版管理システムを用いたクローン履歴分析手法

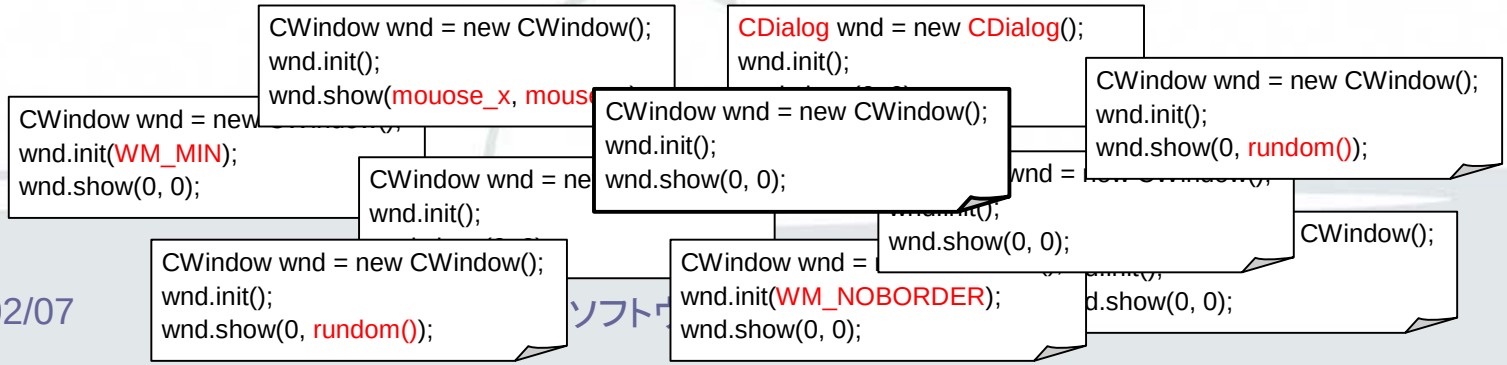
川口真司(奈良先端大), ○松下誠(阪大)
kawaguti@is.naist.jp,
matusita@ist.osaka-u.ac.jp

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

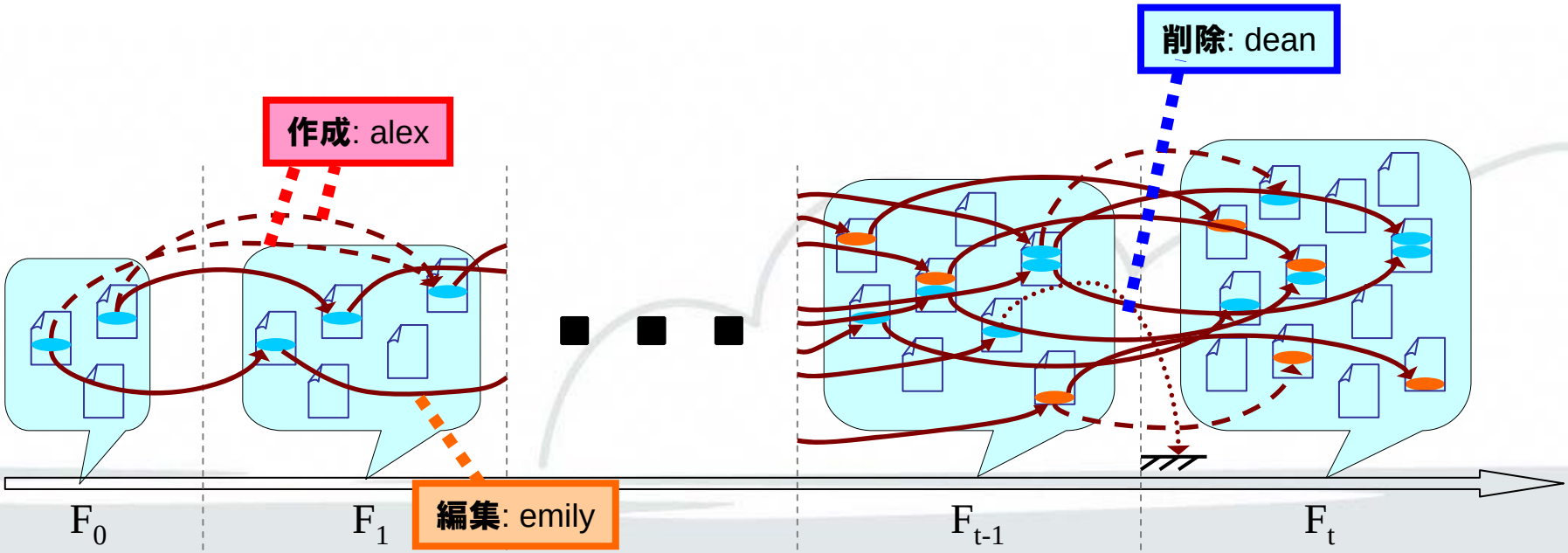
- 「なぜクローンができたのか?」
  →「わからない」

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

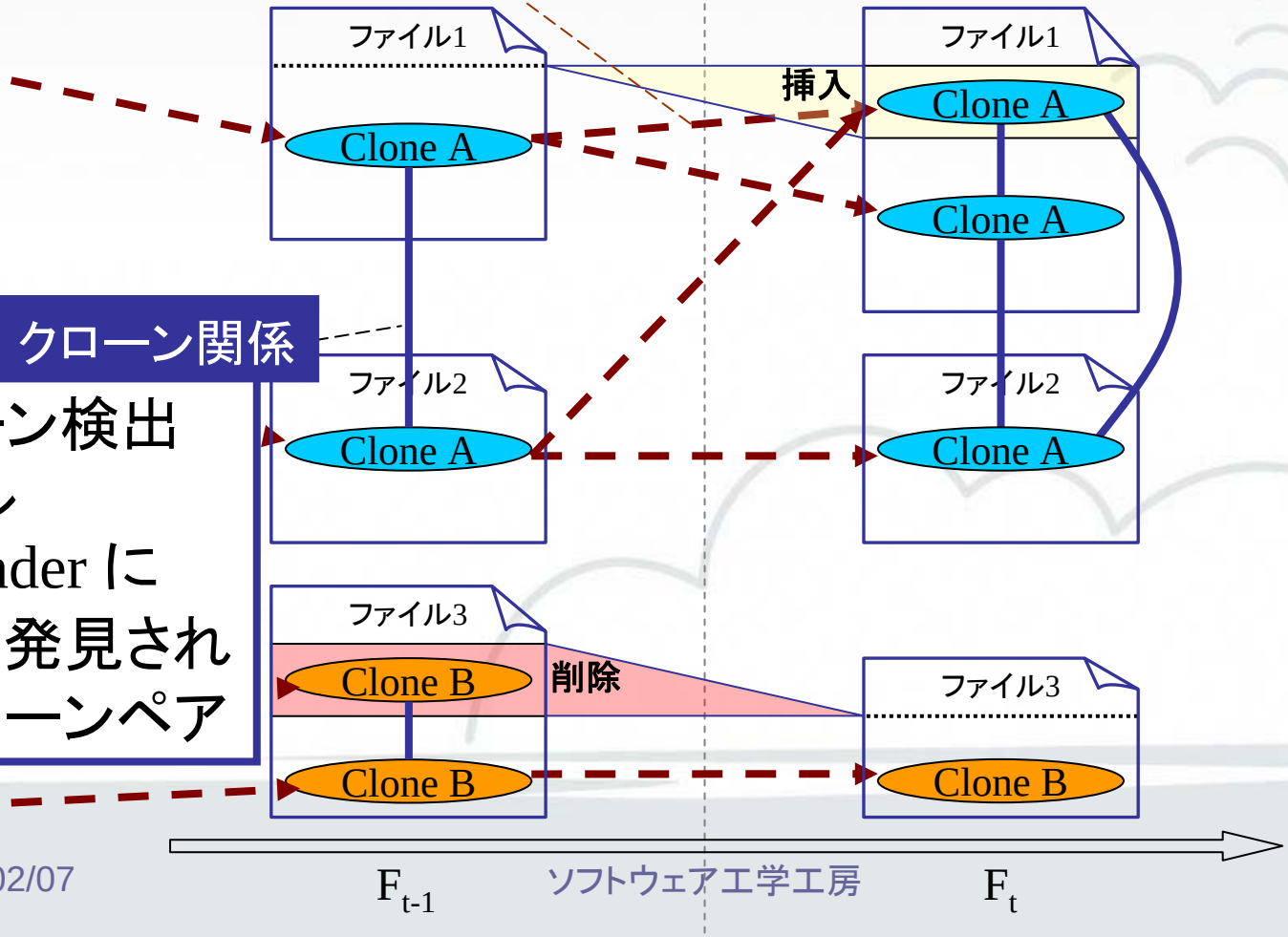
コードクローン履歴とは?

- 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

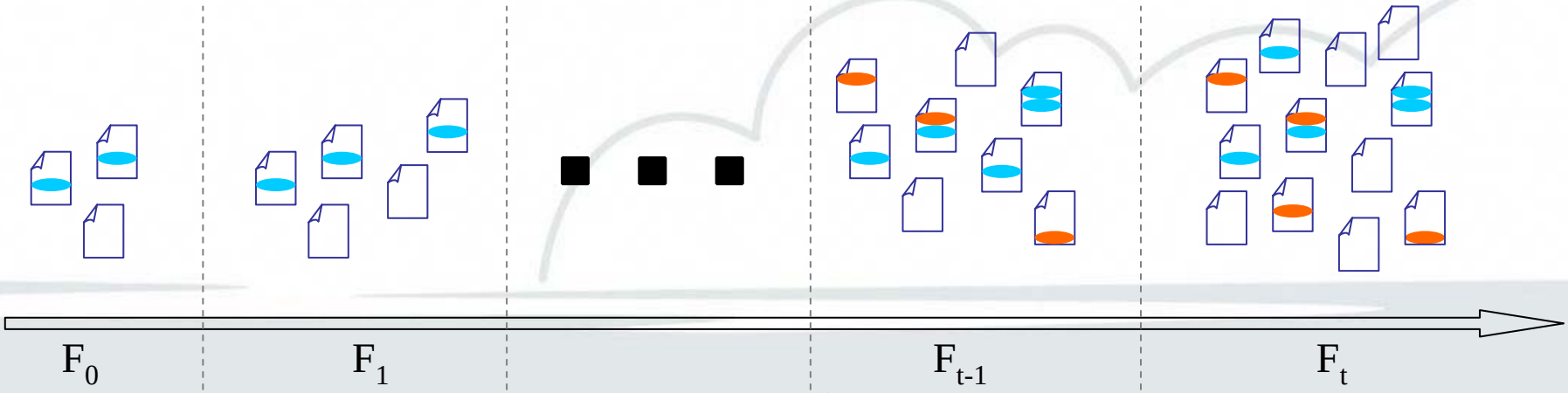


クローン関係

クローン検出ツール CCFinder によって発見されたクローンペア

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プロダクトを取得
- 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

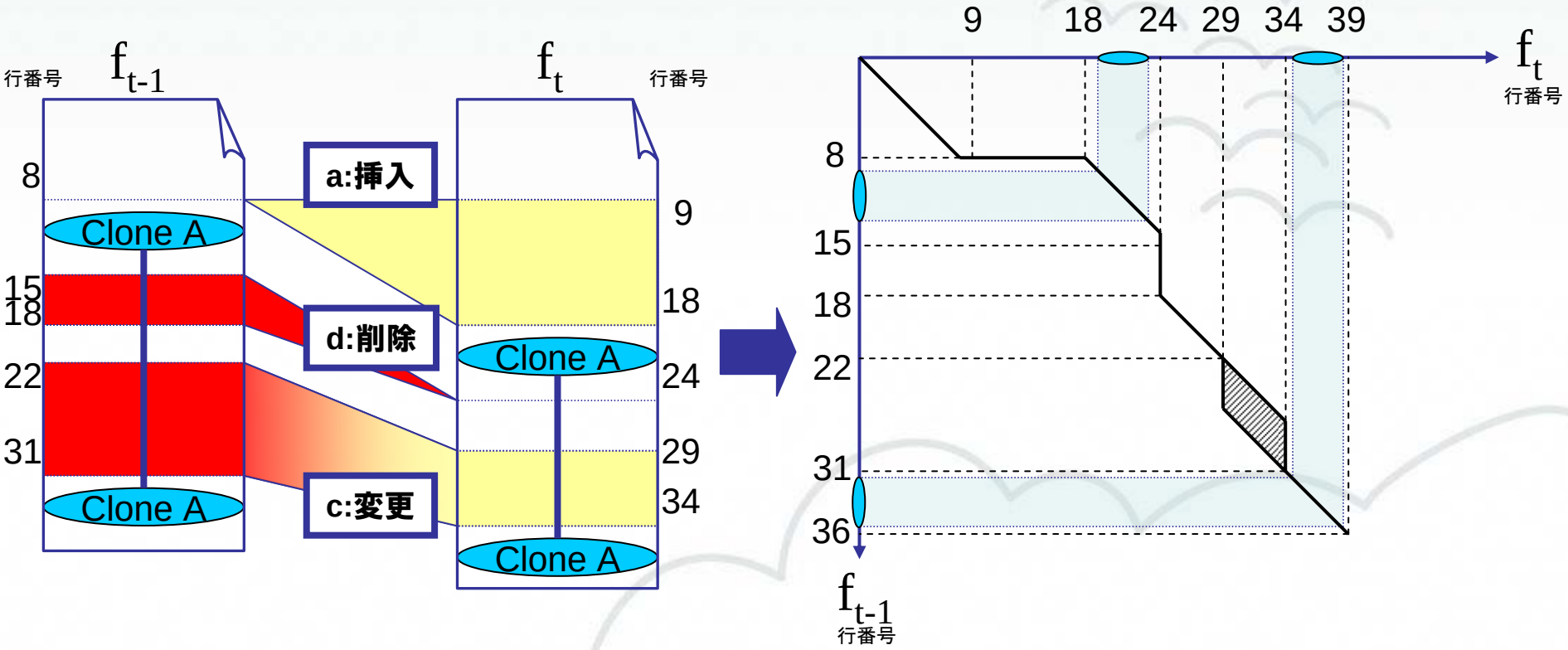


クローン履歴抽出手法(2/2)

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

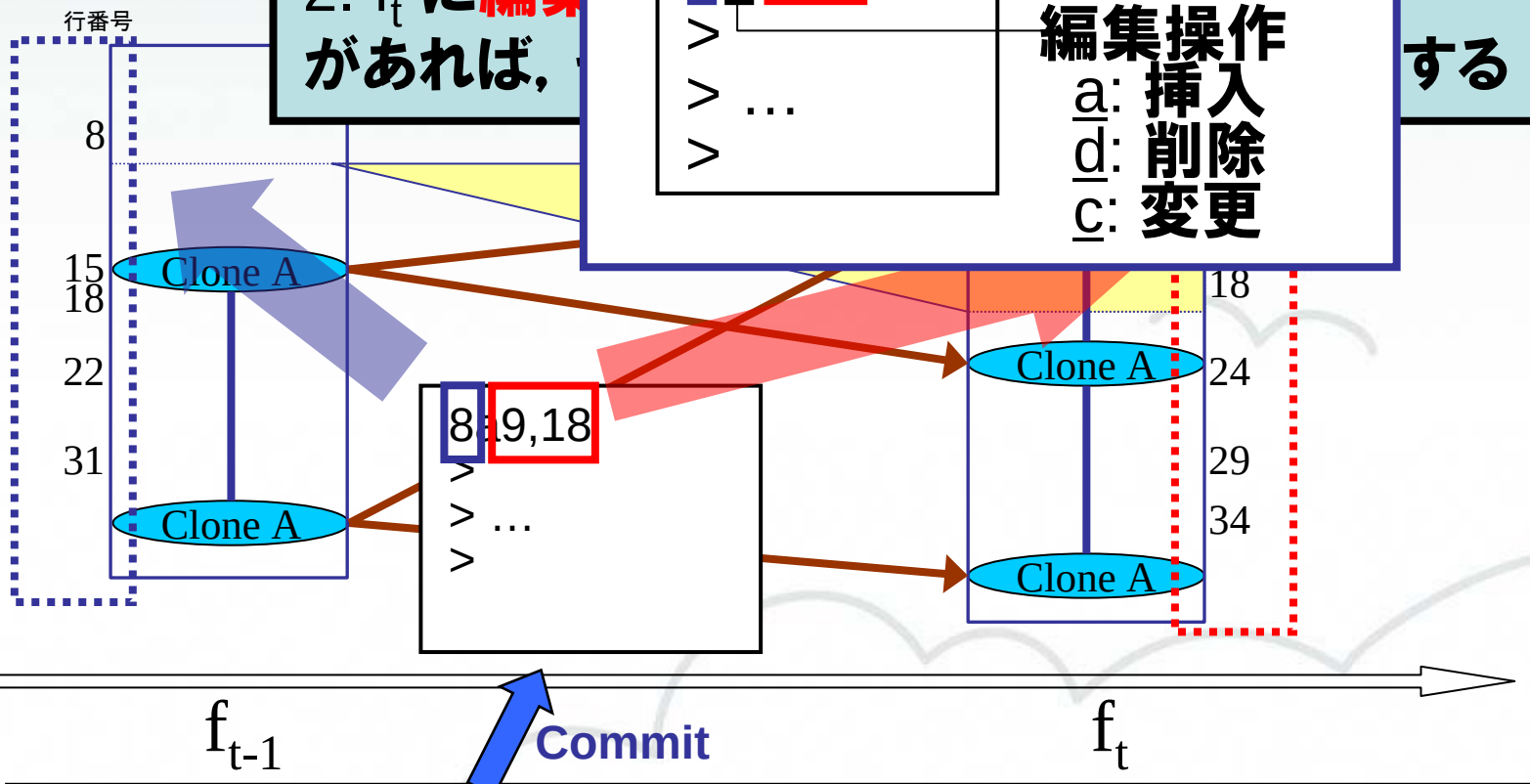
行番号の調整



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**

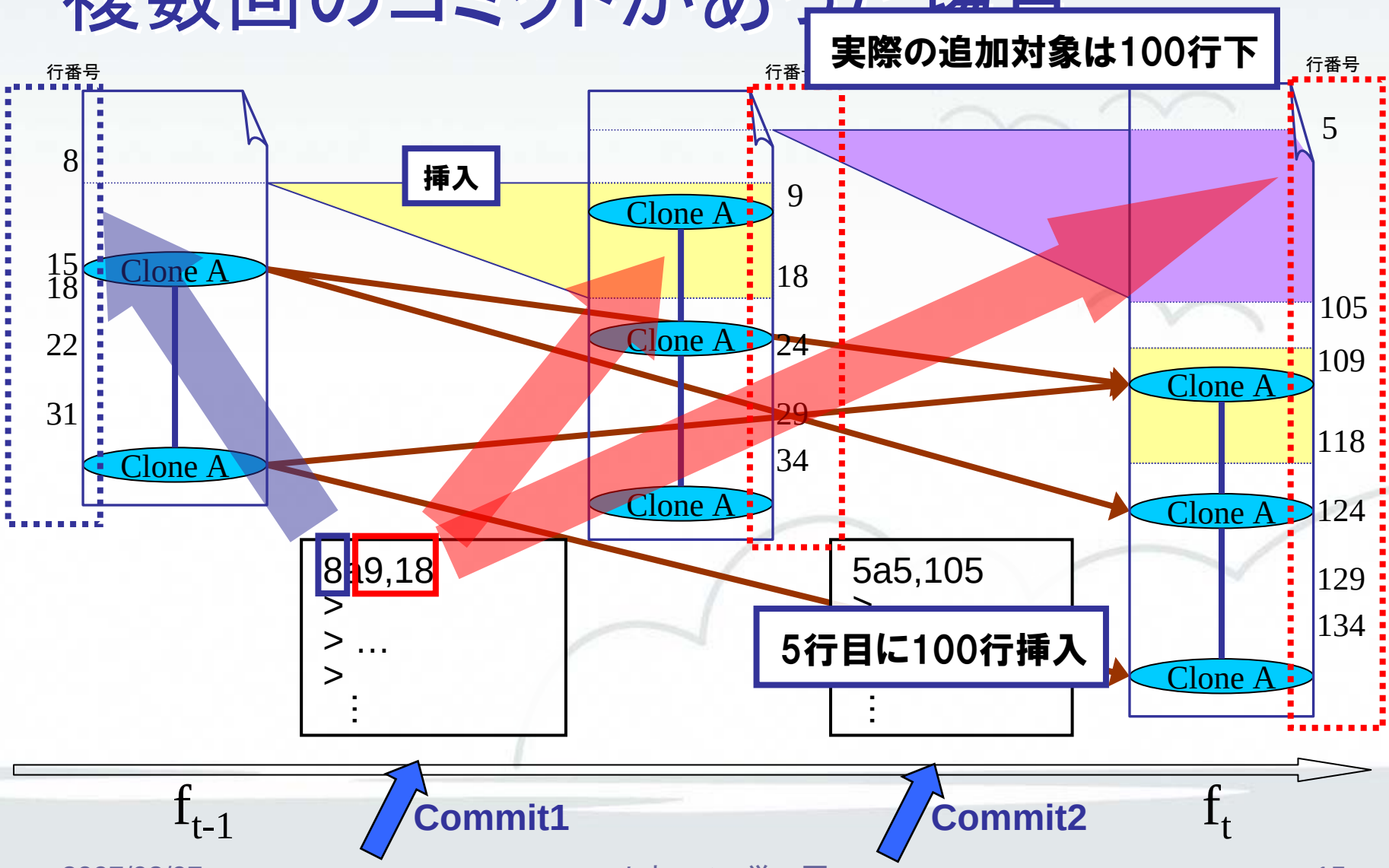
クローン履

版管理シ…いて
1. f_{t-1} に編…
2. f_t に編集…
があれば，…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

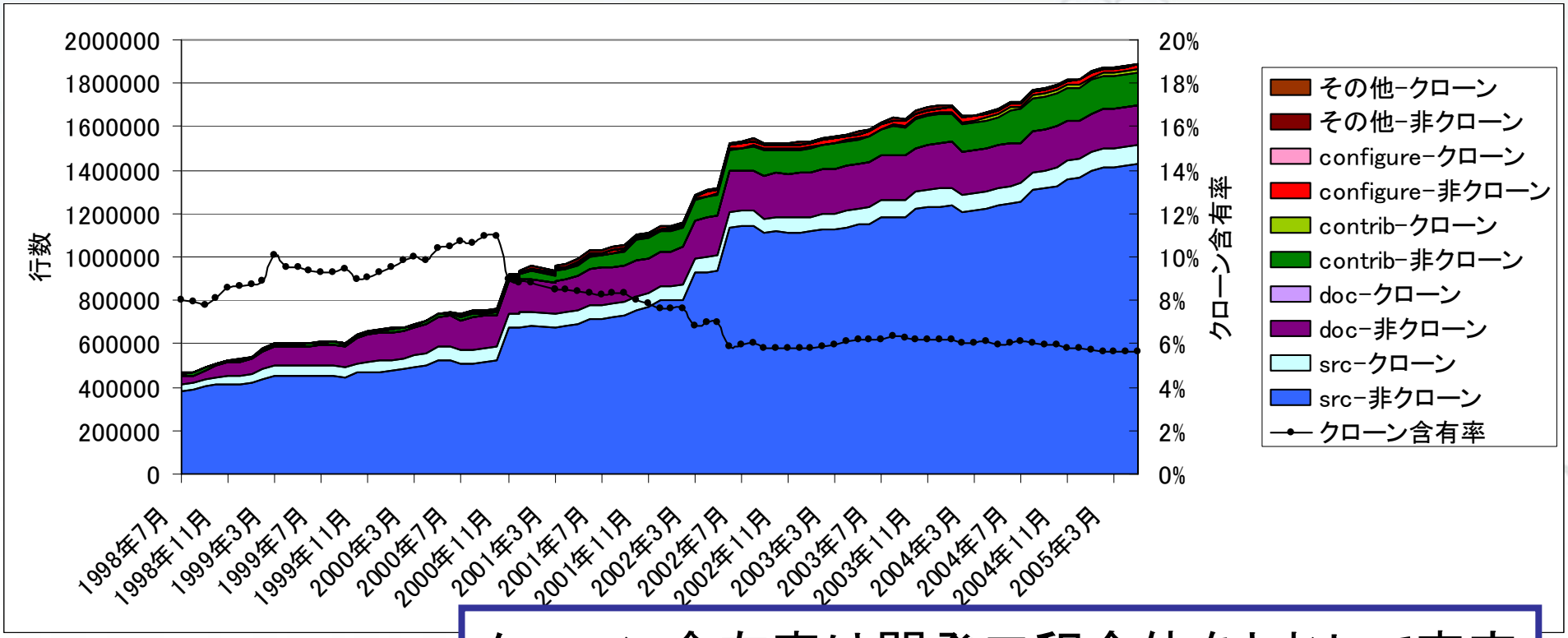
分析事例

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



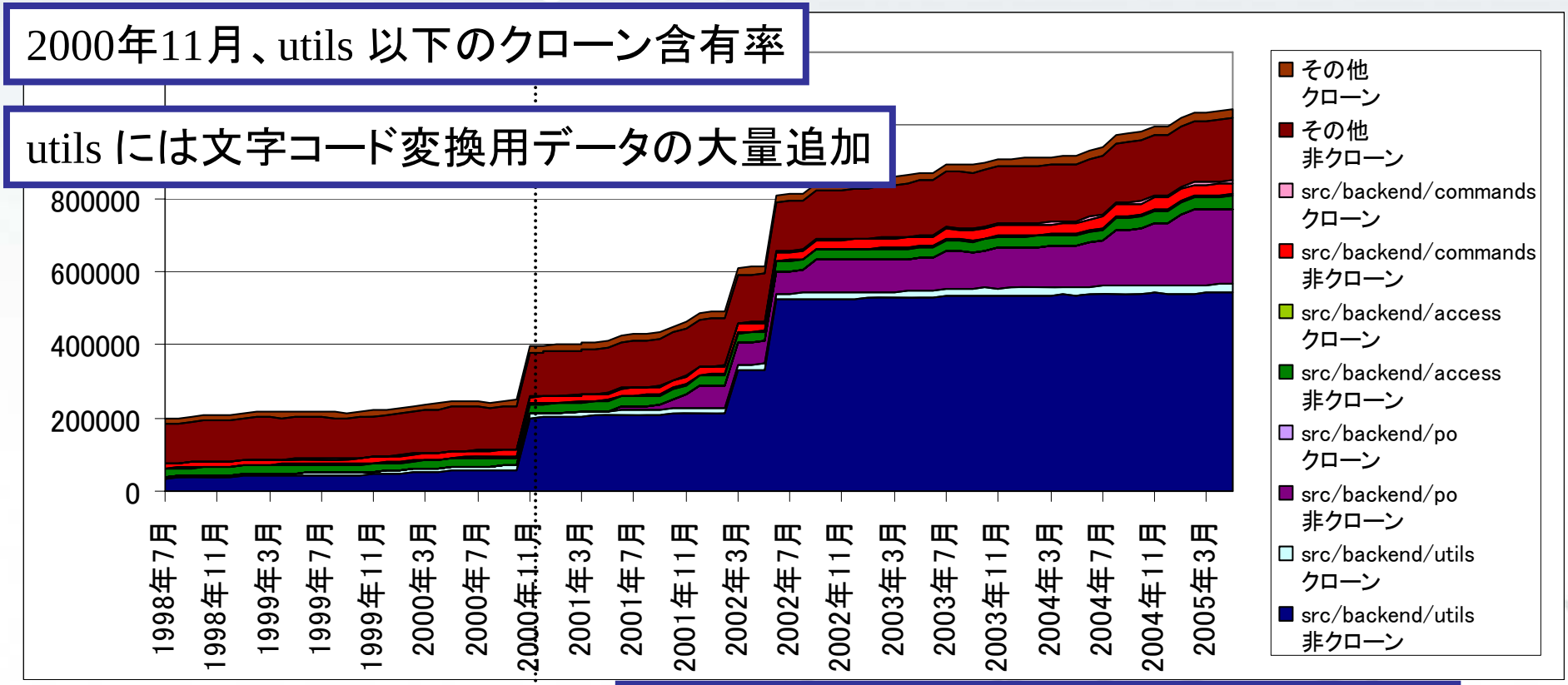
クローン含有率は開発工程全体をとおして安定

src 以下が全体の8割を占めている

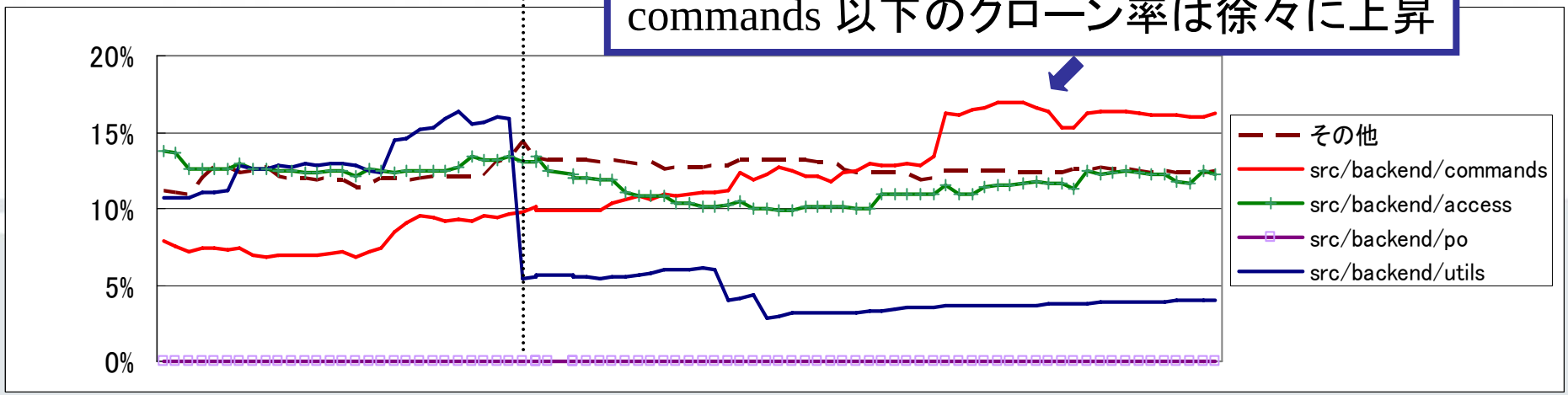
分析1: PostgreSQL コア部分のコード量

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

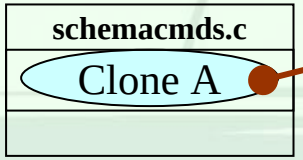


pgsql/src/backend/commands/dbcommands.c 07/01

```
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 }
```

pgsql/src/backend/commands/dbcommands.c 08/04

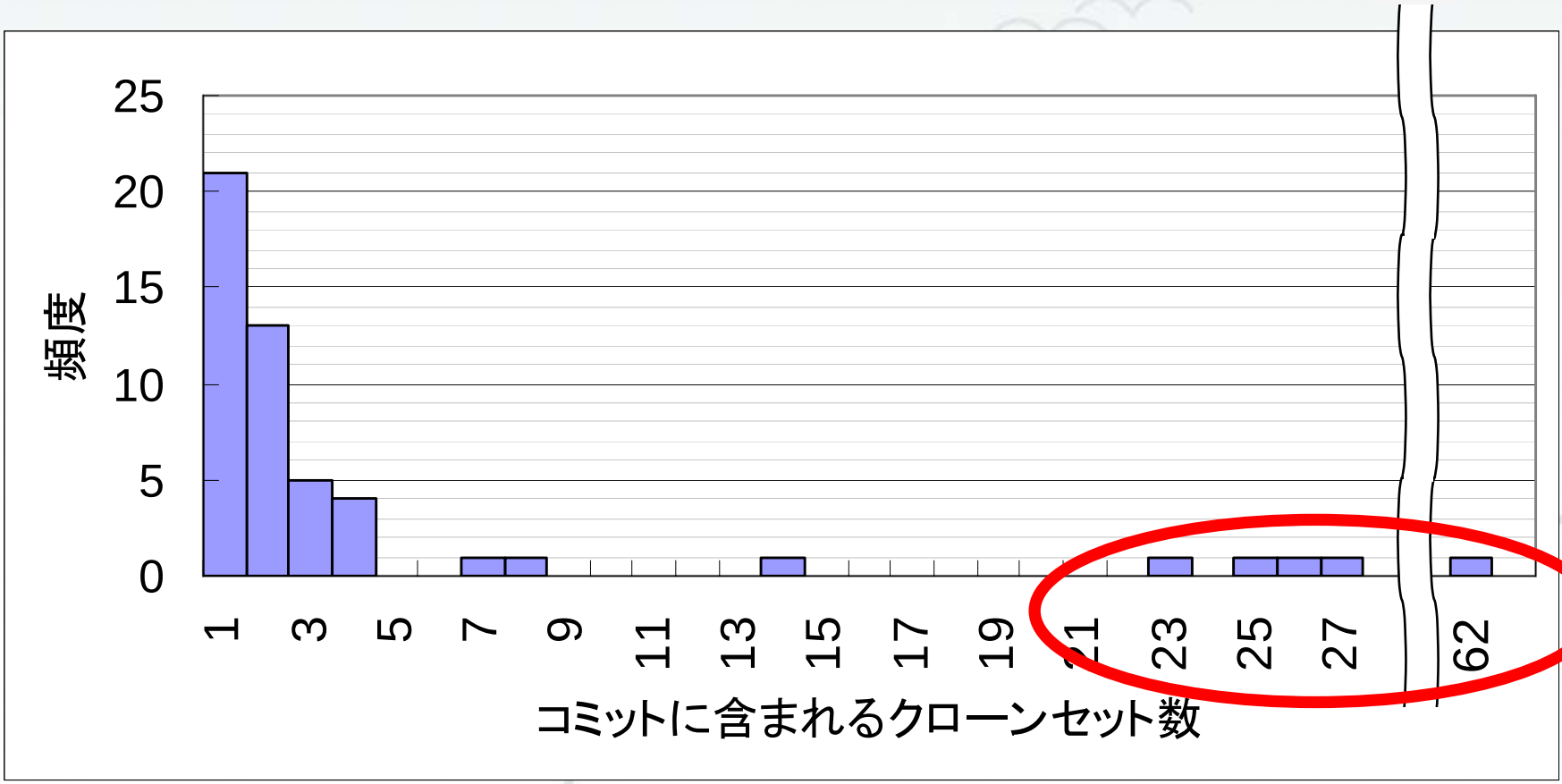
```
765 /*
766  * ALTER DATABASE name OWNER TO newowner
767  */
768 void
769 AlterDatabaseOwner(const char *dbname, AclId newOwnerSysId)
770 {
771         HeapTuple       tuple;
772         Relation        rel;
...
792         /*
793          * If the new owner is the same as the existing owner, consider 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),
...
342         }
343
344         heap_close(rel, NoLock);
345         heap_freetuple(tup);
346 }
```



2004/07/01

08/04

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
- 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

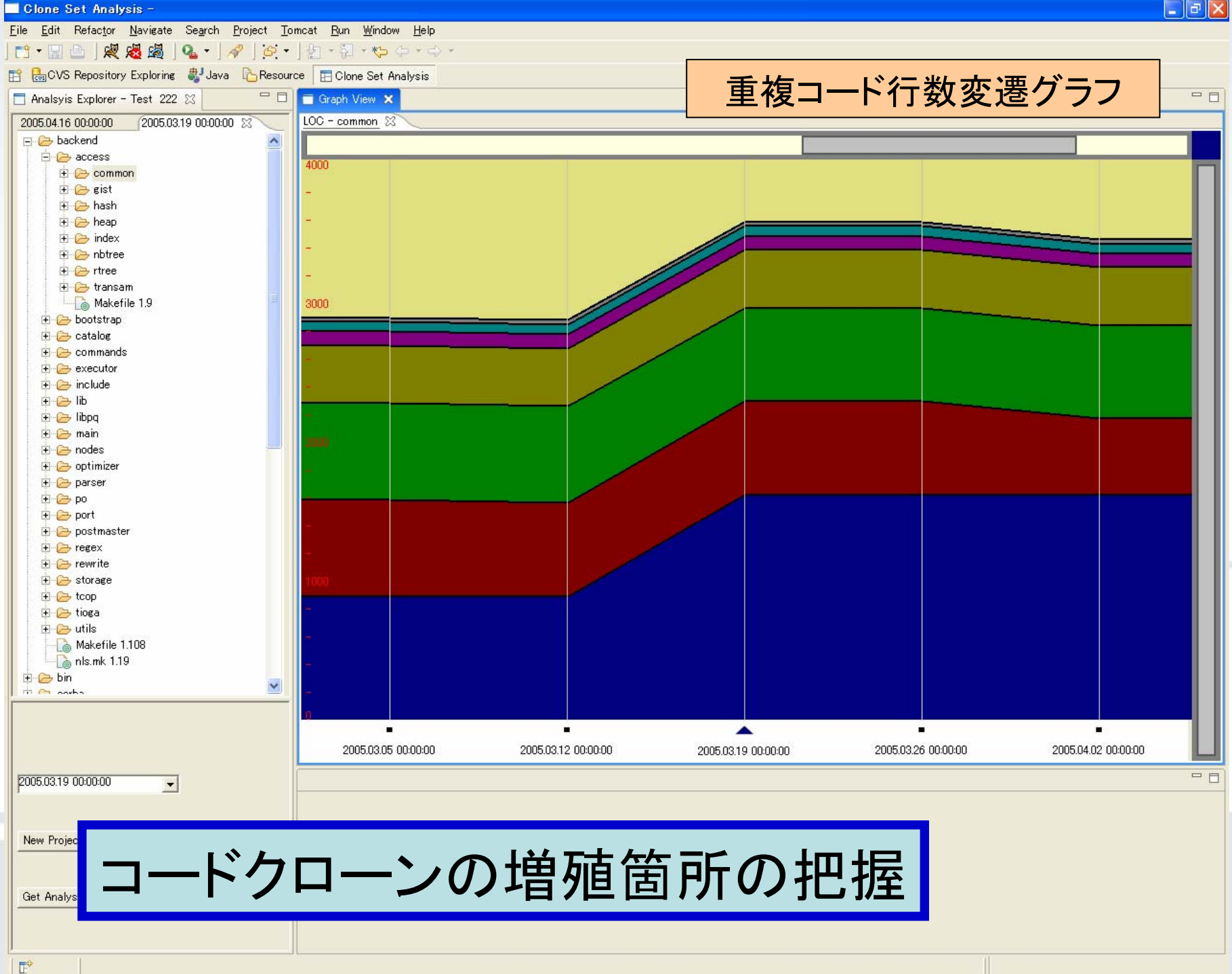
今後の展開

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

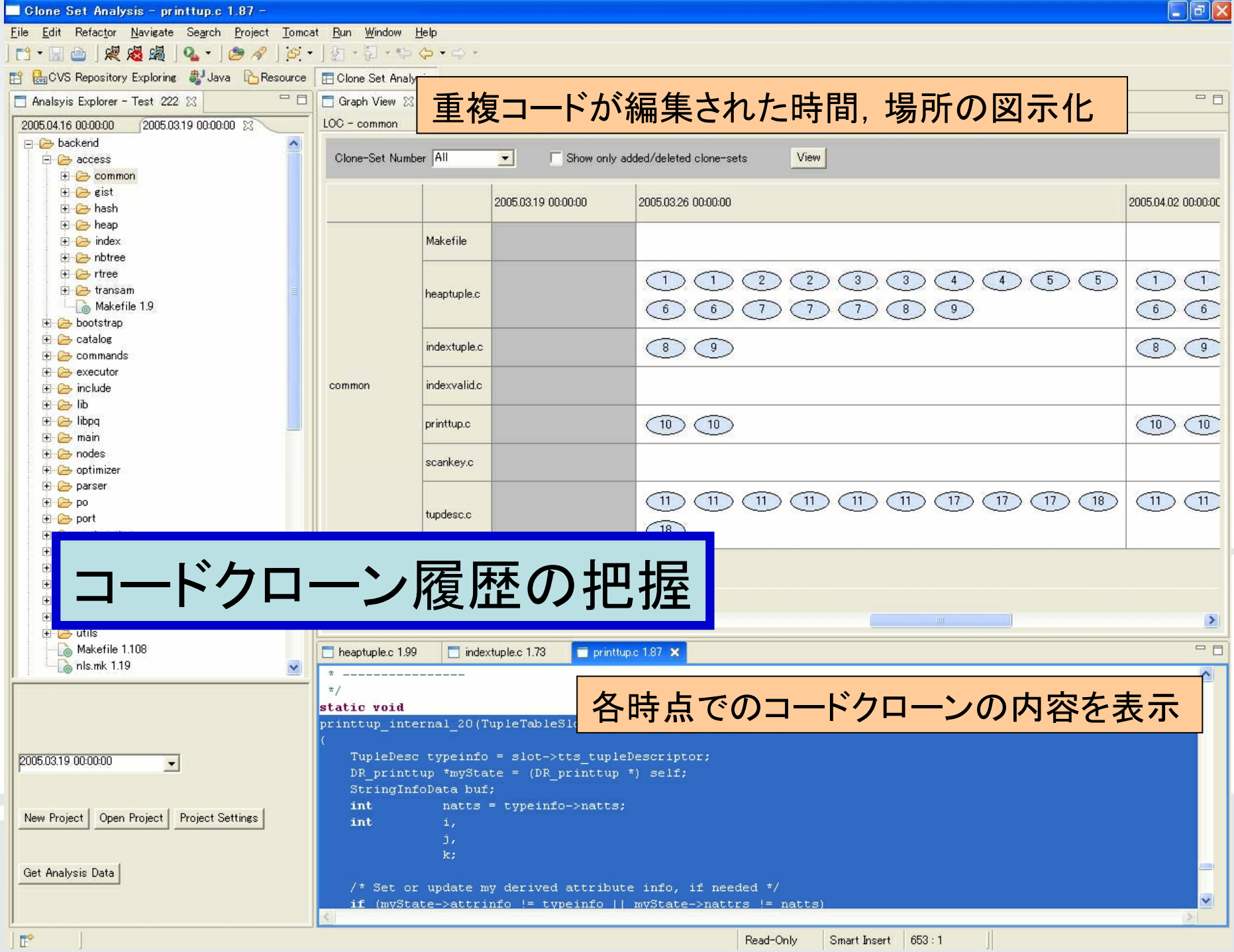
クローン履歴閲覧システム

- 対象
  - CCFinder が対応する言語
- 動作環境
  - Eclipse 上で動作（Eclipse プラグイン）
- 主な機能
  - クローン行数の変遷グラフ
  - クローンヒストリーグラフ
  - 各時点でのクローン内容を表示
  - （どの開発者が編集したかの表示）
  - （開発者ごとのクローン追加・編集・削除回数等のメトリクス表示）



重複コード行数変遷グラフ

コードクローンの増殖箇所の把握


Clone Set Analysis - printtup.c 1.87 -
File Edit Refactor Navigate Search Project Tomcat Run Window Help
CVS Repository Exploring
Java
Resource
Clone Set Analy
Analsyis Explorer - Test 222
2005.04.16 00:00:00
2005.03.19 00:00:00
backend
access
common
gist
hash
heap
index
nbtree
rtree
transam
Makefile 1.9
bootstrap
catalog
commands
executor
include
lib
libpq
main
nodes
optimizer
parser
po
port
utils
Makefile 1.108
nls.mk 1.19
2005.03.19 00:00:00
New Project
Open Project
Project Settings
Get Analysis Data
Graph View
LOC - common
重複コードが編集された時間，場所の図示化
Clone-Set Number
All
Show only added/deleted clone-sets
View
2005.03.19 00:00:00
2005.03.26 00:00:00
2005.04.02 00:00:00
common
Makefile
heaptuple.c
indextuple.c
indexvalid.c
printtup.c
scankey.c
tupdesc.c
コードクローン履歴の把握
heaptuple.c 1.99
indextuple.c 1.73
printtup.c 1.87
各時点でのコードクローンの内容を表示
static void
printtup_internal_20(TupleTableSl
{
TupleDesc typeinfo = slot->tts_tupleDescriptor;
DR_printtup *myState = (DR_printtup *) self;
StringInfoData buf;
int natts = typeinfo->natts;
int i,
j,
k;
/* Set or update my derived attribute info, if needed */
if (myState->attrinfo != typeinfo || myState->nattrs != natts)
Read-Only
Smart Insert
653 : 1