ソフトウェア工学III

プロダクトデータの解析I
——コードクローン——

ソフトウェア工学講座
門田暁人
akito-m@is.naist.jp
B303室，内線5311

ソフトウェア開発に関するデータ

- プロジェクト特性データ
  - プロジェクト種別，利用局面，システム特性，ユーザ要求管理，要員スキル，システム規模，工数，品質など
    →将来の開発の工数予測や計画立案などに役立つ．
- プロダクトデータ
  - 提案依頼書，要求仕様書，概要設計書，詳細設計書，ソースコード，テストケースなど
    →現在の開発・保守や次期バージョン開発に役立つ．
- プロセスデータ，資源データ
  - ガントチャート（作業計画・実績管理表），WBS（作業分解図），設計レビュー報告書，障害管理票
    →プロジェクトマネージメントに役立つ．

ソースコードの解析・計測の目的

- ソースコードを理解する.
  - 機能拡張, 再利用, リファクタリングなどのために
- 影響波及解析をする.
  - デバッグ, テスト, 保守などのために
- ソースコードの良し悪しを判断する.
  - 保守性, 移植性, テスト容易性などを評価する.
- Fault-proneモジュールを判別する.

など

プロダクトの品質特性

ISO/IEC 9126-1, JIS X 0129-1における品質の定義

- 内部・外部特性
  - 機能性
  - 信頼性
  - 使用性
  - 効率性

  （機能性～効率性）主にテスト, レビューにより評価される.

  - 保守性
  - 移植性

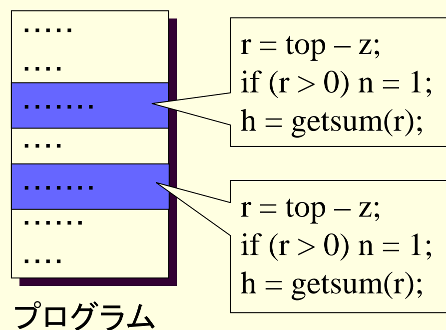
  （保守性・移植性）プロダクトの解析・計測が必要
- 利用時品質
  - 有効性, 生産性, 安全性, 満足性

プログラム解析技術

- プログラムスライシング
  - プログラム内の任意の文の着目する変数に影響を与える一部のコードのみを抽出する技術
  - デバッグ，テスト，再利用，理解支援，リファクタリングなどに用いられる.
- コードクローン解析
  - プログラム中の重複するコード対を特定する技術
  - デバッグ，テスト，リファクタリングなどに用いられる.
- 複雑さメトリクス計測
  - プログラムの特定の要素に着目し，複雑さを計測する.
    - 例：ループの深さ，変数の個数，継承の深さなど
  - Fault-proneモジュール判別やリファクタリングに用いられる.

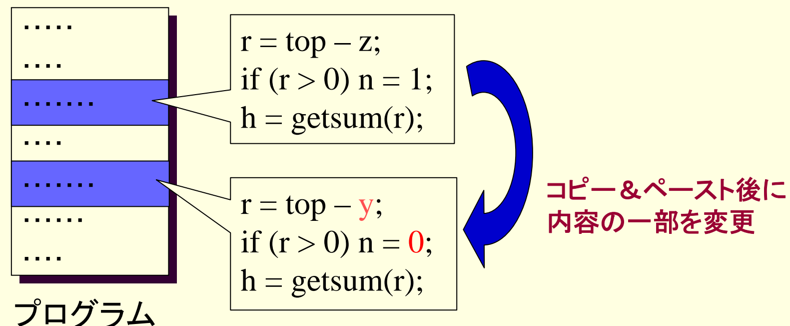
コードクローンとは

ソースコード中の類似したコード片



コードクローンとは

ソースコード中の類似したコード片



プログラム

クローンが生じる理由：

コピー＆ペースト，定型処理，意図的な繰り返し，偶然，プログラミング言語に適切な機能がないため，など

コードクローンの例

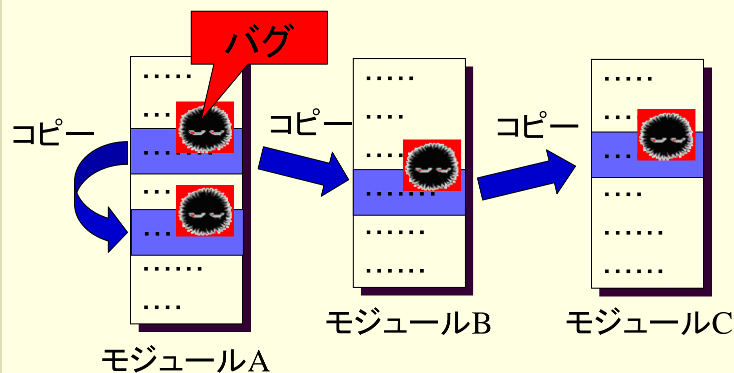
- プログラミング演習での例

```
 . . .
tmp = state[xcounter1+2][0][4];
     state[xcounter1+2][0][4] = state[xcounter1+2][0][3];
     state[xcounter1+2][0][3] = tmp;
. . .
 tmp = state[xcounter1+2][0][4];
     state[xcounter1+2][0][4] = state[xcounter1+2][0][5];
     state[xcounter1+2][0][5] = tmp;
. . .
 tmp = state[xcounter1+2][0][6];
     state[xcounter1+2][0][6] = state[xcounter1+2][0][7];
     state[xcounter1+2][0][7] = tmp;
. . .
 tmp = state[xcounter1+1][0][5];
     state[xcounter1+1][0][5] = state[xcounter1+1][0][8];
     state[xcounter1+1][0][8] = tmp;
```

ソフトウェア保守とコードクローン

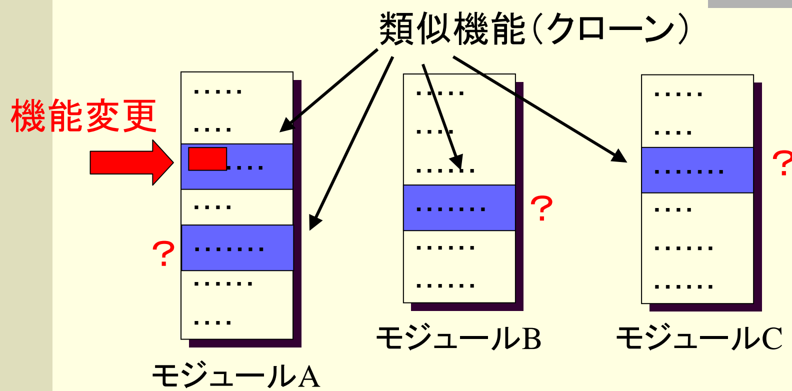
- コードクローンの問題
  - バグが入りやすい／バグを見つけにくい.
  - デバッグが難しくなる.
  - ソフトウェア保守が難しくなる.
- ソフトウェア保守
  - ソフトウェア出荷後の機能追加，拡張，修正作業
  - ソフトウェアの全ライフサイクルに必要とされるコストの50%が費やされることも珍しくない.
  - 効率化が求められる.

バグ修正時の問題



- 全てのバグを発見・除去するのに大きなコストがかかる.
  - 関連するクローンを全て探す.
  - 各クローン断片のバグの有無を調べる.
  - それぞれのバグを修正する.

機能変更時の問題



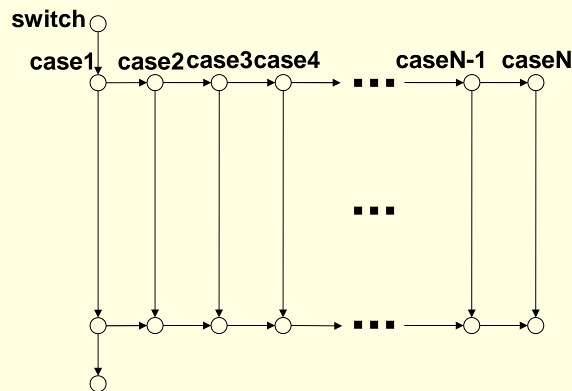
- 過不足なく変更するのに大きなコストがかかる.
  - 関連するクローンを全て探す.
  - 各クローン断片の変更の必要性の有無を調べる.
  - 各クローン断片に変更を加える.

なぜコードクローンが注目されるか

- 保守性との因果関係が明確である.
  - クローンが多い ⇒ 保守性が悪い
- 改善の手立てがある.
  - クローンを減らす ⇒ 保守性の改善
- 従来のソフトウェア評価尺度の問題
  - ソフトウェア複雑さ尺度
    - ループの数，分岐数，変数の数，関数の数．．．
  - 複雑さが大 ⇒ 保守性が悪い（？）
  - 改善の手立てがない.
    - if文や変数を減らせと言われても困る.

参考(サイクロマティック数)

制御フローの閉路の数



サイクロマティック数が大きかったとしても，改善できるとは限らない．

コードクローンに対する現場の意見・コメント

- 後のデバッグや保守のことを考えれば，開発時にコードクローンを生じさせるべきではない，というのは納得できる．
- コードクローンには良いものと悪いものがあるのではないか．
- 大規模になるとクローンを全く含まないソフトウェアを作るのは無理である．許容範囲はどのくらいか？
- コードクローンを減らそうとすると，再設計が必要になるなど，余計な開発工数がかかることも多い．保守とのトレードオフを考えるべきだろう．
- 保守しなくてよいソフトウェア(使い捨てのソフトウェア)の場合は，クローンがあってもよいだろう．
- クローンが既に存在する場合，どうやってクローンを減らせばよいか難しい．マクロ等を使って無理やり削除すると余計に保守がしづらくなる．

本講義の概要

- コードクローンの検出方法
- 分析事例:
  - 大規模なレガシーソフトウェア
  - 多数のオープンソースソフトウェア
  - FreeBSD, Linux, NetBSDの間のクローン
- プロジェクト進捗管理とコードクローン

コードクローンの種類

- 検出可能なクローンの種類
  - Exact Clone
    テキストとして比較したときに完全に一致している
  - Renamed Clone
    変数名や関数名が変化しているもの
  - Gapped Clone
    追加、削除によって部分的に変更されているもの
- 保守に役立てるためには, Renamed CloneもしくはGapped Cloneの検出が不可欠

コードクローンの検出

- マッチングアルゴリズム
  - 単純にやるとO(n^2)かかる．数百万行規模のプログラムではつらい．
  - Suffix Treeマッチングアルゴリズムにより，O(n)でクローンを検出できる．
- 解析の深さ
  - ソースコード ＝ テキストファイルとみなす．
  - 字句解析を行う．
  - 構文解析まで行う．
- 言語依存性
  - 言語に特有の定型処理はクローンとみなしたくない．

クローン検出ツール CCFinder

- 作者：神谷年洋（産業技術総合研究所）
- Webページ： http://www.ccfinder.net/
- 特徴：
  - ソースコードを直接比較することによりクローンを検出する．
  - プログラミング言語の構文を認識して精密に、効率よく
  - 変数名が書き換えられているクローンも検出
  - 現在、C/C++, Java, COBOLなどに対応
  - 大規模なソースコードを対象とする
    - 百万行規模のソースコードを実用時間で解析

コードクローン検出手順

(1)ソースコードを入力し、トークンの列にする

(2)文法知識を用いてトークン列を変形する

(3)Suffix-treeアルゴリズムを用いてクローンを検出する

(4)クローンの位置(ファイル、行)を出力する

クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける

```
AFG::AFG(JaObject* obj) {
  objname = “afg";
  object = obj;
}
AFG::~AFG() {
  for(unsigned int i = 0; i < children.size(); i++)
    if(children[i] != NULL)
      delete children[i];
  for(unsigned int i = 0; i < nodes.size(); i++)
    if(nodes[i] != NULL)
      delete nodes[i];
}
```

クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける

```
AFG :: AFG ( JaObject * obj ) {
objname = "afg" ;
object = obj ;
}
AFG :: ~ AFG ( ) {
for ( unsigned int i = 0 ; i < children . size ( ) ; i ++ )
if ( children [ i ] != NULL )
delete children [ i ] ;
for ( unsigned int i = 0 ; i < node . size ( ) ; i ++ )
if ( node [ i ] != NULL )
delete node [ i ] ;
}
```

クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける

```
$ :: $ ( $ * $ ) {
$ = $ ;
$ = $ ;
}
$ :: ~ $ ( ) {
for ( unsigned int $ = $ ; $ < $ . $ ( ) ; $ ++ )
if ( $ [ $ ] != $ )
delete $ [ $ ] ;
for ( unsigned int $ = $ ; $ < $ . $ ( ) ; $ ++ )
if ( $ [ $ ] != $ )
delete $ [ $ ] ;
}
```

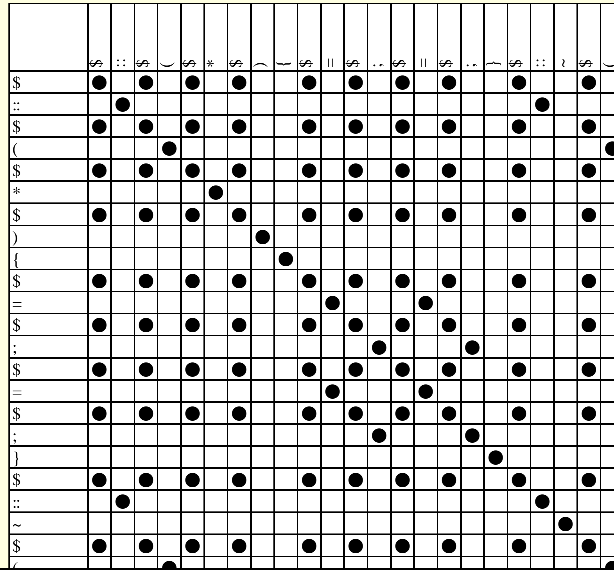
クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける



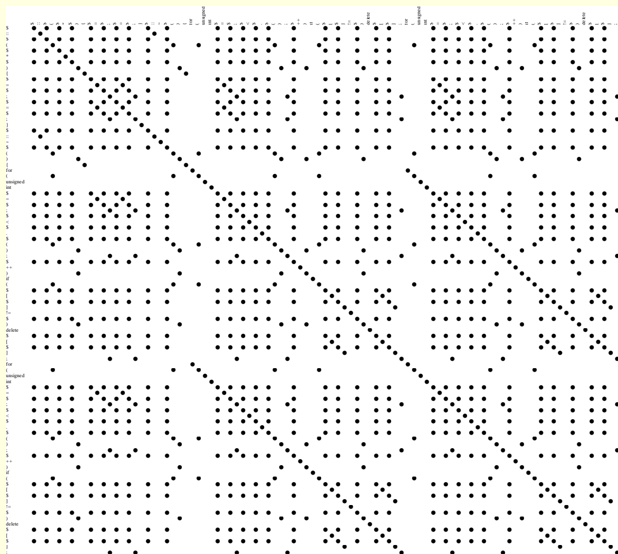
クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける



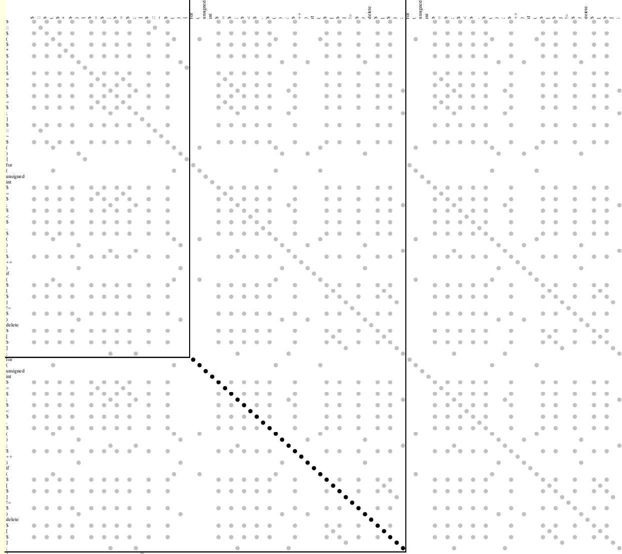
クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける



クローン検出手順 – (例)

(1)トークンに切り分ける

(2)すべてのユーザー定義名を同一トークンに変更する

(3)一致するトークンを見つける

(4)一致列を見つける

```
AFG::AFG(JaObject* obj) {
  objname = “afg";
  object = obj;
}
AFG::~AFG() {
  for(unsigned int i = 0; i < children.size(); i++)
    if(children[i] != NULL)
      delete children[i];
  for(unsigned int i = 0; i < nodes.size(); i++)
    if(nodes[i] != NULL)
      delete nodes[i];
}
```

実際のツールでは

- 構文認識・変形ルール
  - ユーザー定義名の置き換え
  - テーブル初期化部分を取り除く
  - 名前空間のプリフィックスを取り除く
  - 文や関数の区切りを認識する
- 最適化
  - 行列ではなくSuffix-treeアルゴリズムを利用する
  - 枝狩りする.

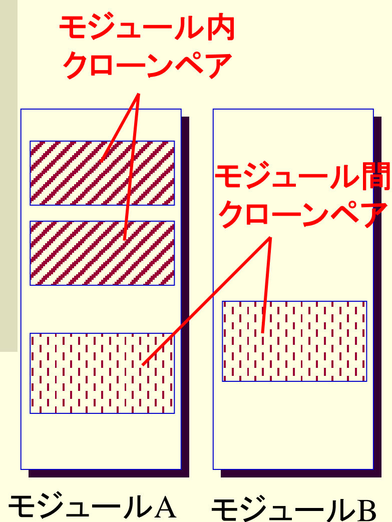
統合コードクローン分析環境

- ICCA ー Integrated Code Clone Analyzer
  http://sel.ist.osaka-u.ac.jp/icca/
  - 大阪大学大学院 情報科学研究科 井上研究室にて
    開発が続けられている.
  - ソースコード中のコードクローンをグラフィカルに解析する.
  - エンジン部分にCCFinderを使用している.
  - 3つのコンポーネント
    - Gemini. . . 汎用可視化ツール
    - Aries. . . リファクタリング向けツール
    - Libra. . . デバッグ用ツール

事例1：レガシーソフトウェア

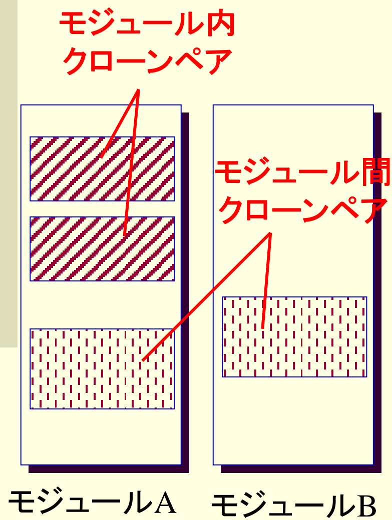
- 1977年に開発されたソフトウェアが対象
  - ある組織の業務基盤システムである．
  - 20年以上にわたって保守が続けられている．
  - 構造化COBOL言語で記述されている．
  - 約100万行，約2000モジュールで構成されている．
- 30行以上一致するコード対をクローンとして検出した．
- モジュール（ファイル）ごとに，クローン含有率，最大クローン長などの値を算出した．

定義：2種類のクローンペア



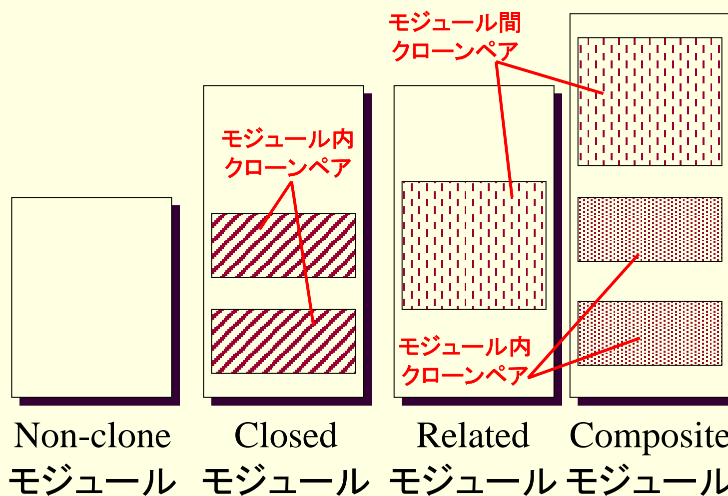
- モジュール内クローンペア
  - ペアを構成する2つのコード列が同一のモジュールに存在する．
- モジュール間クローンペア
  - ペアを構成する2つのコード列が異なるモジュールに存在する．

定義：2種類のクローンペア



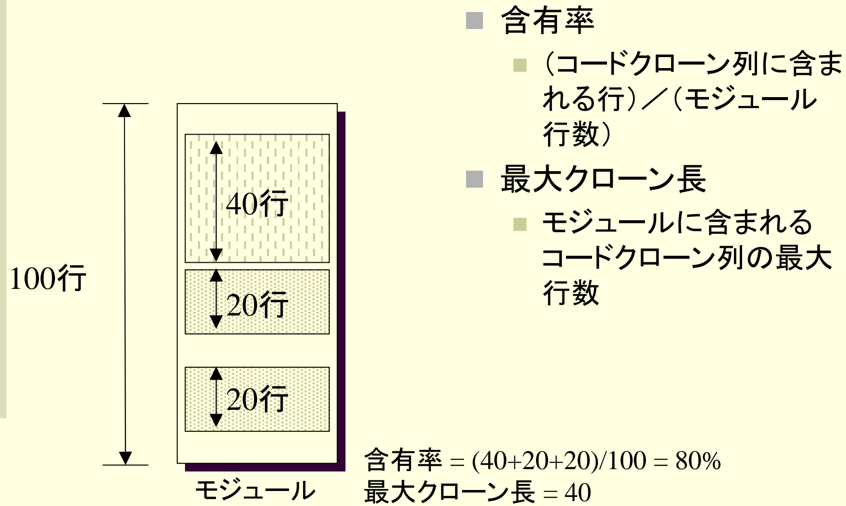
- いずれのクローンも，バグ混入の原因となり得る．
- モジュール間クローンは，モジュール間結合度を強める（独立性を低める）原因となる．
  - モジュール設計の原則＝モジュール間結合度を小さくし，モジュール凝集度を高めるべし

定義：4種類のモジュール

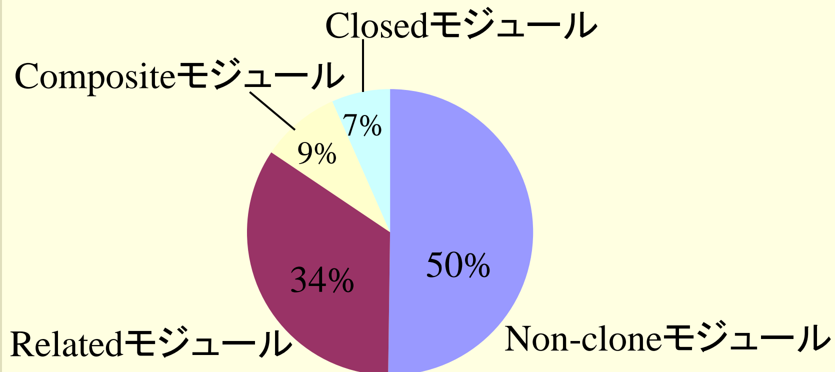


定義：2つのメトリクス

- 含有率
  - （コードクローン列に含まれる行）／（モジュール行数）
- 最大クローン長
  - モジュールに含まれるコードクローン列の最大行数

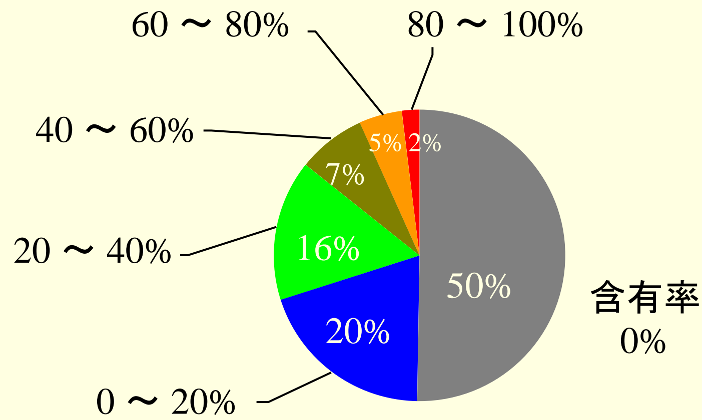


モジュールの分類



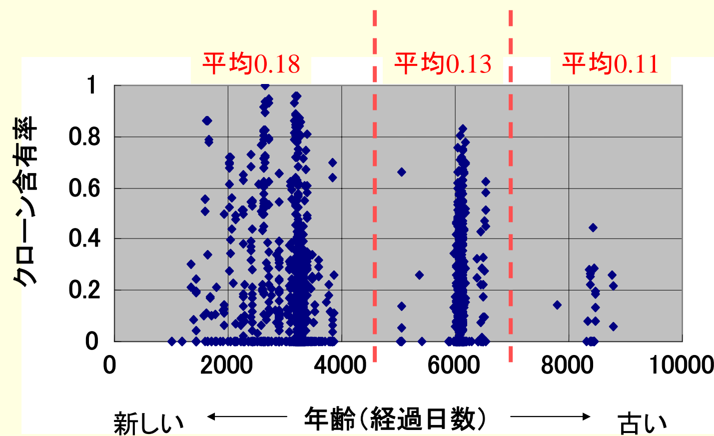
- コードクローン（30行以上一致するコード対）を含むモジュールは，全体の約半分
- モジュール間クローンが多い．

クローン含有率



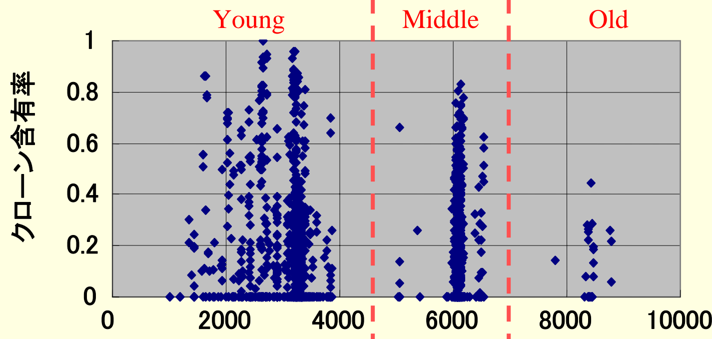
・クローン含有率が80%を超えるモジュールが全体の２%（約40個）存在した.

クローン含有率とモジュール年齢の関係



年齢が約10年以下の新しいモジュールにクローンコードが多く含まれている.

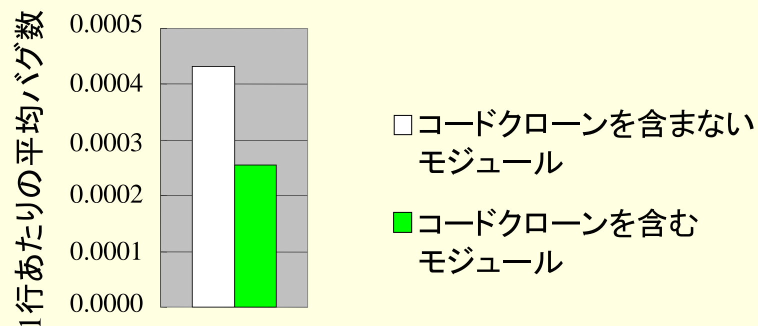
クローンコードとモジュール年代の関係



	Old (29)	Middle (802)	Young (1052)
Old	37.9% (11)	0.2% (2)	0.1% (2)
Middle	／	36.7% (295)	4.1% (77)
Young	／	／	44.7% (471)

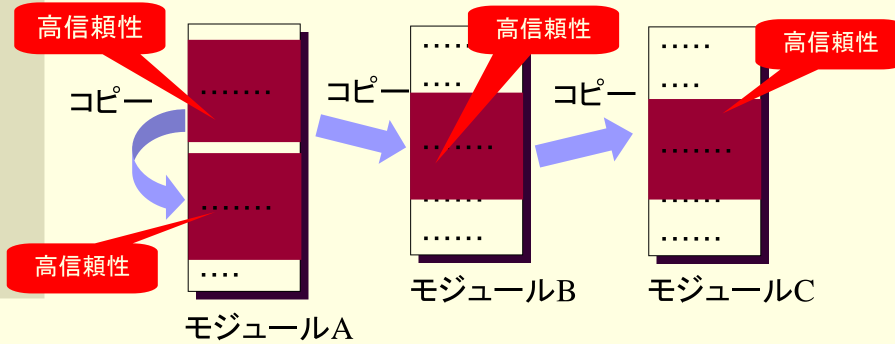
年代にまたがるクローンコードは少ない

クローンの有無と発見バグ数



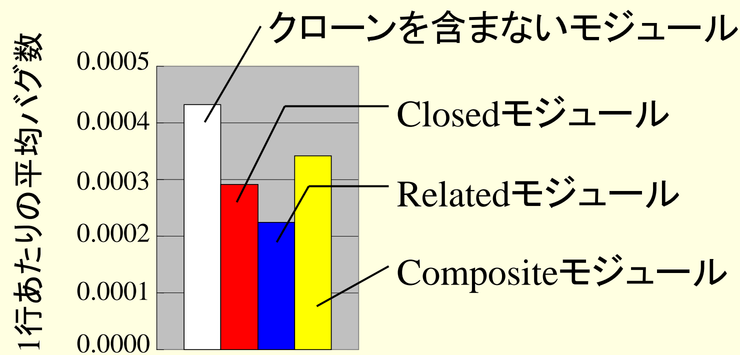
コードクローンを含むモジュールの方が約40%
バグが少ない.

考えられる解釈



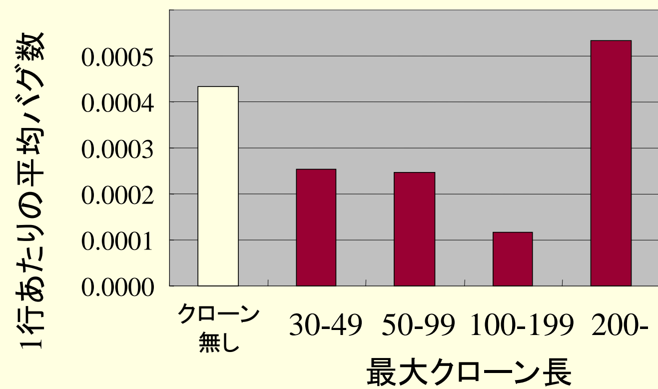
コードが水増しされている分，1行あたりのバグ数は減っている.

モジュール種別と発見バグ数



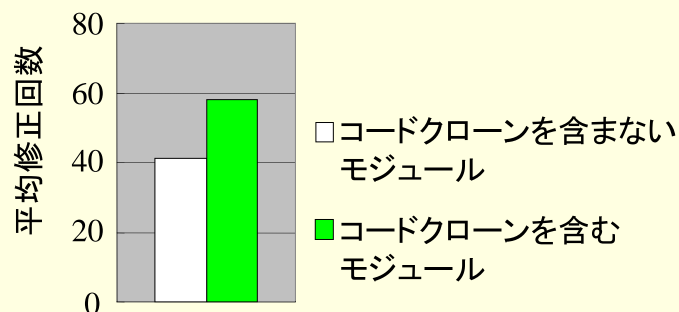
Relatedのバグが少ないのは，1個のモジュールを丸ごとコピーして作られたモジュールを含むためと思われる.
Compositeの値が高いのは興味深い.

最大クローン長と発見バグ数



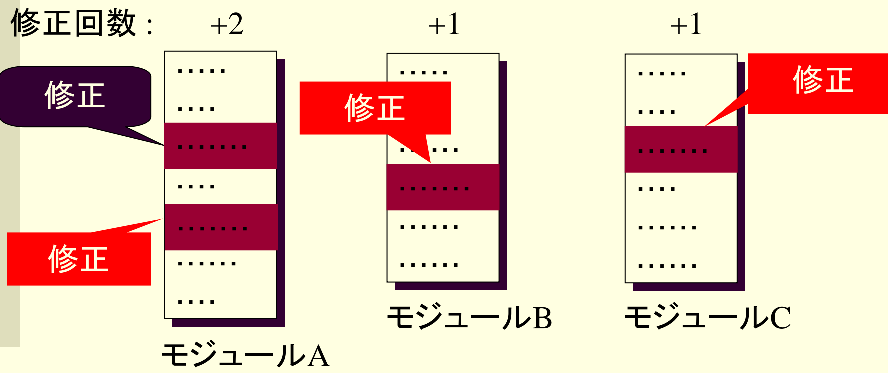
200行以上の巨大なクローンを含むモジュールは，バグ数が多くなっている.

クローンの有無と修正回数



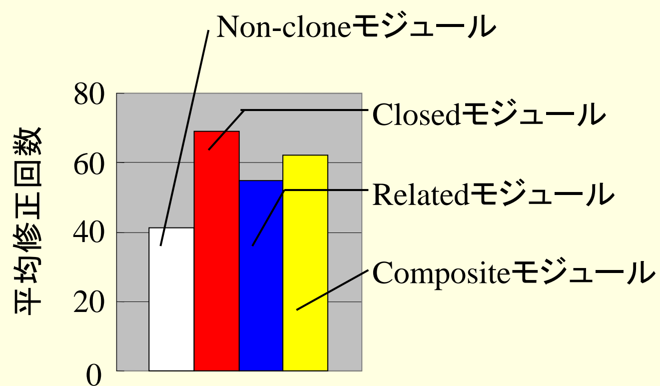
コードクローンを含むモジュールの方が約40%
変更回数が多い.

考えられる解釈



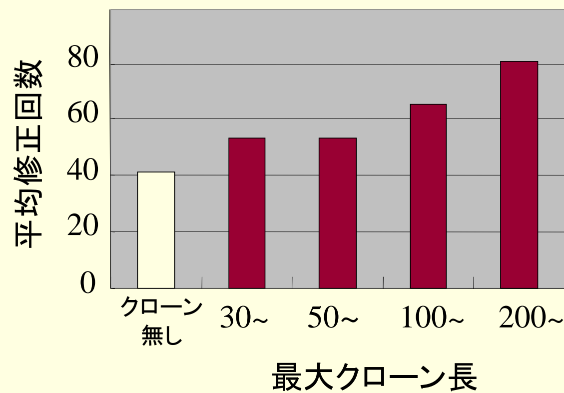
コードクローンがあると余分な修正コストがかかることがうかがえる.

モジュール種別と修正回数



モジュール内クローンがあると，後で修正が入りやすい？

最大クローン長と修正回数



より大きなクローンを含むほど，修正回数が増える.

事例1のまとめ

- コードクローンがソフトウェアの信頼性を低下させるとはいえない.
  - むしろ，1行あたりのバグ数は減る.
  - ただし，ソフトウェア全体でみると(クローンを全く生成しなかった場合と比べて)バグが増えている可能性はある.（実験的に確かめることは難しい）
  - いずれにしても，巨大なクローン(200行以上)は信頼性を低下させるといえそうだ.
- コードクローンが修正回数を押し上げていることがうかがえた.
  - より大きなクローンを含むほど，修正回数が増えている.

保守作業者へのインタビュー結果

- Q:クローンを意図的に作ることはありますか.
  - A:あります. 機能追加を行う際に, 安全に変更できそうにない場合, コピー＆ペーストして作成したクローンに手を加えます. （元コードの信頼性は保たれます. ）
- Q: クローンをどう管理してますか.
  - A:「横並びチェック表」を使います. この表には, ソフトウェア中の関連のある部分が掲載されています. ソフトウェアに変更を加えた場合には, この表を見て, 関連のある部分が他にないかどうかをチェックします. （クローンもここでチェックされます. ）

事例2 : 125種類のオープンソースソフト

•GNUプロジェクトで公開されているソフトウェアからランダムに選択(115種類)

•オープンソースコミュニティにおける成功事例(10種類) †

- perl, emacs, cvs, httpd, apache, samba, sendmail, tcl/tk, python, linux-kernel

•全てC言語で書かれている

†青山幹夫 : オープンソースソフトウェアの現状, 情報処理, Vol.43, No.12, pp.1319-1324 (2002).

分析のねらい

- 統計的な分析を行う.
  クローン含有率の平均は？最小は？最大は？
- クローンの量的な標準値を探る.
  オープンソース＝保守しやすく書かれている？
- コードクローン生成の原因を調査する.
  本当に全てのクローンが保守性に悪影響を与えているのか？

コードクローン計測結果

50トークン（約21行）以上の一致列をコードクローンとして検出した.

	モジュール数	行数	CRate	inMCRate	interMCRate
平均値	85.96	55674.848	11.3%	9.1%	3.0%
中央値	22	11297	8.7%	5.9%	0.8%
最大値	3755	2678939	61.2%	61.2%	37.6%
最小値	1	478	0.0%	0.0%	0.0%
標準偏差	339.98	242694.56	11.46	10.11	5.66

- ソースコードの約10%がコードクローンである.
- ソースコードの公開を前提としたオープンソースソフトウェアであってもコードクローンを多く含むものが存在する.

成功事例のソフトウェア

perl, emacs, cvs, httpd, apache, samba, sendmail, tcl/tk, python, linux-kernel

	モジュール数	行数	CRate	inMCRAte	interMCRate
平均値	584	436,539	10.5%	6.0%	5.2%
中央値	174	207,443	10.5%	4.6%	3.6%
最大値	3,755	2,678,939	18.0%	13.6%	13.7%
最小値	115	100,342	4.2%	2.6%	1.3%
標準偏差	1120.241	791386.979	0.048	0.034	0.041

- 全体と比較して
  - CRate、inMCRateの平均値は低い
  - interMCRateは平均値は高い

クローン含有率の高いソフトウェア

プログラム	モジュール数	行数	CRate	inMCRate	interMCRate
superopt-2.5	1	3080	61.2%	61.2%	0.0%
chemtool-1.5	12	27254	51.6%	26.5%	37.6%
gwcc-0.9.8	19	9332	47.0%	47.0%	0.0%
bucob-0.1.2	43	44828	43.7%	9.9%	35.5%
xcdroast-0.98	17	36193	42.2%	39.4%	15.5%
rubrica-0.9.1	57	30542	35.3%	34.1%	10.7%
gaby-2.0.2	106	60652	33.3%	27.4%	15.9%
galculator-0.8	8	2999	32.5%	32.5%	0.0%
pitchtune-0.0.:	10	4373	31.9%	31.9%	0.0%
hme-1.3.1	17	7211	30.9%	26.6%	7.4%

- ソースコードの30%以上がコードクローンである
- 発生要因
  (1)コピー＆ペーストによるコード、(2)プラットフォーム依存コード、(3)自動生成ツールによるコード、(4)GUIに関するコード

要因1：コピー＆ペースト

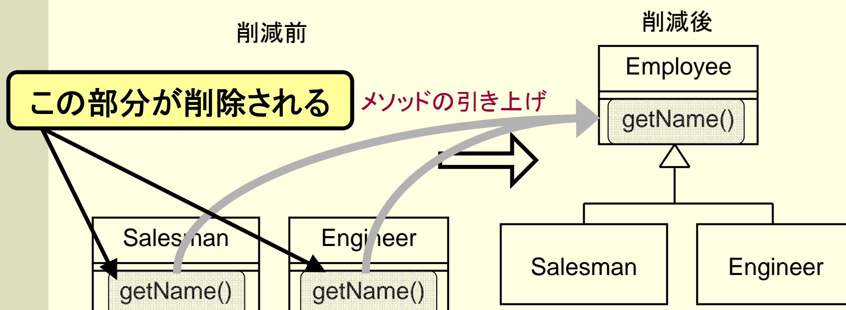
HME：

2次元標高地図表示ツール

```
if (!terrain_height)return;
change_cursor(cursor_wait);
copy_to_undo_buffer();
if(selection_x_1<selection_x_2){
        start_x=selection_x_1;
        end_x=selection_x_2;
}else{
        start_x=selection_x_2;
        end_x=selection_x_1;
}
if(selection_y_1<selection_y_2){
        start_y=selection_y_1;
        end_y=selection_y_2;
}else{
        start_y=selection_y_2;
        end_y=selection_y_1;
}
```

- 保守性に悪影響を与えている可能性がある.
- 継承やラッパー関数を用いてコードクローンを減らせる可能性がある.

継承を用いたコードクローン削減方法



コードクローンに含まれるメソッド(getName())が定義されているクラス(Salesman,Engineer)を継承する新たなクラス(Employee)を作成する

要因2：プラットフォーム依存コード

Superopt：
CPUを指定して数式を入力すると，そのCPUに対応したアセンブリコードを出力する

```
case LSHIFTR_CO:
  printf("shrl      %s,%s",NAME(s2),NAME(d));break;
case ASHIFTR_CO:
  printf("sarl      %s,%s",NAME(s2),NAME(d));break;
case SHIFTL_CO:
  printf("shll      %s,%s",NAME(s2),NAME(d));break;
case ROTATEL_CO:
  printf("roll      %s,%s",NAME(s2),NAME(d));break;
```

```
#elif ALPHA
  ....
#elif HPPA
  ....
#elif SH
  ....
```

•保守性に悪影響を与えているとは限らない．
•プラットフォーム依存部分と非依存部分の切り分けを適切に行う必要がある．
（10個の成功事例ソフトウェアは，多プラットフォーム対応のためにモジュール間クローンが多くなっている）

要因3：自動生成コード

- プログラムトランスレータ
  - Pascal-to-C translator
    自動生成されたマクロ定義と関数定義
- GUIビルダツール
  - GLADE
    GUI部品の定義
- 字句・構文解析器ジェネレータ
  - flex
    自動生成されたマクロ定義と関数定義

•自動生成されたコードは，人手で編集することが想定されていない．
•保守性に悪影響を与えているとはいえない．

要因4：GUIに関するコード

Xcdroast：

CDライティングソフト

```
tbl = gtk_table_new(3,8,TRUE);
gtk_table_set_row_spacings(GTK_TABLE(tbl),10);
gtk_table_set_col_spacings(GTK_TABLE(tbl),10);
gtk_box_pack_start(GTK_BOX(vbox),tbl,FALSE,F
ALSE,10);
gtk_widget_show(tbl);
l1 = rightjust_gtk_label_new(text(103));
gtk_table_attach_defaults(GTK_TABLE(tbl),l1,0,2,
0,1);
gtk_widget_show(l1);
```

- GUI部品を生成するための定型処理がクローンとなっている．
- 保守性に悪影響を与えているとは限らない．

コードクローン含有率

CRate

		自動コード生成 有	自動コード生成 無
GUI	有	16.9%	13.8%
GUI	無	13.4%	7.2%

inMCRate

		自動コード生成 有	自動コード生成 無
GUI	有	13.6%	11.7%
GUI	無	7.9%	6.1%

interMCRate

		自動コード生成 有	自動コード生成 無
GUI	有	4.9%	3.4%
GUI	無	6.6%	1.3%

許容できるコードクローン含有率の目安となる．

事例2のまとめ

- 全てのコードクローンが保守性を低下させるとはいえない．
  - 自動生成コード，GUI部品の定義，プラットフォーム依存コードの定型処理
- ソースコードからGUI生成の定型処理と自動生成コードを除いた状態で，
  - クローン含有率7.2%
  - モジュール内クローン含有率6.1%
  - モジュール間クローン含有率1.3%

を大きく超えるならば，プログラムの設計・コーディングに問題がないか確認すべきであろう．

事例3：FreeBSD, Linux, NetBSD

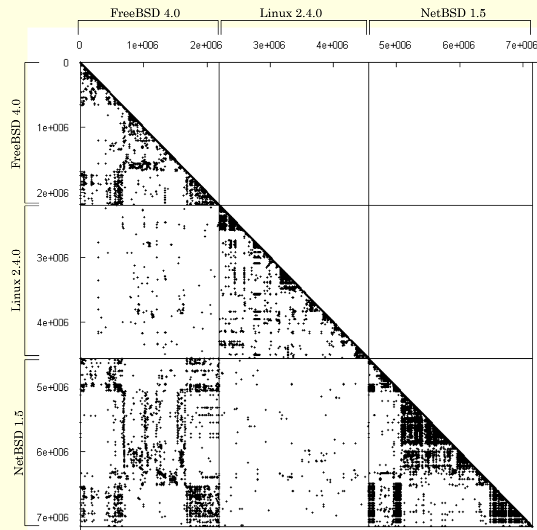
異なるソフトウェア間のクローンの検出・分析

- FreeBSD 4.0 (C 220万行)
- Linux 2.4.0 (C 240万行)
- NetBSD 1.5（C 260万行)
  - FreeBSDとNetBSDは同じソースコードから，Linuxは異なるソースコード

出典：

Toshihiro Kamiya, Shinji Kusumoto, and Katsuro 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, (2002-7).

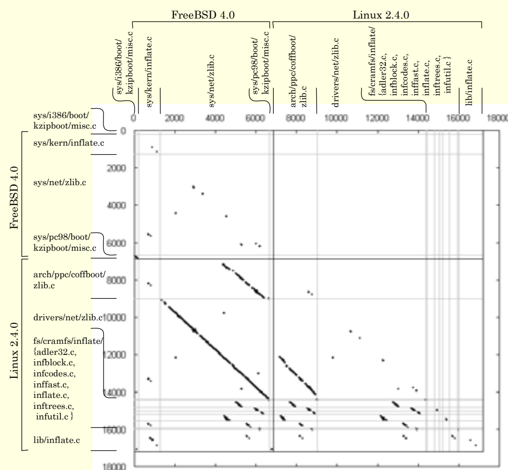
3つのOS間のコードクローン



FreeBSDとNetBSD間にクローンが多く見られる.

FreeBSDとLinuxのコードクローン

- ドライバ部分に多くのクローン「ファイル」が存在する.
- 共通のソースから分岐したソースファイル
- 名前が付け替えられたソースファイル
- 複数に分割されたソースファイル.

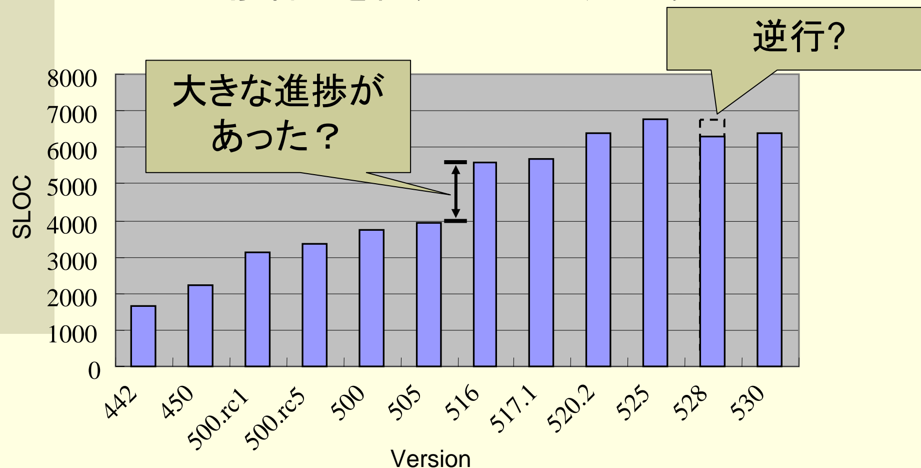


事例3のまとめ

- コードクローンは，ソースコードの派生や流用の有無を調べるのに役立つ.
- その他
  - 著作権侵害の発見，立証
  - プログラミング演習での他人のプログラムのコピーの発見なども

コードクローンと進捗管理

SLOC (ソースコードの行数)は有効な指標ではあるがプロジェクト進捗管理を行うには不十分である.



プロジェクトの進捗は，行数だけでは分からない.

コードクローンと進捗管理

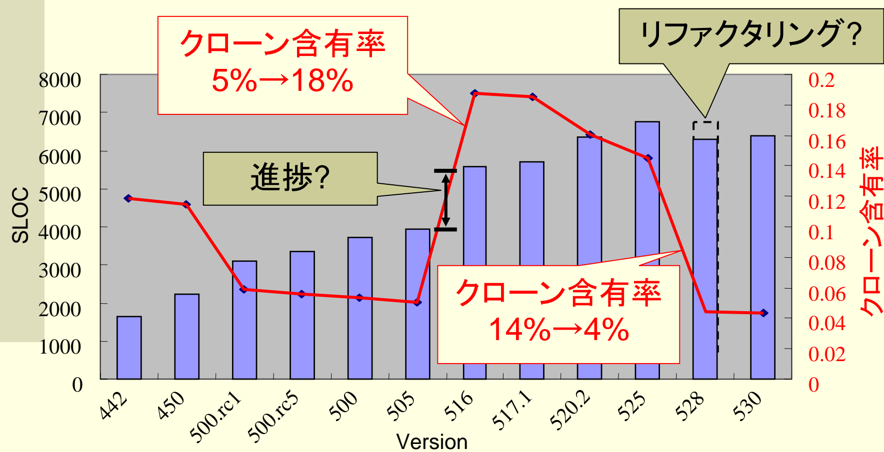
しばしば，次のような話を聞きます …

- 協力会社がコピー＆ペーストによってソースコードの行数の水増しを行い，見かけの仕事量（行数/月）を増やしている.
- 開発者がデバッグコードを挿入したり削除することがことで，行数が急に増えたり減ったりする.

プロジェクト管理者は，行数に代わる指標を必要としている.

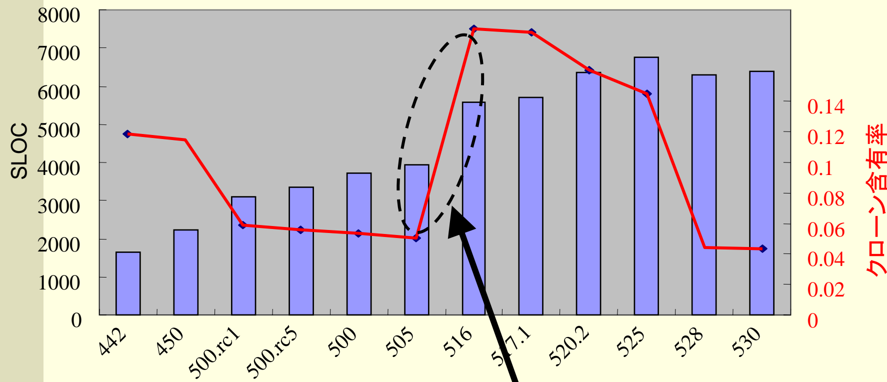
コードクローンを用いたプロジェクト進捗管理

クローン含有率により，行数の増減の原因を推測できる.



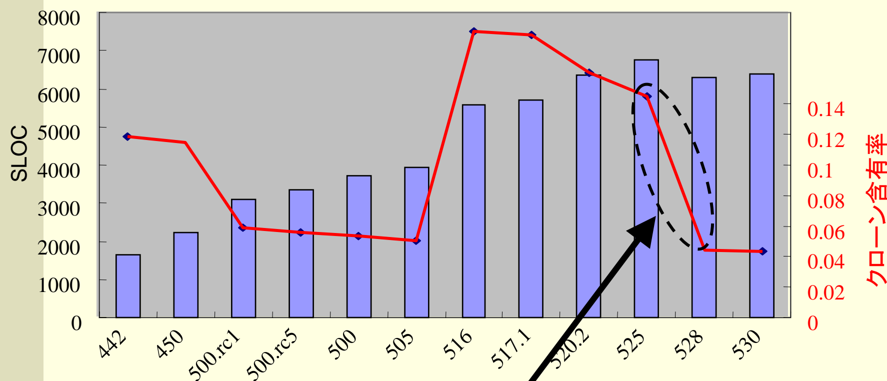
プロジェクトの進捗状況がより詳しく分かる.

より詳細な分析 (1)



FILE	Ver. 505 SLOC	Ver. 505 CRate	Ver. 516 SLOC	Ver. 516 CRate
charproc.c	566	0	792	7.58%
mml2mid.c	681	0	891	8.31%
mmlproc.c	1585	9.3%	2261	12.47%
note.c	805	4.78%	1348	29.97%

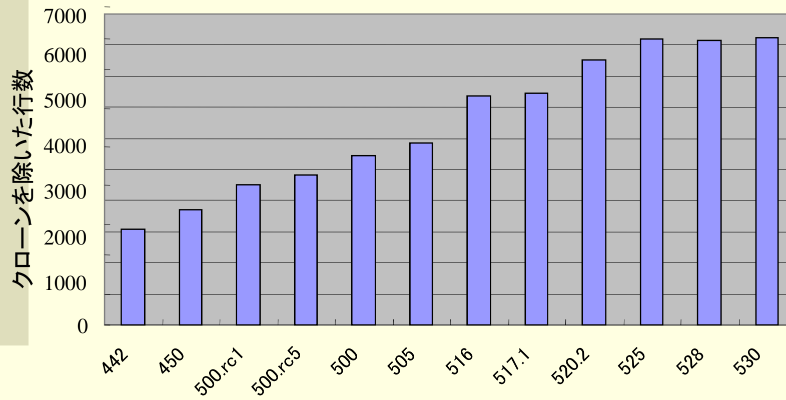
より詳細な分析 (2)



FILE	Ver. 525 SLOC	Ver. 525 CRate	Ver. 528 SLOC	Ver. 528 CRate
charproc.c	970	4.62%	979	4.51%
mml2mid.c	1318	0	1342	0
mmlproc.c	2506	19.58%	2360	9.06%
note.c	1591	26.36%	1129	0

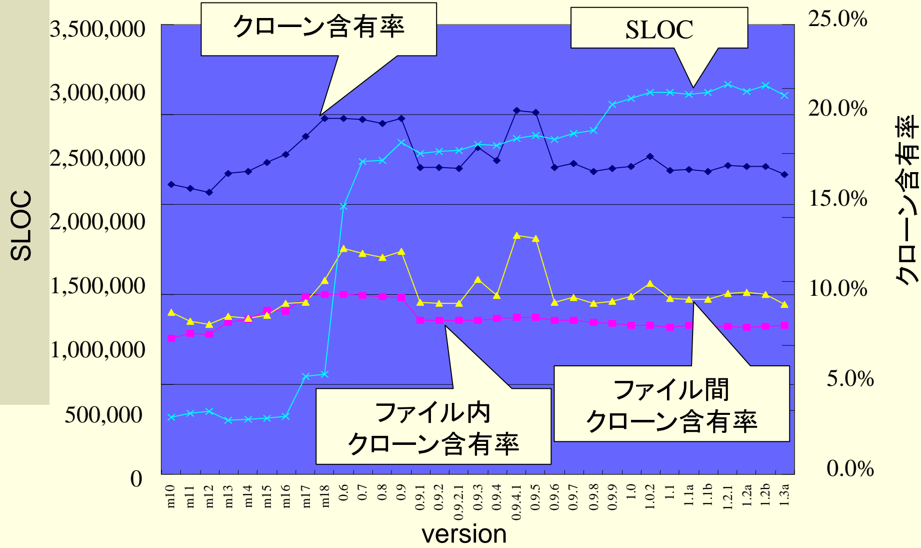
クローンを除いた行数

行数 ー（クローン含有行÷2）



ほぼ右上がりに行数が増えている.

Mozillaの場合



まとめ

- コードクローンとは，ソースコード中の類似したコード片である.
- クローン検出
  - ソフトウェア保守
  - ソースコードを調べるための手段
- 適用事例
  - レガシーソフトウェアやオープンソースソフトウェア
- 進捗管理への応用

演習レポート課題

- 任意のプログラム（ソースコード）について，講義で紹介したツール ICCAを使って，コードクローンの調査を行い，以下の点について調査結果を報告してください.
  - どのようなプログラムに対してツールを適用したか
  - どのような結果が得られたのか
  - 得られた結果から何か言えることがあるか.
  - 得られた結果に基づいてコードを改善することが可能か．可能なら，どのようにコードを改善することができるか.
  - 調査の結果得られた知見に基づいて，コードクローン検出の改良方法や，検出結果の新たな応用方法が考えられるか.
  - その他気づいたこと，感想など.
- 調査対象とするプログラムは，合法的に入手したものであれば，自作，他作は問いません．他作のコードを使用する場合は，レポートにその出所を明記してください.

演習課題 ― 提出期限

- レポート提出期限
  - 2006年1月19日（金）2限
  - 希望者はレポート課題の内容を発表すること
    - レポートの内容について説明する.
    - レポート用紙をスクリーンに映す，もしくは，パワーポイント等のプレゼン資料を用いる.
  - 時間が余れば，提出されたレポートの中からいくつかを講義中に（門田が）紹介します.
- 連絡先
  - 門田暁人 akito-m@is.naist.jp
  - B303室，内線5311