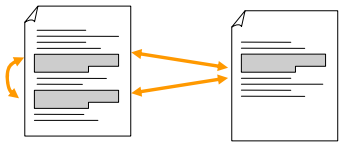
コードクローン問題に対処する技術の動向（コードクローン入門）

神谷 年洋
科学技術振興機構さきがけ

Special Thanks to:
植田 泰士(Jaxa), 肥後 芳樹(大阪大学), 吉田 則裕(大阪大学)

コードクローンとは

- ソースコード中に同一あるいは類似したコード断片があるとき、それらをコードクローンという



コードクローンが引き起こす問題

- コードクローンはソフトウェア保守を困難にする
  - バグ修正や機能の変更が困難になる
    - コードクローンにバグが含まれると、該当するすべてのコード断片について修正を検討する必要がある
- 大規模ソフトウェアでは、開発者がコードクローンをすべて覚えておく（発見する）ことはできない

発表の概略

- コードクローンとは
  - コードクローンが引き起こす問題
  - コードクローンが発生する原因
  - コードクローンへの対策
- われわれのアプローチ
  - コードクローン検出ツールCCFinder
  - Gemini
  - Aries
- 関連研究
  - コードクローン検出手法
    - ツリーのマッチング
    - DP (Dynamic Programming)
    - 特徴メトリクス
  - Simultaneous Editing, Linked Editing
  - Editing Process Patterns

コードクローンへの対策マップ

コードクローン検出手法マップ

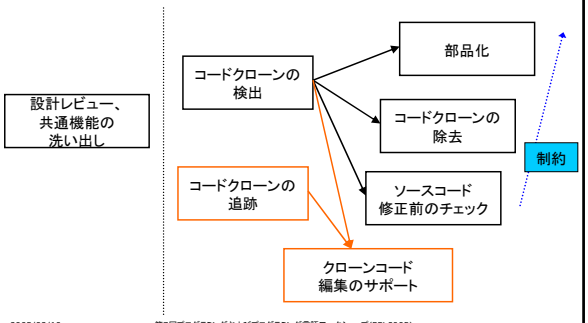
コードクローンが発生する原因

- 直接的な原因（開発者の作業）
  - コピー＆ペースト
  - 意図的な作りこみ
    - 実行時性能を稼ぐためのループの展開
  - 定型コード
    - エラー処理
    - 初期化
  - ツールで生成されたコード
    - ただし、生成ツールとそのソースが管理できていれば問題にはならない
      - （！生成ツールが対応できない処理）
      - （！ソースが失われる、あるいは、提供されない）

コードクローンが発生する原因（続き）

- 間接的な原因（開発プロセス）
  - プログラミング言語に適切な機能がない
    - スコープや動的なディスパッチなど
    - 長期にわたって運用されているシステムでは深刻な問題
  - 納期
    - コードや設計をレビューする（見直す）時間がない
    - →共通機能の洗い出しの不足
  - 「動いているコードには触るな」方針
    - ＝積極的にコードクローンを作る方針

コードクローンへの対策マップ



コードクローンの検出

- 検出
  - 与えられたソースコードから、文字列比較、木の比較、特徴メトリクスなどによって、同一部分を抽出する
    - Dup, Duploc, CloneDR, Balazinska
  - （！検出誤差）
- (参考) コードクローンの予防
  - モジュールと機能の対照表を作ってレビューする

コードクローンの除去

- 自動的な除去
  - クローンをまとめるマクロを生成する
  - ソースコードを修正する
  - （！自動的に除去可能なコードクローンだけを扱うことになる）
- 除去作業のサポート
  - 適用可能なリファクタリングパターンを提示する
  - （！手作業でソースコードを編集する必要がある）

クローンコード編集のサポート

- Simultaneous Editing
  - コードクローンになっているコード断片のひとつに対する修正を、他のコード断片に波及させる
- コードのバリエーションを管理するツール
  - プログラミング言語独立の、任意のテキストを対象とした構成管理ツール(XVCL)
  - 強力なマクロ

コードクローンの追跡

- テキストエディタのコピー＆ペースト操作を記録することで、コードクローンの発生を検出する

発表の概略

- コードクローンとは
  - コードクローンが引き起こす問題
  - コードクローンが発生する原因
  - コードクローンへの対策
- われわれのアプローチ
  - コードクローン検出ツールCCFinder
  - Gemini
  - Aries
- 関連研究
  - コードクローン検出手法
    - ツリーのマッチング
    - DP (Dynamic Programming)
    - 特徴メトリクス
  - Simultaneous Editing, Linked Editing
  - Editing Process Patterns

コードクローンへの対策マップ

コードクローン検出手法マップ

われわれのアプローチ

- コードクローンの検出 → コードクローンの除去
  - コードクローン検出ツールCCFinder
  - 分析ツールGemini
  - リファクタリングサポートツールAries

コードクローン検出ツール CCFinder

- 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ツール
- 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用などがある
- 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

- 以下の4つのステップを順に行う
  (1) ソースコードを入力してトークンの列を作る
  (2) プログラミング言語の文法の知識を利用した前処理をする
  (3) 一致部分列を特定する
  (4) 一致部分列のソースコード上の位置を出力する

単純化した例を使って説明・・・

検出手順 入力

(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力

```
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];
}
```


検出手順 入力(続き)

(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力

トークンに切り分ける

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


検出手順 前処理

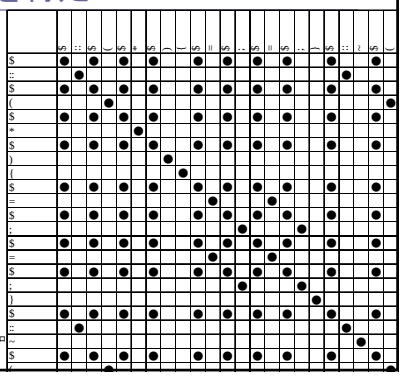
(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力

ユーザー定義名を$に置換

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

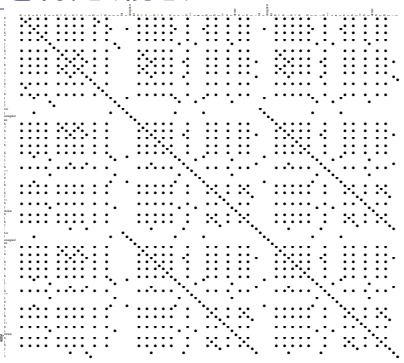
検出手順
一致部分列を特定

(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力



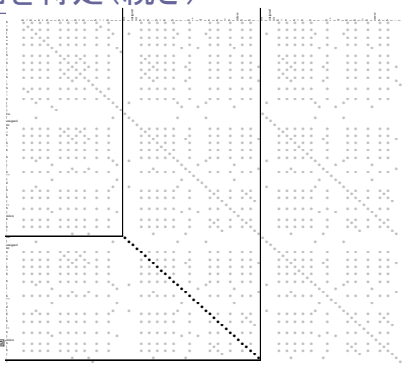
検出手順
一致部分列を特定（続き）

(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力



検出手順
一致部分列を特定（続き）

(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力



検出手順
出力

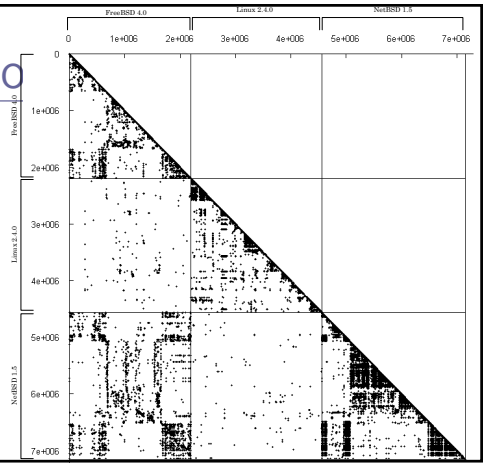
(1)入力
(2)前処理
(3)一致部分列を特定
(4)出力

```
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…

- 前処理には、パターンマッチと変形ルールを用いて、
  - 名前空間を除去
  - ユーザ定義名の置き換え
  - テーブル初期化部分の除去
  - モジュールの区切りを認識
- 最適化
  - 行列ではなく接辞尾木（suffix tree）を用いたアルゴリズム O(n+m)
  - いろいろ枝狩り

3つのO



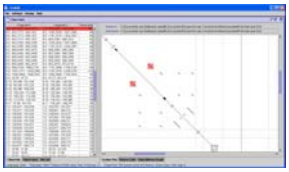
コードクローン分析へ

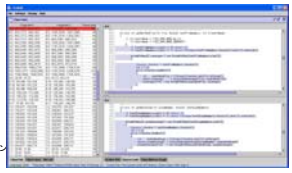
- CCFinderはコードクローンを検出できた
  - CCFinderはコードクローンのファイルや行、カラムを出力する
- でも、数百万行のソースコードから検出される何千、何万ものコードクローンを分析するのはやっぱり手間がかかる
- コードクローンの分析をサポート！

実は、ここまでの散布図はgnuplotで描いてました

コードクローン分析ツール Gemini

- GUIを用いた対話的な分析
  - 散布図上で拡大縮小、選択
  - 選択部分のソースコードを表示
  - メトリクスを用いたコードクローンの絞り込み
  - コードクローンの発生状況によるソースコードの整列
  - ギャップト・クローン（隣接したコードクローン）の検出機能





コードクローンの絞込みに用いられるメトリクス

メトリクスはクローンセット（互いに類似するコードの断片の集合）に対して定義される

分散
- RAD：そのクローンセットのコード断）を含むソースファイルの、ディレクトリ内での広がり
  - → RADが大きいコードクローンは複数のサブシステムで共通に用いられるコード

サイズ
- LEN：そのクローンセットに含まれるコード断片の長さ（トークン数）
- LNR：そのクローンセットに含まれるコード断片の、繰り返し部分以外の長さ（の最大値）
  - → LNR/LENが小さいコードクローンは、繰り返しを含むコード
- POP：そのクローンセットに含まれるコード断片の数
- DFL：そのクローンセットのコードをまとめたときに減少するコードの量の予測値

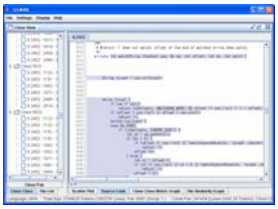
プログラミング言語にあまり依存しない

RADが大きいコードクローンの例

JDK 1.5.0.01
RAD＝9

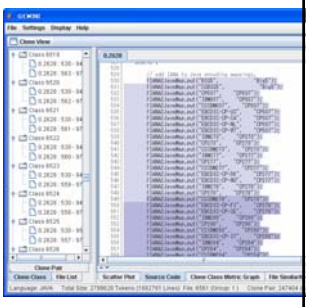
LNR/LENが大きいコードクローンの例

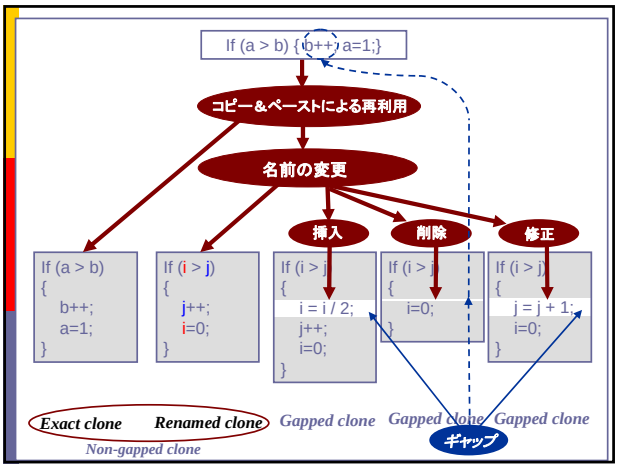
- LNR(/LEN)大
  com.sun.org.apache.xerces.internal.impl.xpath.regex.RegularExpressionの2つのメソッド
  - private int matchString(Context con, Op op, int offset, int dx, int opts)と
  - private int matchCharacterIterator(Context con, Op op, int offset, int dx, int opts)
  - 約400行（1500トークン）



LNR/LENが小さいコードクローンの例

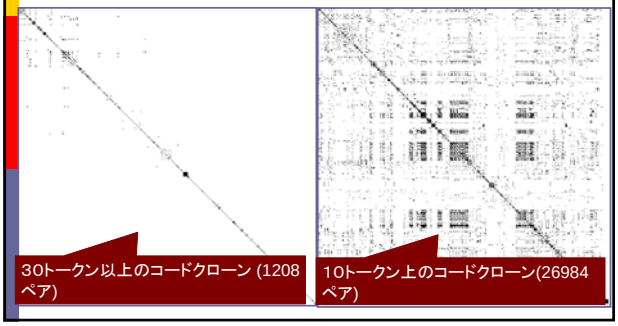
- LNR(/LEN)小
  - com.sun.org.apache.xerces.internal.util.EncodingMapのスタティックイニシャライザ
  - 約430行（LEN＝約2000, LNR ＜ 10）

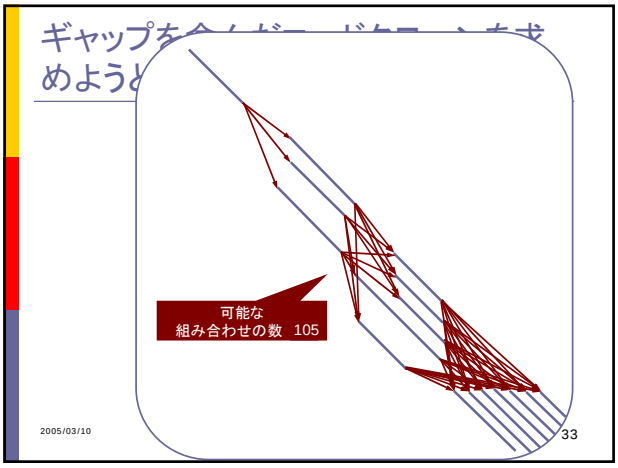




ギャップを想定して、短いコードクローンを検出すると‥

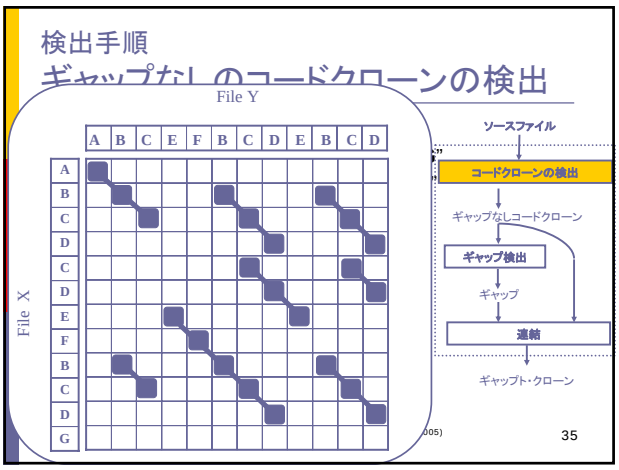
- 短いコードクローンはたくさん検出される

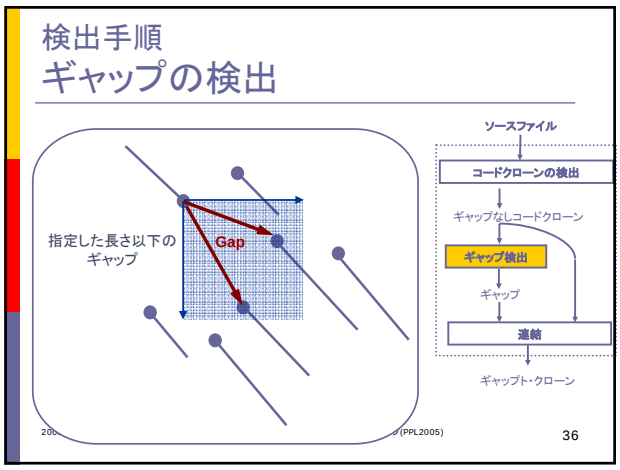


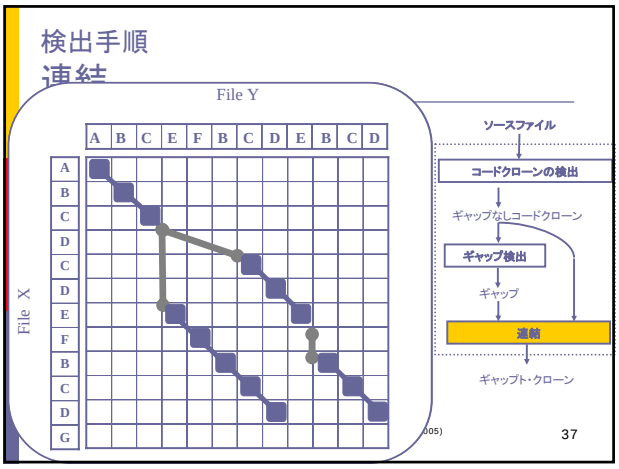


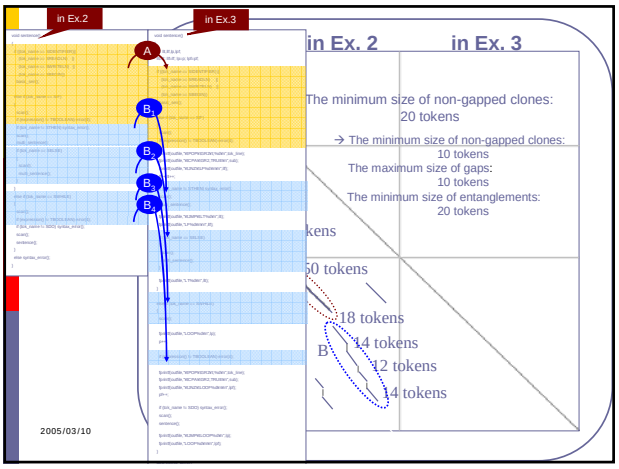
ギャップト・クローンの検出手順

(1)（ギャップなしの）コードクローンの検出
(2)ギャップを検出
(3)ギャップをはさんでコードクローンを連結する









リファクタリングのサポートへ

- Geminiによって、コードクローンの分析をする環境はできたけれど
- 「検出されたコードクローンをどうすればよいのか分からない」
- 対策をアドバイスできないか
  - ここのコードクローンに対して、適用可能なリファクタリングパターンを提示する

リファクタリングサポートツール Ariesのアプローチ

- 検出したクローンの集約にはリファクタリングパターンを用いる.
  - メソッドの抽出：クローンをメソッドとして抽出
  - メソッドの引き上げ：クローンを親クラスへ引き上げる
- 検出したクローンが上記のどちらに適しているか，またはどちらにも適していないのかをメトリクスを用いて判定する.

リファクタリングパターン メソッドの抽出

ある部分を新たなメソッドとして抽出するためには，周囲との結合度が低いことが望ましい(抽出部分の外側で定義された変数を用いていないことが望ましい)

```
void methodA(int i){
    methodZ();
    System.out.println("name:" + name);
    System.out.println("amount:" + i);
}
void methodB(int i){
    methodY();
    System.out.println("name:" + name);
    System.out.println("amount:" + i);
}
```

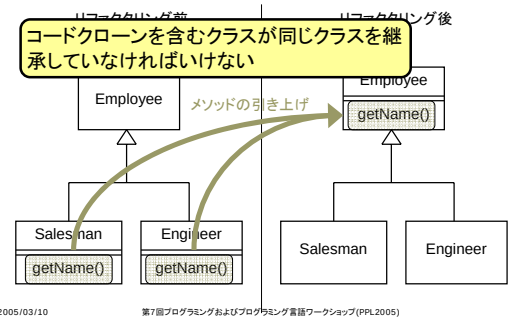
メソッドの抽出

```
void methodA(int i){
    methodZ();
    methodC(i);
}
void methodB(int i){
    methodY();
    methodC(i);
}
void methodC(int i){
    System.out.println("name:" + name);
    System.out.println("amount:" + i);
}
```

抽出したメソッドの呼び出し

リファクタリングパターン メソッドの引き上げ

コードクローンを含むクラスが同じクラスを継承していなければいけない



リファクタリングパターンを判定するためのメトリクス（1/2）

- 以下の2つについて計測する
  - 結合度の計測：クローンの外側で定義された変数の，クローンの内側での使用（代入，参照）数
  - クローンの分散度の計測：クローンが含まれるクラスがクラス階層においてどのような関係にあるか

解析結果はメトリクスとして数値化する

- クローンセットS_1にはコード片f_1とf_2が含まれている．
- コード片f_1内では外部定義の変数a，bをそれぞれ1，2回使用している．
- コード片f_2内では外部定義の変数c，dをそれぞれ3，4回使用している．
  - $RVK(S_1)$：(2 + 2)/2 = 2
  - $RVN(S_1)$：((1 + 2) + (3 + 4))/2 = 5

$$RVK(S) = \frac{1}{n}\sum_{i=1}^{n} m_i$$

$$RVN(S) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m_j} UC_{fi}(v^i{}_j)$$

となる．

- クローンセットS_2内の全てのコード片が，ある1つのクラス内に存在する場合
  - $DCH(S_2)$：0
- クローンセットS_3内の全てのコード片があるクラスとその子クラス内に存在する場合
  - $DCH(S_3)$：1
- クローンセットS_4内のコード片が含まれるクラスが共通の親を持たない場合
  - $DCH(S_4)$：-1

$$DCH(S) = \max(D(C_1, C_p), D(C_2, C_p), \cdots, D(C_n, C_p))$$

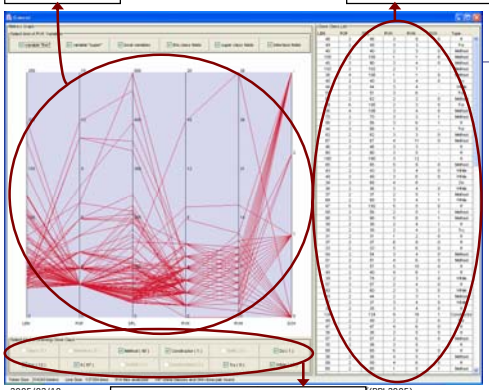
$C_1, C_2, \cdots, C_n$が共通の親クラスを持たないとき，

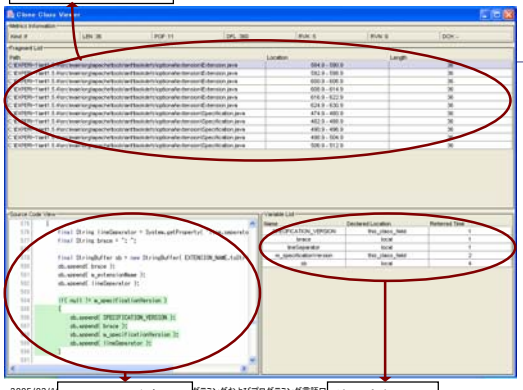
$$DCH(S) = -1$$

とする

ツールAries

- 宣言単位
  - class宣言，interface宣言
- メソッド単位
  - メソッド本体，コンストラクタ，スタティックイニシャライザ
- 文単位
  - do文，for文，if文，switch文，synchronized文，try文，while文





適用実験
概要

- Ant (version 1.6.0)
- 入力
  - ソースファイル数： 627個
  - 総行数： 約18万行
- 構造的なクローン検出には約30秒
  - 151個のクローンセットを検出
  - 実験環境
    - FreeBSD 4.9
    - CPU Xeon2.8G×2
    - メモリ 4GB

適用実験
Ant（メソッドの抽出）

- 「メソッドの抽出」を適用するためにクローンを絞り込んだ条件は次の3つ
  - 文単位のクローンである
    - クローンユニット選択チェックボックスで文単位のクローンにチェックを入れる
  - 全てのコード片が1つのクラス内に存在する
    - メトリクスグラフのDCHの上限を1より小さくする.
  - クローンの外部で定義されたローカル変数を高々1つしか使用していない
    - メトリクスグラフのRVKの上限を2より小さくする.

151個のうちの32個が該当した

適用実験
Ant（メソッドの抽出）

- 32個の内訳は以下の通り

```
if (!isChecked()) {
    // make sure we don't have a circular reference here
    Stack stk = new Stack();
    stk.push(this);
    dieOnCircularReference(stk, getProject());
}
```

外部定義の変数を引数として抽出	18個
引数とした変数を返り値として返す	7個
その他	4個

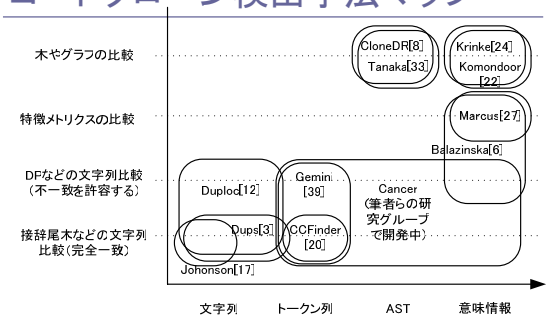
発表の概略

- コードクローンとは
  - コードクローンが引き起こす問題
  - コードクローンが発生する原因
  - コードクローンへの対策
- われわれのアプローチ
  - コードクローン検出ツールCCFinder
  - Gemini
  - Aries
- 関連研究
  - コードクローン検出手法
    - ツリーのマッチング
    - DP (Dynamic Programming)
    - 特徴メトリクス
  - Simultaneous Editing, Linked Editing
  - Editing Process Patterns

コードクローンへの対策マップ

コードクローン検出手法マップ

コードクローン検出手法マップ



神谷 年洋, "コードクローンとは：コードクローンが引き起こす問題、その対策の現状", 電子情報通信学会誌 Vol. 87, No. 9, pp. 791-797 (2004-9).

コードクローンの定義

- （定義1）ソースコード中に同一あるいは類似したコード断片があるとき、それらをコードクローンという
  - （！「同一あるいは類似」の意味は人に依存する）
- （定義2）開発者がソースコードをコピー＆ペーストしたとき、コピーとオリジナルはコードクローンである
  - （！それ以外の理由によって発生した類似部分はコードクローンではない）
- （定義3）2つのソフトウェア部品が、ある基準によって同じであると判定されるとき、それらはコードクローンである
  - （！人間の判断と、その基準がどれだけ一致するか）

関連研究 CloneDR

- コードクローン検出＋除去ツール CloneDR
  - ASTの部分木の比較
  - ソースコードの自動的な修正
- 特徴
  - 検出されるものはマージ可能なコードクローン
  - 検出時間はO(n^2) n:ASTのノード数

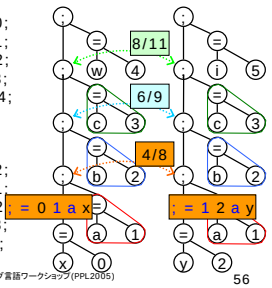
I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," Proc. IEEE International Conference on Software Maintenance (ICSM) '98, pp. 368-377, Bethesda, Maryland, Nov. 1998.

CloneDRのコードクローン検出アルゴリズム

(1) ソースコードをパースして、ASTを作成
(2) ハッシュ値によるコードクローン検出
- ASTの各ノードのハッシュ値を計算
- ハッシュ値が同じノードはコードクローン

(3) 含んでいるトークンの共通度によるコードクローン検出
- それぞれのノードを根とする部分木に含まれるトークンの種類の類似度を計算
- 類似度がしきい値以上ならコードクローン

```
void f()
{
  x = 0;
  a = 1;
  b = 2;
  c = 3;
  w = 4;
}
void g()
{
  y = 2;
  a = 1;
  b = 2;
  c = 3;
  i = 5;
}
```



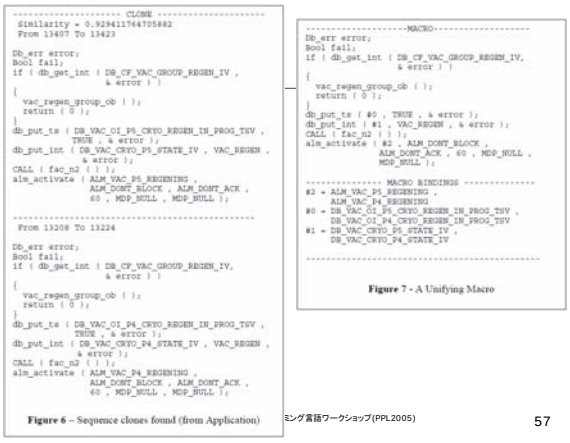


Figure 6 – Sequence clones found (from Application)

Figure 7 - A Unifying Macro

関連研究 Balazinska

- コードクローン検出ツール
  - DP＋特徴メトリクス
- 検出アルゴリズム
  - (1)メソッドを特徴メトリクスによって比較し、類似しているメソッドをコードクローンの候補とする
  - (2)候補となったメソッドをDPによってさらに比較し、類似しているものをコードクローンとする

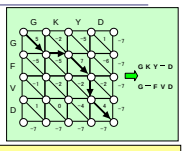
M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K.A. Kontogiannis, "Measuring Clone Based Reengineering Opportunities," Proc. 6th IEEE Int'l Symposium on Software Metrics (METRICS '99), pp. 292-303, Boca Raton, Florida, Nov. 1999.

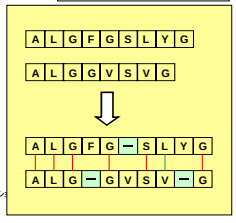
特徴メトリクス

- コードのブロックや関数を対象として、以下のメトリクスを計測する
  - 利用されているグローバル変数の数
  - 呼ばれている関数の数
  - 引数や戻り値の数
  - McCabeのサイクロマチック数
- Balazinskaらの研究ではメトリクスの計測値が±1の範囲で一致するものをコードクローンとして検出している
- （！小さなコード断片については計測できない）
- （！ソースコードの小さな修正が、メトリクス値の大きな違いを生むことがある）

動的計画法（Dynamic Programming）

- Diffのアルゴリズム
  - 与えられた2つの文字列のedit distanceを計算する
- ギャップ・クローンを検出することができる
- 計算時間はO(mn) (m, n : length of sequences)





Linked Editing

(1)コードクローンを検出する
(2)開発者が、コードクローンのコード断片の間に「リンク」を設定する
(3)リンクが設定されたコード断片への編集は、他のコード断片へも反映される
(4)ファイルを保存するときは、リンク情報も同時に保存する

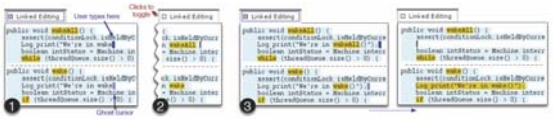


Figure 2. (1) Adding a line to two clones. (2) Modifying one instance. (3) Deleting line in one instance.

Michael Toomim, Andrew Begel, Susan L. Graham, "Managing Duplicated Code with Linked Editing", Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04) pp. 173-180

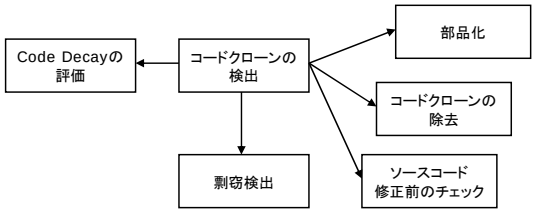
Editing Process Patterns

- テキストエディタのコピー＆ペースト作業を記録する
  - Support for Programmer Observation
  - Support for C&P Programming
  - Support for Program Understanding
  - Support for Programming by Demonstration
- Miryung Kim, Vibha Sazawal, and David Notkin, "Supporting Uses of Editing Process Patterns", Workshop on Behavior Based User Interface Customization,Intelligent User Interface, Jan 13-16, 2004

[1] S.F. Altschul, W. Gish, W. Miller, E. Myers, D. Lipman, "Basic Local Alignment Tool," *Journal of Molecular Biology*, 215, pp. 403-410, (1990).
[2] A. Aiken, "A System for Detecting Software Plagiarism (Moss Homepage)", http://www.cs.berkeley.edu/~aiken/moss.html [Last visited 1st Feb. 2003].
[3] B. S. Baker, "A Program for Identifying Duplicated Code," *Computing Science and Statistics*, 24:49-57, 1992.
[4] B. S. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," *Proc. IEEE Working Conf. on Reverse Engineering*, pp. 86-95, July 1995.
[5] B. S. Baker, "Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance," *SIAM Journal on Computing*, 26(5):1343-1362, 1997.
[6] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K.A. Kontogiannis, "Measuring Clone Based Reengineering Opportunities," *Proc. 6th IEEE Int'l Symposium on Software Metrics* (METRICS '99), pp. 292-303, Boca Raton, Florida, Nov. 1999.
[7] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K.A. Kontogiannis, "Partial Redesign of Java Software Systems Based on Clone Analysis," *Proc. 6th IEEE Working Conference on Reverse Engineering* (WCRE '99), pp. 326-336, Atlanta, Georgia, Oct. 1999.
[8] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," *Proc. IEEE International Conference on Software Maintenance* (ICSM) '98, pp. 368-377, Bethesda, Maryland, Nov. 1998.
[9] K.W., Church, and J.I. Helfman, "Dotplot: A Program for Exploring Self-Similarity in Millions of Lines of Text and Code," *Journal of Computational and Graphical Statistics*, V2, N2, pp. 153-174 (1993).
[10] James O. Coplien, *Multi-Paradigm Design for C++*, Pearson Education (2001). ジェームス・O・コプリン(著), 金沢 典子, 平鍋 健児, 羽生田 栄一(訳), マルチパラダイムデザイン, ピアソン・エデュケーション (2001).
[11] Elizabeth Burd, John Bailey, "Evaluating Clone Detection Tools for Use during Preventative Maintenance," *Proc. 2nd IEEE International Workshop on Source Code Analysis and Manipulation* (SCAM) 2002, pp. 36-43. Montreal, Canada, Oct. 2002.
[12] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," *Proc. IEEE International Conference on Software Maintenance* (ICSM) '99, pp. 109-118. Oxford, England. Aug. 1999.
[13] M. Fowler, *Refactoring: improving the design of existing code*, Addison Wesley, 1999.
[14] FreeBSD. http://www.freebsd.org/.
[15] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, pp. 89-180. Cambridge University Press, 1997.
[16] T. Imai, Y. Kataoka, and T. Fukaya, "Evaluating Software Maintenance Cost Using Functional Redundancy Metrics," Proc. 26th International Computer Software and Applications Conference (Compsac 2002), pp.299-306, 2002.
[17] J.H. Johnson, "Identifying Redundancy in Source Code Using Fingerprints," Proc. IBM Centre for Advanced Studies Conference (CAS CON) '93, pp. 171-183, Toronto, Ontario. Oct. 1993.
[18] J.H. Johnson, "Substring Matching for Clone Detection and Change Tracking," Proc. IEEE Int'l Conf. on Software Maintenance (ICSM) '94, pp. 120-126. Victoria, British Columbia, Canada. Sep. 1994.

[19] J.H. Johnson, "Using Textual Redundancy to Understand Change," *Proc. IBM Centre for Advanced Studies Conference (CASCON) '95*, pp. 34-40, Nov. 1999.
[20] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Multi-Linguistic Token-Based Code Clone Detection System for Large Scale Source Code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654-670, (2002-7).
[21] 金久 實 (編), ヒューマンゲノム計画, 共立出版 (1997).
[22] R. Komondoor and S. Horwitz, "Using Slicing to Identify Duplication in Source Code," *Proc. of the 8th International Symposium on Static Analysis(SAS), LNCS 2126*, pp. 40-56, Springer (2001).
[23] K.A. Kontogiannis, R. De Mori, E. Merlo, M. Galler, and M. Bernstein, "Pattern Matching Techniques for Clone Detection and Concept Detection," *Journal of Automated Software Engineering* Kluwer Academic Publishers, vol. 3, pp. 770-108, 1996.
[24] Jens Krinke, "Identifying Similar Code with Program Dependence Graphs," *Proc. of the 8th Working Conference on Reverse Engineering*, 2001.
[25] B. Laguë, E.M. Merlo, J. Mayrand, and J. Hudepohl. "Assessing the Benefits of Incorporating Function Clone Detection in a Development Process," *Proc. IEEE Int'l Conf. on Software Maintenance* (ICSM) '97, pp. 314-321, Bari, Italy. Oct. 1997.
[26] Linux Online. http://www.linux.org/.
[27] A. Marcus, and J.I., Maletic, "Identification of High-Level Concept Clones in Source Code," Proc. Automated Software Engineering (ASE'01), San Diego, pp. 107-114, Nov. 2001.
[28] J. Mayrand, C. Leblanc, and E. M. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," *Proc. IEEE Int'l Conference on Software Maintenance* (ICSM) '96, pp. 244-253, Monterey, California, Nov. 1996.
[29] 門田 暁人, 佐藤 徹一, 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 (2003-8).
[30] NetBSD Project. http://www.netbsd.org/.
[31] 大駒 誠一, COBOL入門 改訂版, 培風館 (1986).
[32] L. Prechelt, G. Malpohl, M. Philippsen, "JPlag: Finding Plagiarisms Among a Set of Programs," Technical Report, University of Karlsruhe, Department of Informatics, 2000.
[33] 田中 哲, "データマイニングを利用したプログラムの改善," 日本ソフトウェア科学会 第7回プログラミングおよび応用のシステムに関するワークショップ(SPA 2004), pp. 1-8, (2004-3).
[34] Will Tracz, *Confessions of a Used Program Salesman --- Institutionalizing Software Reuse* ---, Pearson Education (1995). W. トレイツ (著), 畑崎 隆雄, 林 雅弘, 鈴木 博之 (訳), ソフトウェア再利用の神話 --- ソフトウェア再利用の制度化に向けて ---, ピアソン・エデュケーション (2001).
[35] 山本 哲男, 松下 誠, 神谷 年洋, 井上 克郎, "ソフトウェアシステムの類似度とその計測ツールSMMT," 電子情報通信学会論文誌, vol. J85-D-I, no. 6, pp. 503-511. (2002-6).
[36] 植田 泰士, 神谷 年洋, 楠本 真二, 井上 克郎, "開発保守支援を目指したコードクローン分析環境," 電子情報通信学会論文誌 D-I, Vol. J86-D-I, No. 12, pp. 863-871 (2003-12).

コードクローン検出手法応用マップ



（参考）コードクローンを用いたソースコード評価・比較

- コードクローンによる評価
  - クローンをまとめることにより削減できるソースコードの割合[1]
  - 機能が重複して実装されている割合、回数[6]
  - クローンの分布
    - クローンが多く含まれるモジュール？
    - クローンクラス(同値類)の広がり
  - フォールトとクローンの統計的な関係？
- コードクローンによる比較
  - プロダクト間・バージョン間の類似度