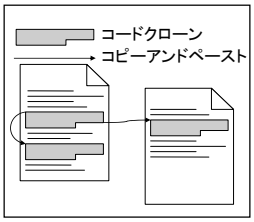
コードクローン検出技術とその利用法

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 井上研究室
肥後 芳樹(y-higo@ist.osaka-u.ac.jp)

コードクローンとは

- ソースコード中に存在する，他のコード片と一致または類似しているコード片
- さまざまな理由により生成される
  - コピーアンドペーストによる再利用
  - 定型的な処理
    - 例：ファイルオープン，データベース接続
  - 意図的な繰り返し
    - パフォーマンス重視
- ソフトウェアの保守を困難にする
  - あるコード片にバグがあると，そのコードクローン全てについて修正の検討を行う必要がある
  - 機能を追加する場合も同様のことがいえる



コードクローンの定義

コードクローンの定義

- コードクローンの一般的な定義はない
  - これまでにいくつかのコードクローン検出手法が提案されているが，それらはどれも異なるコードクローンの定義を持つ
- 紹介するコードクローン検出技術
  - 行単位での検出手法
  - AST(Abstract Syntax Tree)を用いた検出手法
  - PDG(Program Dependency Graph)を用いた検出手法
  - メトリクスを持ちいた検出手法
  - トークン単位での検出手法

コードクローンの定義
行単位での検出手法

- ソースコードを行単位で比較し，コードクローンを検出する[1]
  - 比較の前に空白，タブは取り除かれる
- 初期の検出技術
- 検出の精度が良くない
  - コーディングスタイルが違うと検出できない
    - 例：if 文や while 文の括弧の位置
  - 変数名が異なると検出できない
    - 変数名が違っても，ロジックが同じ部分はコードクローンとして検出したい

[1]B. S. Baker, *A Program for Identifying Duplicated Code*, Proc. Computing Science and Statistics 24th Symposium on the Interface, pp.49-57, Mar. 1992.

コードクローンの定義
ASTを用いた検出手法

- ソースコードを構文解析し，AST（Abstract Syntax Tree）を作成．一致する部分木をコードクローンとして検出する[2]
  - 変数名の違いは吸収される
- 実用的な検出手法であり，商用ツールとして実装されている
  - CloneDR: http://www.semanticdesigns.com/Products/Clone/

[2] I.D. Baxter, A. Yahin, L. Moura, M.S. Anna, and L. Bier, *Clone Detection Using Abstract Syntax Trees*, Proc. International Conference on Software Maintenance 98, pp368-377, 16-19, Nov. 1998.

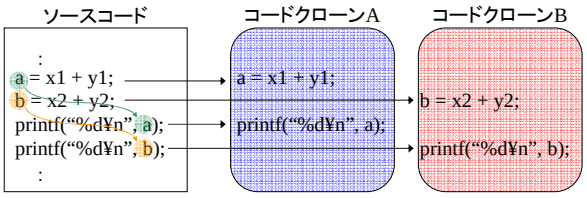
コードクローンの定義
PDGを用いた検出手法

- ソースコードを意味解析し，コントロールフロー・データフローを抽出．この情報からPDG(Program Dependence Graph)を作成し，一致する部分グラフをコードクローンとして検出[3].
- 非常に高い検出精度
- 他の検出手法では検出できないコードクローンも検出可能
  - インタートバインドクローン，リオーダードクローン
- 非常に計算コストが大きい
  - 実用的でない

[3] R. Komondoor and S. Horwitz, *Using slicing to identify duplication in source code*, Proc. the 8th International Symposium on Static Analysis, pp.40-56, July, 16-18, 2001.

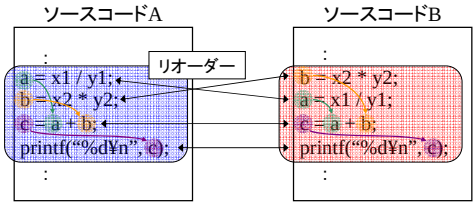
コードクローンの定義
PDGを用いた検出手法（インタートバインドクローン）

- 互いにインタートバインド（絡み合っている）なコードクローン



コードクローンの定義
PDGを用いた検出手法（リオーダードクローン）

- 文の順番が入れ替わっている（リオーダード）コードクローン



コードクローンの定義
メトリクスを用いた検出手法

- 関数・メソッド単位で（21種類の）メトリクスを計測，その値の近似・一致によりコードクローンを検出[4].
- 検出されるコードクローンは関数・メソッド単位
  - 関数の一部のみが重複している場合，検出できない
  - リファクタリングには有効
    - リファクタリングを行いやすい単位

[4] J. Mayland, C. Leblanc, and E.M. Merlo, *Experiment on the automatic detection of function clones in a software system using metrics*, Proc. International Conference on Software Maintenance 96, pp.244-253, Nov. 1996.

コードクローンの定義
トークン単位での検出手法

- ソースコードをトークン単位で直接比較することによりコードクローンを検出[5]
  - 型名，変数名などを表すトークンは，特別なトークンに置き換える.
- 非常に高いスケーラビリティ
  - ASTやPDGなどの作成を必要としない

[5] 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, Jul. 2002.

コードクローンの定義
比較

- クローンの検出精度が優れている手法
  - PDGを用いた検出手法
  - 行単位での検出手法，メトリクス単位手法での検出は検出漏れが多い
- 実用的に使える手法
  - ASTを用いた検出手法，トークン単位での検出手法
  - PDGを用いた検出手法は計算コストが大きすぎる

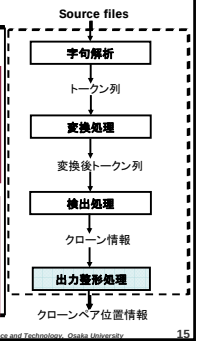
コードクローン検出ツール: CCFinder

コードクローン検出ツール: CCFinder 概要

- ソースコードをトークン単位で直接比較することにより，コードクローンを検出する（トークン単位での検出手法）
- より実用的なコードクローンを見つけることができるように設計されている
  - ユーザ定義名の置き換え
  - テーブル初期化部分の取り除き
  - モジュールの区切りの認識
- 解析結果はテキスト形式で出力
- 数百万行規模でも実用的な時間で解析可能

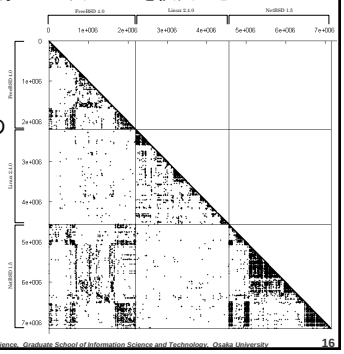
コードクローン検出ツール: CCFinder 検出プロセス

```
1. static void foo() throws RESyntaxException {
2.   String a[] = new String [] { "123,400", "abc", "orange 100" };
3.   org.apache.regexp.RE pat = new org.apache.regexp.RE("[0-9,]+");
4.   int sum = 0;
5.   for (int i = 0; i < a.length; ++i)
6.     if (pat.match(a[i]))
7.       sum += Sample.parseNumber(pat.getParen(0));
8.   System.out.println("sum = " + sum);
9. }
10. static void goo(String [] a) throws RESyntaxException {
11.   RE exp = new RE("[0-9,]+");
12.   int sum = 0;
13.   for (int i = 0; i < a.length; ++i)
14.     if (exp.match(a[i]))
15.       sum += parseNumber(exp.getParen(0));
16.   System.out.println("sum = " + sum);
17. }
```



コードクローン検出ツール: CCFinder 検出例

- 三つのOSのソースコードを対象にコードクローンを検出した
  - NetBSD
  - FreeBSD
  - Linux
- NetBSDとFreeBSDは多くのコードクローンが存在している
  - 祖先が同じだから



コードクローン検出ツール：Gemini

コードクローン分析ツール: Gemini 背景

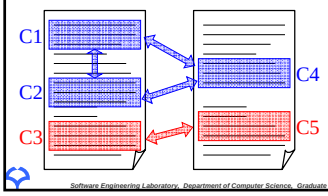
- CCFinder を用いることにより，大規模なソフトウェアから短時間でコードクローンを検出できるようになった
  - しかし，大量のコードクローンが検出されてしまい，手作業ですべてのコードクローンをチェックするのは非現実的
- 大量のコードクローン情報を上手に扱うメカニズムが必要
  - ソフトウェア内でのコードクローンの分布状態の表示
  - ユーザが興味のある特徴を持ったコードクローンの提示

コードクローン分析ツール: Gemini
概要

- CCFinderの出力した検出結果ファイル（テキストファイル）を読み込み，コードクローン情報を視覚的に表示
- インタラクティブなコードクローン分析を実現
- 主なビュー
  - **スキャタープロット**：コードクローンの量・分布状態を俯瞰的に表示
  - **メトリクスグラフ・ファイルリスト**：コードクローン・ファイルを定量的に特徴づける．またその特徴を用いたコードクローン・ファイルの選択機構を実現
  - **フィルタリングメトリクス RNR**：ユーザが目で確認を行う必要のないコードクローンのフィルタリングを行う

コードクローン分析ツール: Gemini
クローンペアとクローンセット

- クローンペア
  - 互いに一致または類似しているコード片の対（ペア）
- クローンセット
  - 互いに一致または類似しているコード片の集合（セット）
- 共にコードクローンを表す用語であるが，これらを使い分けることにより，よりスムーズに議論を行うことができる



クローンペア	クローンセット
(C1, C2)	{C1, C2, C4}
(C1, C4)	{C3, C5}
(C2, C4)	
(C3, C5)	

コードクローン分析ツール: Gemini
説明に用いる例

- Geminiの各特徴を説明するために以下の例を用いる
  - ディレクトリ D_1 以下には2つのファイル F_1, F_2 が存在する
  - ディレクトリ D_2 以下には2つのファイル F_3, F_4 が存在する
- 各ファイルは以下の5トークンから成る
  - F_1: a b c a b,
  - F_2: c c* c* a b,
  - F_3: d e f a b,
  - F_4: c c* d e f,

$C(F_i, j, k)$ はファイル F_i の j 番目のトークンから k 番目までのトークンのコード片を表す

- これら4つのファイルから長さが2以上のコードクローンを検出すると，次の3つのクローンセットが得られる．
  - S_1: { $C(F_1, 1, 2)$, $C(F_1, 4, 5)$, $C(F_2, 4, 5)$, $C(F_3, 4, 5)$ },
  - S_2: { $C(F_2, 1, 2)$, $C(F_2, 2, 3)$, $C(F_4, 1, 2)$ },
  - S_3: { $C(F_3, 1, 3)$, $C(F_4, 3, 5)$ }

コードクローン分析ツール: Gemini
フィルタリングメトリクス RNR（定義）

- CCFinderの検出するコードはトークンの列であり，重要でないコードクローンを多数検出してしまう
  - switch文の各caseエントリ
  - 連続した変数宣言や関数呼び出し
- フィルタリングメトリクス RNR(S)
  - クローンセット S に含まれるコード片の非繰り返し度を表す

定義

$$RNR(S) = 1 - \frac{\sum_{C \in S} Tokens_{repeated}(C)}{\sum_{C \in S} Tokens_{all}(C)}$$

$Tokens_{all}(C)$: コード片Cの総トークン数

$Tokens_{repeated}(C)$: コード片C中の繰り返し部分のトークン数

コードクローン分析ツール: Gemini
フィルタリングメトリクス RNR（例）

- アスタリスク＊の付いたトークンは，その直後のトークン列の繰り返しであることを表している
- クローンセット S_1, S_1, S_3 のRNRは，

$$RNR(S_1) = 1 - \frac{0+0+0+0}{2+2+2+2} = \frac{8}{8} = 1.0$$

$$RNR(S_2) = 1 - \frac{1+2+1}{2+2+2} = \frac{2}{6} = 0.33333\ldots$$

$$RNR(S_3) = 1 - \frac{0+0}{3+3} = \frac{6}{6} = 1.0$$

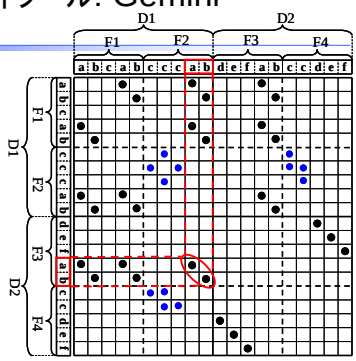
繰り返しの多いクローンであることを表している

F1: a b c a b,
F2: c c* c* a b,
F3: d e f a b,
F4: c c* d e f,

S_1: { $C(F_1, 1, 2)$, $C(F_1, 4, 5)$, $C(F_2, 4, 5)$, $C(F_3, 4, 5)$ },
S_2: { $C(F_2, 1, 2)$, $C(F_2, 2, 3)$, $C(F_4, 1, 2)$ },
S_3: { $C(F_3, 1, 3)$, $C(F_4, 3, 5)$ }

コードクローン分析ツール: Gemini
スキャタープロット

- 水平・垂直方向にソースコード中のトークンを出現順に配置
  - 原点は左上隅
- ●●●はその水平方向のトークンと垂直方向のトークンが等しいことを意味する
  - クローンペアは線分として出現する
  - ●は RNR が閾値以上のコードクローンを表す
  - ●は RNR が閾値未満のコードクローンを表す
- ひと目で分布状態を把握できる



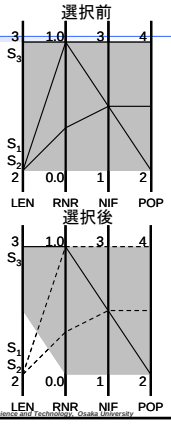
F1, F2, F3, F4 : files
D1, D2 : directories
●: matched position detected as a practical code clone
●: matched position detected as a non-interesting code clone

コードクローン分析ツール: Gemini
メトリクスグラフ(用いているメトリクス)

- ***LEN(S):*** クローンセット ***S*** 内に含まれるコード片のトークン数の平均値を表す
  - $LEN(S_1) = 2$, $LEN(S_2) = 2$, $LEN(S_3) = 3$
- ***POP(S)*: *S*** に含まれるコード片の数を表す
  - $POP(S_1) = 4$, $POP(S_2) = 3$, $POP(S_3) = 2$
- ***NIF(S)*: *S*** に含まれるコード片を所有しているファイルの数を表す
  - $NIF(S_1) = 3$, $NIF(S_2) = 2$, $NIF(S_3) = 2$
- ***RNR(S)*:** S に含まれるコード片の非繰り返し度を表す
  - $RNR(S_1) = 1.0$, $RNR(S_2) = 0.33$, $RNR(S_3) = 1.0$

コードクローン分析ツール: Gemini
メトリクスグラフ(メカニズム)

- 多次元並行座標表現を用いている
  - 各メトリクスにつき1つの座標軸が存在する
  - 各クローンセットにつき1つの折れ線がメトリクス値に基づいて描画される
- ユーザは各メトリクスの上限・下限を変更することでクローンセットの選択・選択解除を行う
  - 全てのメトリクスが上限と下限の間にあるクローンセットが選択状態になる
  - 選択状態にあるクローンセットのソースコードは簡単に閲覧可能



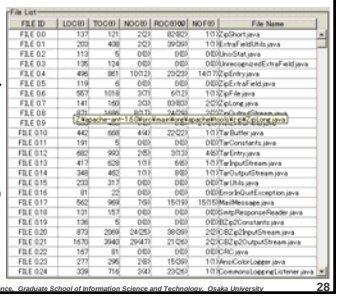
コードクローン分析ツール: Gemini
ファイルリスト(用いているメトリクス)

- 全てのメトリクスは閾値 *th* を用いている.
  - RNRの値が *th* 以上のコードクローンのみを対象にする
- ***ROC*$_{th}$*(F)*: ファイル *F*** の重複度を表す
  - 完全にクローンになっている場合は 1.0
  - 全くクローンを含まない場合は 0.0
  - $ROC_{0.5}(F_1) = 0.8$, $ROC_{0.5}(F_2) = 0.4$,
  - $ROC_{0.5}(F_3) = 1.0$, $ROC_{0.5}(F_4) = 0.6$
- ***NOC*$_{th}$*(F)*: *F*** に含まれるコードクローンの数を表す
  - $NOC_{0.5}(F_1) = 2$, $NOC_{0.5}(F_2) = 1$,
  - $NOC_{0.5}(F_3) = 2$, $NOC_{0.5}(F_4) = 1$
- ***NOF*$_{th}$*(F)*: *F*** とコードクローンを共有しているファイルの数を表す
  - $NOF_{0.5}(F_1) = 2$, $NOF_{0.5}(F_2) = 2$,
  - $NOF_{0.5}(F_3) = 3$, $NOF_{0.5}(F_4) = 1$

F1: a b c a b,
F2: c c* c* a b,
F3: d e f a b,
F4: c c* d e f

コードクローン分析ツール: Gemini
ファイルリスト(メカニズム)

- 対象ファイル一覧を表示
  - 各ファイルはファイル名・メトリクスと共に表示される
    - 括弧の外の値は閾値 *th* の値，括弧の中の値は閾値 0 の値
- ソーティング機能
  - ファイルを任意のメトリクスの昇順・降順で並び替え可能
  - 興味のある特徴をもったファイルをすぐに特定できる
- メトリクスグラフとして実装しなかった理由
  - クローンセットはメトリクス値のみに基いて選択される
  - ファイルはファイル名などでも選択されることがある



デモンストレーション

デモンストレーション(1/2)

- 対象
  - 大阪大学の学生演習プログラム
    - C言語で簡単なコンパイラを作成
    - 5人の学生のプログラムを用いる
    - 演習は3回に分けてプログラムを作成
      - ✓STEP1: 構文チェッカーを作成
      - ✓STEP2: 構文チェッカーを利用して意味チェッカーを作成
      - ✓STEP3: 意味チェッカーを利用してコンパイラを作成
- 目的
  - 学生間の類似度の調査
    - 他人のプログラムを利用している学生はいないか

デモンストレーション(2/2)

- １人の学生のソースコードを1つのグループとしてコードクローン検出を行う
  - ◆５人の学生のプログラムから5つのグループが作成される
  - ◆適切にグループを設定することでコードクローン分析を効率的に行うことができる
- ソースコードビューではコードクローンが強調表示される
  - ◆水平方向のコードクローン：青
  - ◆垂直方向のコードクローン：赤

これまでの適用事例

オープンソースソフトウェアへの適用 概要

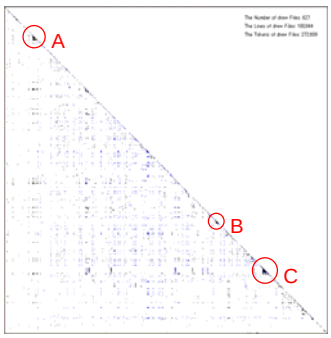
- 目的
  - ◆CCFinder，Geminiを用いてどのようなコードクローンが見つかるか調査する
- オープンソースソフトウェア Ant (version 1.6.0)
  - ◆ビルドツールの一種，Java言語で記述されている
  - ◆ソースファイル数: 627
  - ◆総行数: 約18万行
- 検出対象: 30トークン以上
  - ◆2,406個のクローンセット
  - ◆190,004個のクローンペア

オープンソースソフトウェアへの適用 調査するコードクローン

- スキャタープロットを用いた調査
  - ◆目立つ部分に存在するコードクローン
- メトリクスグラフを用いた調査
  - ◆ 要素数の多いクローンセットの特定
  - ◆ トークン数の多いクローンセットの特定
  - ◆ 多くのファイルを巻き込んでいるクローンセットの特定

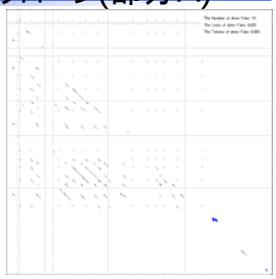
オープンソースソフトウェアへの適用 目立つ部分に存在するクローン(全体)

- 右図は対象ソースコード全体を表したクローン散布図
- A, B, Cの部分がどのようなコードクローンであるかを調査した



オープンソースソフトウェアへの適用 目立つ部分に存在するクローン(部分A)

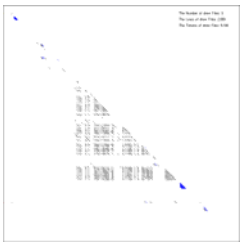
- クローンの場所: ファイルを読み込む機能を実装している部分
  - ◆ 先頭の数行のみを読み込み
  - ◆ ユーザが指定した文字列を含む行のみを読み込み
  - ◆ 各行にプレフィックスを付けて読み込み



- クローンが実装している機能:
  - ◆ ストリームから1文字読み込む．終端まできたら，それに応じた処理をする
  - ◆ 新しく java.io.Reader オブジェクトを生成し，それを返す

オープンソースソフトウェアへの適用
目立つ部分に存在するクローン(部分B)

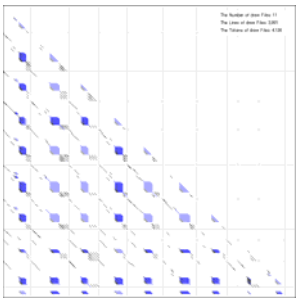
- クローンの場所: 簡単なGUIを実装しているファイル
  - ビルド情報をAntに渡す
  - Antの処理状況の閲覧



- クローンが実装している機能:
  - イベントがどこで起こったかを判定している if文
    - ➢イベントのソースに応じて処理を変更
  - GUIの部品を作成しているメソッド
    - ➢一つの部品につき，一つのメソッドが存在

オープンソースソフトウェアへの適用
目立つ部分に存在するクローン(部分C)

- クローンの場所: ClearCaseの各機能を実装しているファイル
  - Checkin,
  - Checkout,
  - Update,
- ファイルの特定の部分ではなく，ほぼ全体がクローンになっていた



オープンソースソフトウェアへの適用
要素数の多いクローンセット

- 予めRNRを用いて，その値が0.5未満のクローンセットは除外
- 最も要素数の多いクローンセット
  - 要素数:31個
  - クローンの場所: 簡単なGUIを実装しているファイル
  - クローンが実装している機能:GUIの部品を生成しているメソッド
  - 大まかな把握（Bの部分）のクローンの一部

```
        } catch (Throwable iExc) {
            handleException(iExc);
        }
    }
    return iAboutCommandPanel;
}

private Label getAboutContactLabel() {
    if (iAboutContactLabel == null) {
        try {
            iAboutContactLabel = new Label();
            iAboutContactLabel.setName("AboutContactLabel");
```


オープンソースソフトウェアへの適用
トークン数の多いクローンセット

- 予めRNRを用いて，その値が0.5未満のクローンセットは除外
- 最もトークン数の多いクローンセット
  - クローンの大きさ:282トークン（77行）
  - クローンの場所:WebLogicとWebShereのタスクを定義しているファイル
  - クローンが実装している機能:メソッド isRebuildRequired（引数で与えられたJarファイルがリビルドする必要があるかどうかを判断）
  - 一部の使用変数，メソッド名が異なる
  - インデント，空行，コメントなど他のコードスタイルが全く同じ
    - ➢コピーアンドペーストによる作成を示唆

オープンソースソフトウェアへの適用
多くのファイルを巻き込んでいるクローンセット

- 予めRNRを用いて，その値が0.5未満のクローンセットは除外
- 最も多くのファイルを巻き込んでいるクローンセット
- 巻き込んでいるファイル数:19ファイル
  - クローンの場所:さまざまなファイル
  - クローンが実装している機能:連続したアクセサ
  - Antだからではなく，Java言語で記述されているから存在しているクローン
- このクローンセットに限らず，多くのファイルを巻き込んでいるクローンセットの多くが，Java言語で記述されていることがその存在理由と思われた

ベンダーの開発したソフトウェアへの適用

- スキャタープロットを用いた分析
  - 予期しない部分間のコードクローンの発見
- メトリクスグラフを用いた分析
  - コピーアンドペースト後，修正漏れのあるコードクローンの発見
  - リファクタリングを行うべきと思われるコードクローンの発見
- ファイルリストを用いた分析
  - 使われていないファイルの検出
  - 同じ機能を実装しているファイルの検出

これまでの活動

これまでの活動
ツールの配布

- コードクローン検出・可視化ツール
  - 検出ツール: CCFinder[1]
  - 分析ツール: Gemini[2]
- 国内外の個人・組織に配布
  - 研究機関での利用
  - 企業での商用ソフトウェアの開発プロセスへの導入
  - 配布先からのフィードバックを得ている
- その他の利用
  - 大学の演習
  - プログラム著作権関係の裁判証拠

[1] 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*, 28(7):654-670, 2002.
[2] Y. Ueda, T. Kamiya, S. Kusumoto and K. Inoue, "Gemini: Maintenance Support Environment Based on Code Clone Analysis", *Proc. Of the 8th IEEE International Symposium on Software Metrics*, 67-76, 2002.

これまでの活動
コードクローンセミナー

- ツール開発者（大学）と利用者（産業界）との意見交換との場としてコードクローンセミナーを開催
  - 第1回（2002年11月）：大阪
  - 第2回（2003年3月）：東京
  - 第3回（2003年6月）：大阪
  - 第4回（2005年3月）：東京
  - 第5回（2005年12月）：東京
  - 第6回（2006年3月）：大阪
- ツールの利用法，コードクローンの分析法などについて議論を行っている
  - 詳しくはをhttp://sel.ist.osaka-u.ac.jp/kobo/を参照

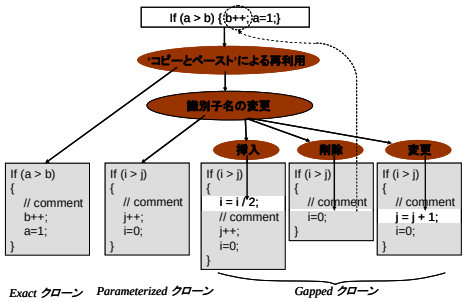
今後の展開

今後の展開
ソフトウェア開発プロセスへの組込み

- コードクローン検出技術をどのようにソフトウェア開発プロセスへ組み込むか
  - レビュー支援
  - リファクタリング支援
  - ソースコード修正支援
- 実プロジェクトでの使用・評価が必要

今後の展開
ギャップドコードクローンの検出

- コピーアンドペーストされたコード片は，その後修正が加えられる
- 現在検出できているのは，Exactクローン，Renamedクローンのみ



今後の展開
CCFinderXの開発

- 大阪大学井上研究室出身の神谷年洋氏（現：産業技術総合研究所勤務）による CCFinderX の開発
  - 2004年度第2回未踏ソフトウェア創造事業の支援
  - 天才プログラマー／スーパークリエータの認定を受ける
- CCFinderX の特徴
  - 検出スピードの向上
  - 前処理のカスタマイズ
  - SWTを用いたGUI
- CCFinderX ウェブページ
  - http://www.ccfinder.net/

最後に

- ツールに興味を持たれた方はメールでご連絡下さい
  - 大阪大学 肥後芳樹: y-higo@ist.osaka-u.ac.jp
- 現在，特許申請中につき，配布管理を行っています
- CCFinderXの配布管理は行っていません
  - ➢CCFinderXの配布については http://www.ccfinder.net/をご覧下さい