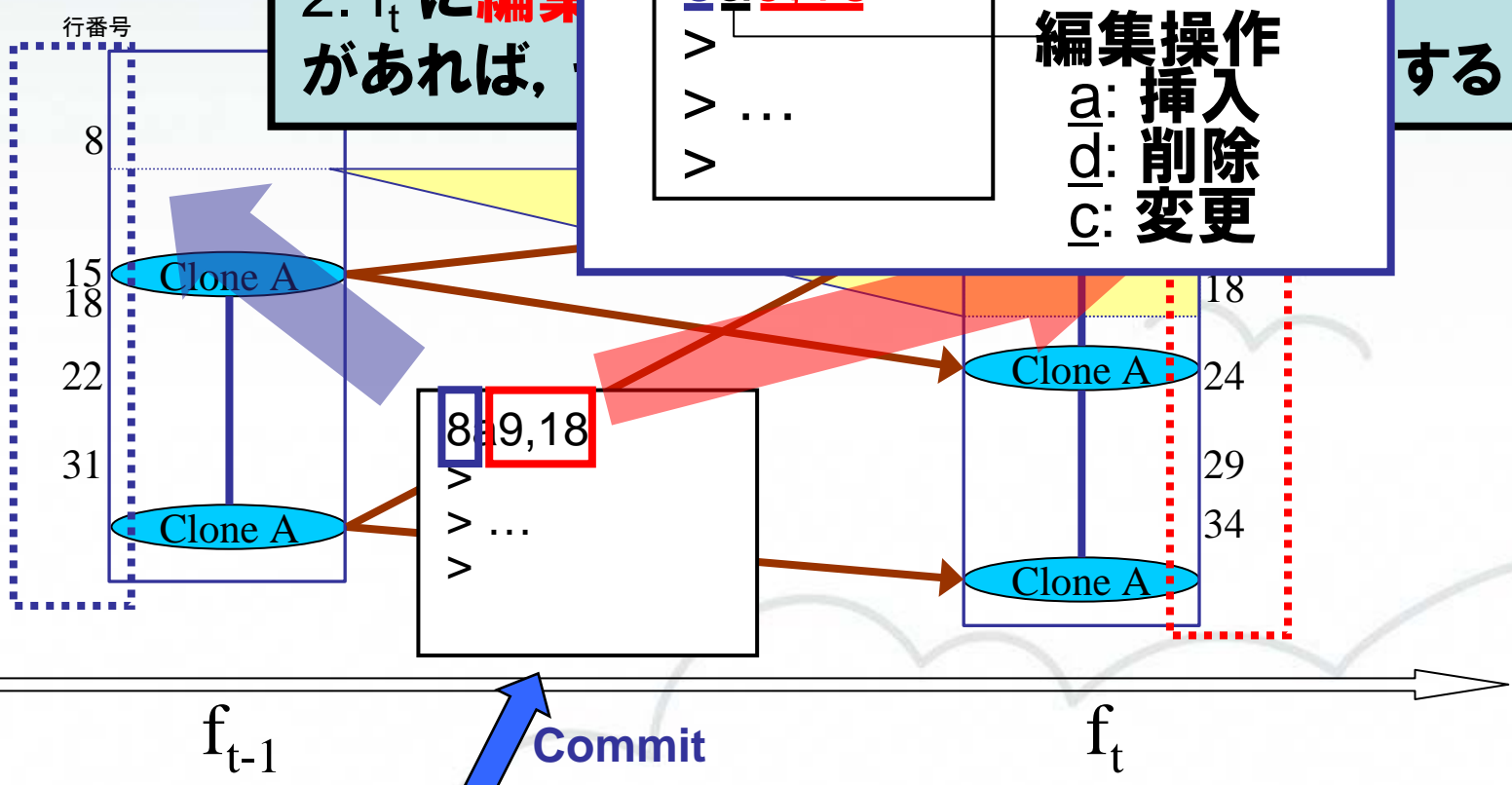
```
8a9.18
>
> ...
>
```

**編集先の領域**

**編集操作**

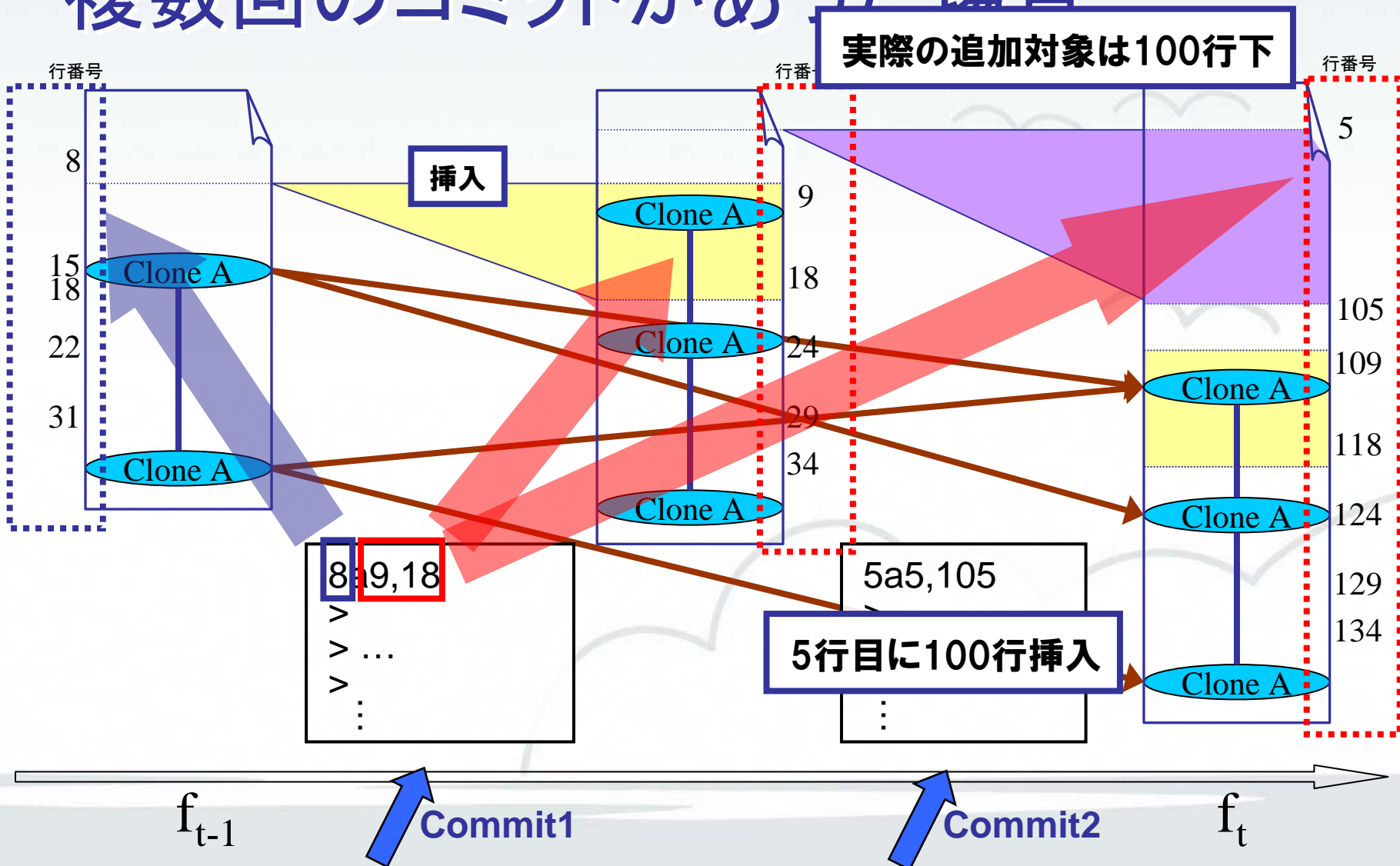
- a: 挿入
- d: 削除
- c: 変更

する



```
revision 1.82
date: 2003/07/20 21:56:32; author: tgl;
state: Exp; lines: +51 -29
Another round of error message editing, covering backend/commands/.
```

# 複数回のコミットがあった場合



# コードクローン履歴で実現できること

## クローンの観点からみた成果物・開発プロセスの評価

- ✧ 危険な兆候を持つモジュールの提示
  - ✧ クローンが増えつつある → 設計に問題があるのでは
- ✧ 削除すべきクローンの提示
  - ✧ クローンの生存期間を用いた分析
  - ✧ クローンの作成者や編集者の情報からのクローン品質推定
- ✧ 開発者にもとづいたクローンの評価
  - ✧ クローンを不用意に大量コピーしている開発者
  - ✧ 正当な理由があるクローンを作成している開発者



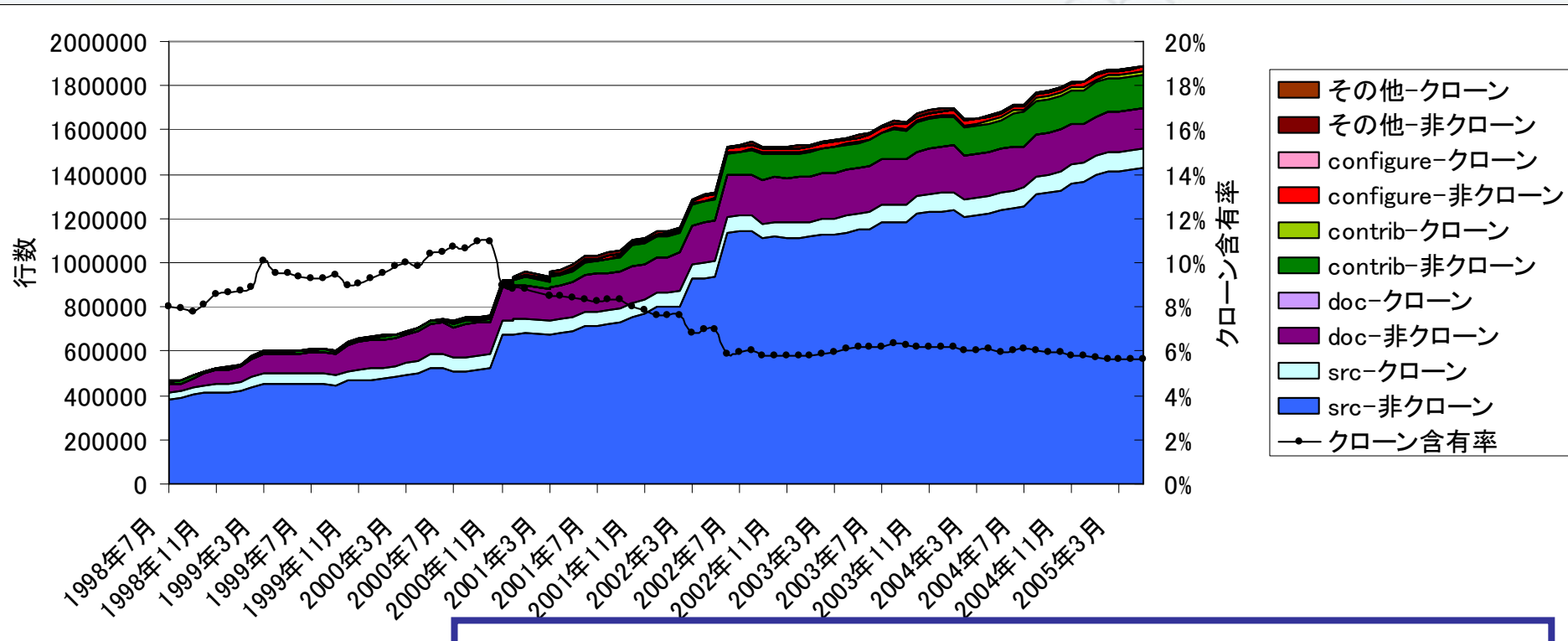
# 分析事例

1. クローン量変遷のグラフとその分析
2. 分岐しているクローンの例
3. コミットと、そこで編集されたクローンの性質との関連
4. 開発者と、クローン作成回数との関連

## 分析対象

- ✂ 1: PostgreSQL 全体  
1998年1月から2006年1月まで
- ✂ 2-4: PostgreSQL のDB処理実装部  
(src/backend/commands)  
2004年1月から2005年1月まで

# 分析1: PostgreSQL 全体のコード量



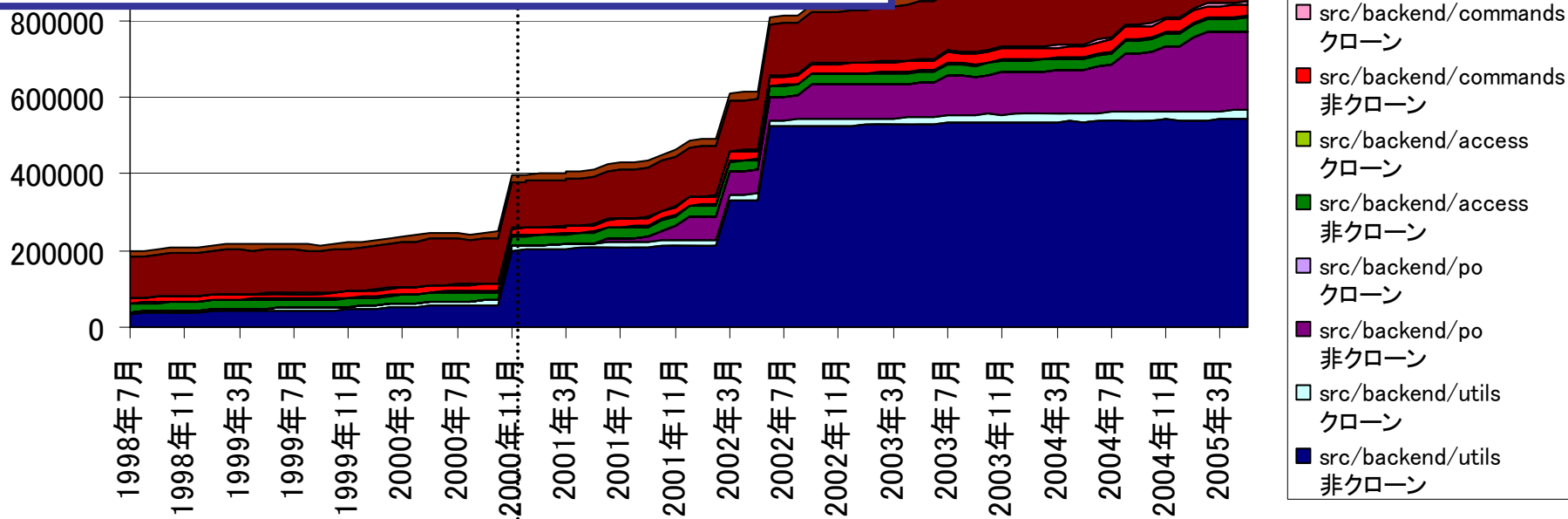
クローン含有率は開発工程全体をとおして安定

src 以下が全体の8割を占めている

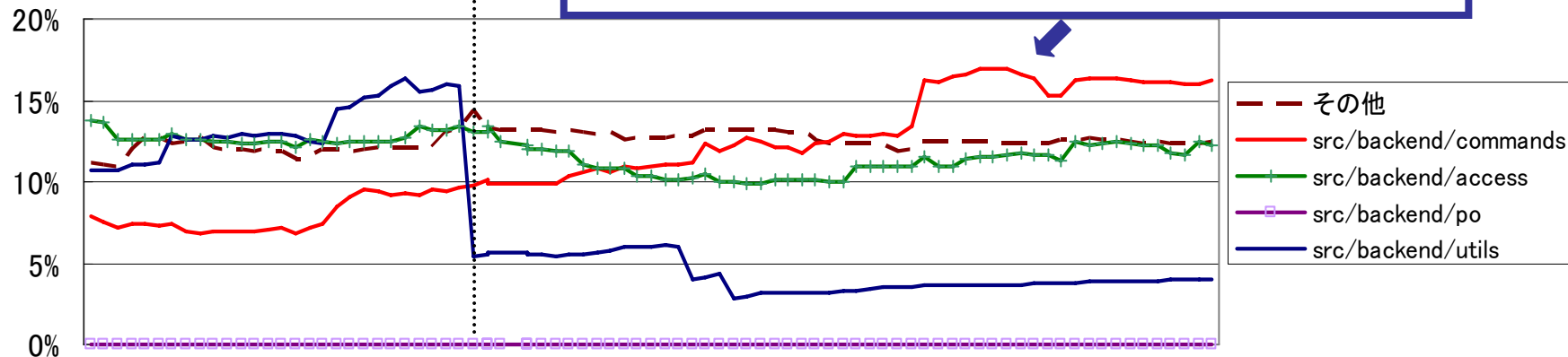
# 分析1: PostgreSQL コア部分のコード量

2000年11月、utils 以下のクローン含有率

utils には文字コード変換用データの大量追加



commands 以下のクローン率は徐々に上昇



p

DL

```

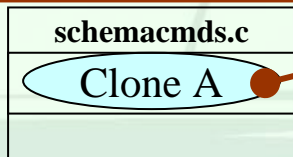
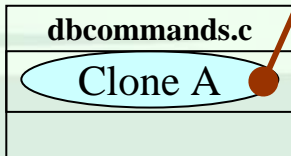
pgsql/src/backend/commands/dbcommands.c 07/01
765 /*
766  * ALTER DATABASE name OWNER TO newowner
767 */
768 void
769 AlterDatabaseOwner(const char *dbname, AclId newOwnerSysId)
770 {
771     HeapTuple     tuple,
772                 newtuple;
773     ...
774
775     newtuple = heap_copypuple(tuple);
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 }

```

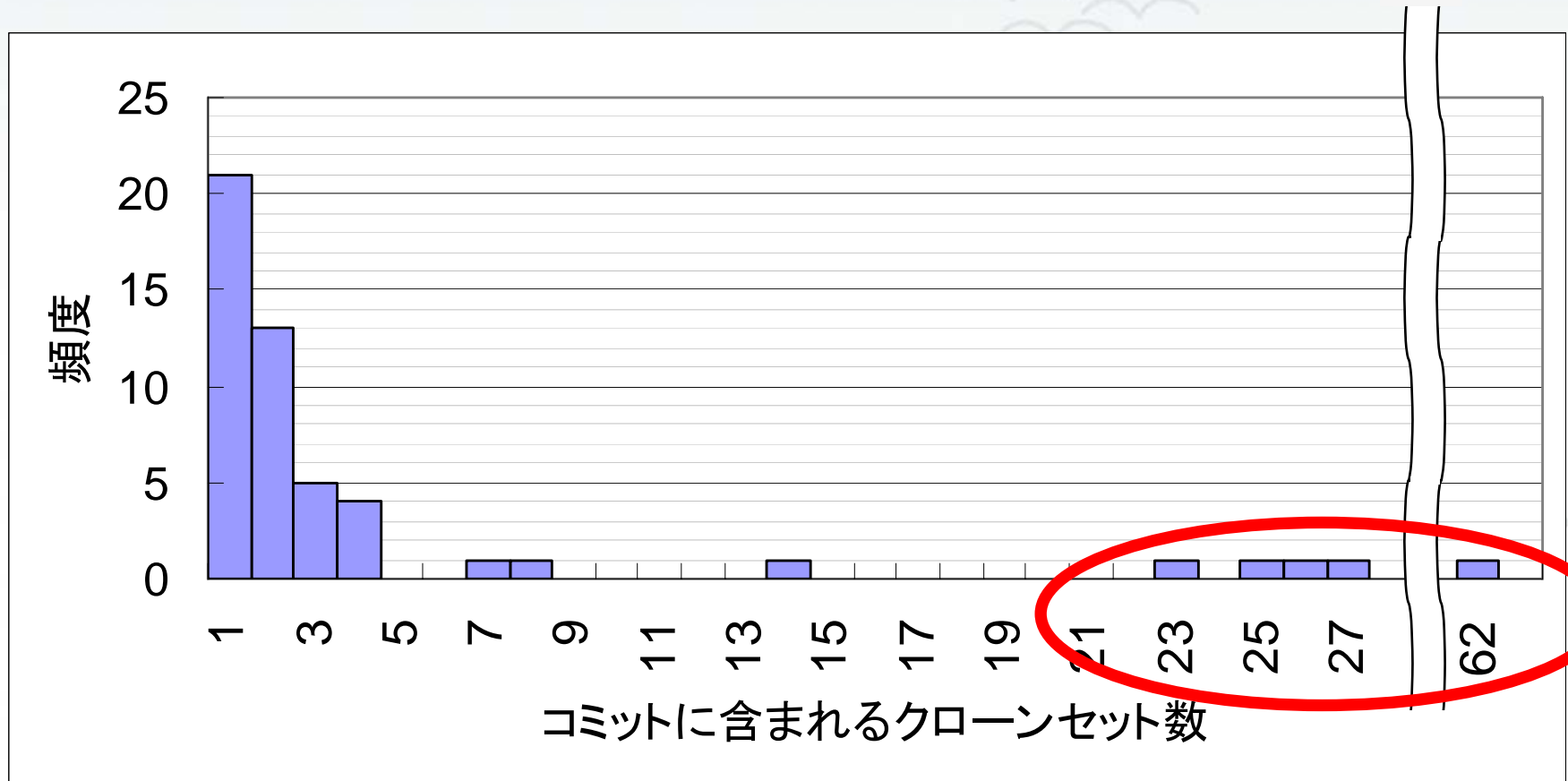
```

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;
773     ...
774
775     /*
776      * If the new owner is the same as the existing owner, consider the
777      * command to have succeeded. This is to be consistent with other objects.
778      */
779     if (datForm->datdba != newOwnerSysId)
780     {
781         Datum      repl_val[Natts_pg_database];
782         char        repl_null[Natts_pg_database];
783         char        repl_repl[Natts_pg_database];
784         Acl          *newAcl;
785         Datum      aclDatum;
786         bool        isNull;
787         HeapTuple   newtuple;
788
789         /* changing owner's database for someone else: must be superuser */
790         /* note that the someone else need not have any permissions */
791         if (!superuser())
792             ereport(ERROR,
793                     (errcode(ERRCODE_INSUFFICIENT_PRIVILEGE),
794                      errmsg("must be superuser to change owner")));
795
796         memset(repl_null, ' ', sizeof(repl_null));
797         memset(repl_repl, ' ', sizeof(repl_repl));
798
799         repl_repl[Anum_pg_database_datdba - 1] = 'r';
800         repl_val[Anum_pg_database_datdba - 1] = Int32GetDatum(newOwnerSysId);
801
802         /*
803          * Determine the modified ACL for the new owner. This is only
804          * necessary when the ACL is non-null.
805          */
806         aclDatum = heap_getattr(tuple,
807                                Anum_pg_database_dataacl,
808                                RelationGetDescr(rel),
809                                ...);
810
811         heap_close(rel, NoLock);
812         heap_freetuple(tup);
813     }
814 }

```



# 分析3: コミットから見た特性



☞ 大多数のコミットでは、そのとき編集されたクローンセットは一つ

☞ ただし、いくつかのコミットでは多数のクローンセットを一斉に編集している

# 大量のクローン編集時の編集内容

```
<   elog(WARNING, "DefineAggregate: attribute ¥"%s¥" not recognized",
<       defel->defname);
---
>   ereport(WARNING,
>           (errcode(ERRCODE_SYNTAX_ERROR),
>            errmsg("aggregate attribute ¥"%s¥" not recognized",
>                   defel->defname)));
```

“Another round of error message editing, covering backend/commands/.”

## ✂ 機械的な変換作業

エラーメッセージ出力部の一括変換

## ✂ クローンの発見が困難でなかったと考えられる

grep 等を利用した文字列検索可能なクローン

## ✂ このようなクローンは対処優先度が低い

✂ 機械的に処理をすべき対象を同定可能

✂ 保守工程において障害とならない

# 分析4: 開発者ごとの特性

- 対象サブモジュールに関係した開発
- クローンの変更に関わっていたのは

全コミット数に対して、  
クローン追加が多い

	momjian	petere	tgl
クローン追加	6	10	32
クローン削除	2	3	17
クローン編集	35	27	135
(小計)	43	40	184
総コミット数	1317	143	1428

# 今後の展開

## クローン解析の”とっかかり”を作る

### ✿ 分析仮説の立案

- ✿ 履歴に基づいた分析

  - ✿ 開発者

- ✿ 履歴と他のデータを組み合わせた分析

  - ✿ クローンそのもののメトリクス, バグ帳票,

### ✿ さまざまなソフトウェアにおいて仮説の検証

- ✿ オープンソースソフトウェアを対象とした解析のみ  
実際の開発現場との乖離が大きい

### ✿ クローン履歴閲覧システムの作成

- ✿ クローンの昔を容易に閲覧できる