

コードクローン検出とその関連技術

(A Survey of Code Clone Detection and Its Related Techniques)

肥後 芳樹 楠本 真二 井上 克郎

2018年11月2日

60216087 木村 祐太

コードクローン

(Code Clone)

コードクローン

- ソースコード中に存在する互いに一致, または, 類似したコード片
- コードクローンの存在は, ソフトウェアの保守を困難にする

コードクローンの発生理由

- 既存コードのコピーアンドペーストによる再利用
- 定型処理
- プログラミング言語における適切な機能の欠如
- パフォーマンス改善

コードクローンの発生理由

- コード生成ツールの生成コード
- 複数のプラットフォームに対応したコード
- 偶然

コードクローンの定義

- 厳密で普遍的な定義は存在しない
- 3つのタイプに分類

コードクローンの定義

1. 空白やタブの有無，括弧の位置などのコーディングスタイルを除いて，完全に一致する
2. 変数名や関数名などのユーザ定義名，また変数の型などの一部の予約語のみが異なる
3. 2における変更に加えて，文の挿入や削除，変更が行われた

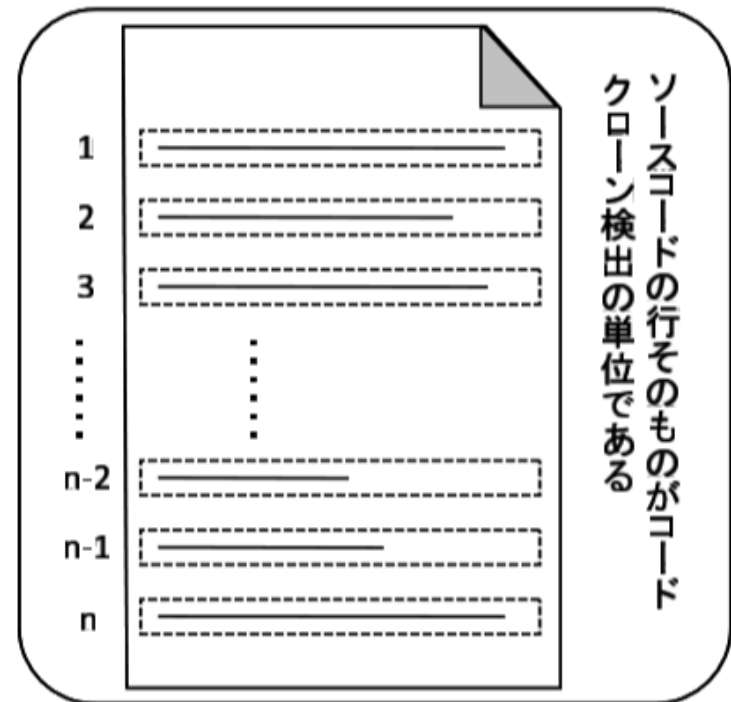
検出技術

検出技術

- 行単位の検出
- 字句単位の検出
- 抽象構文木を用いた検出
- プログラム依存グラフを用いた検出
- メトリックスやフィンガープリントなど、
そのほかの検出技術を用いた検出

行単位の検出

- ソースコードを行単位で比較し，重複コードを検出



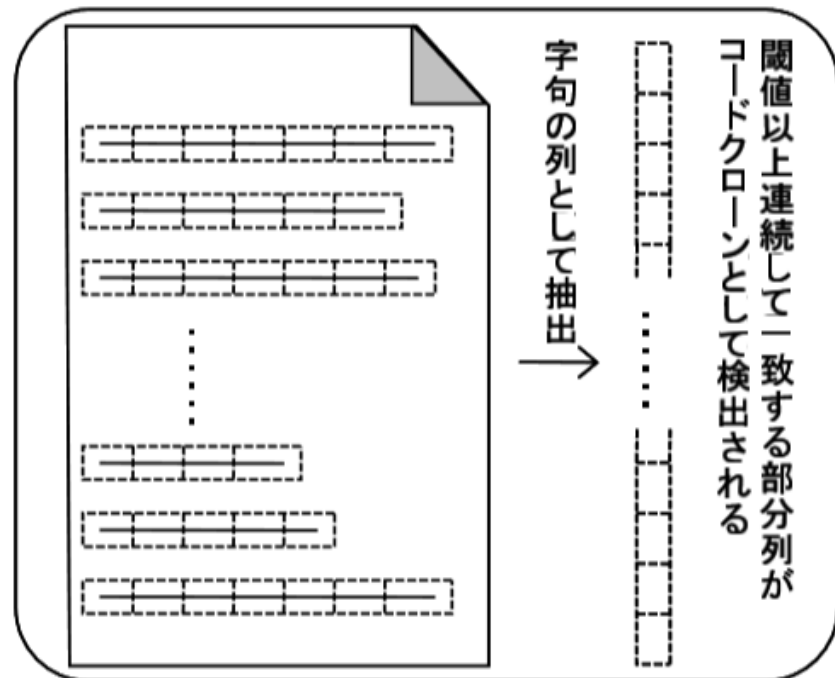
(a) 行単位の検出

検出技術

- 行単位の検出
- 字句単位の検出
- 抽象構文木を用いた検出
- プログラム依存グラフを用いた検出
- メトリックスやフィンガープリントなど、
そのほかの検出技術を用いた検出

字句単位の検出

- 前処理としてソースコードは字句の列に変換
- 連続して一致している字句の部分列を検出



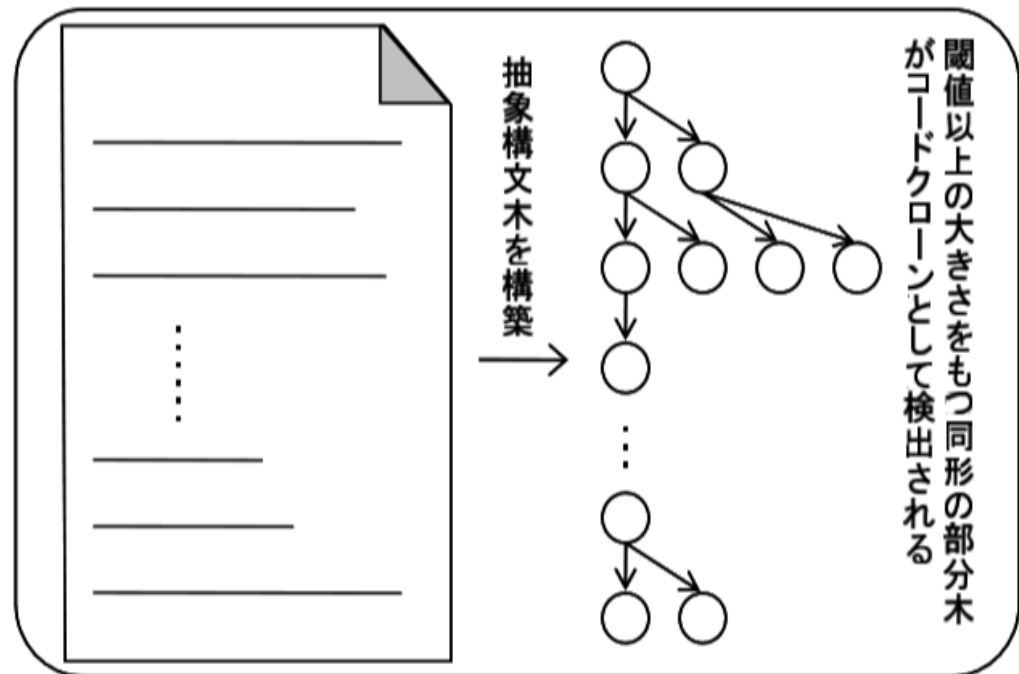
(b) 字句単位の検出

検出技術

- 行単位の検出
- 字句単位の検出
- 抽象構文木を用いた検出
- プログラム依存グラフを用いた検出
- メトリックスやフィンガープリントなど、
そのほかの検出技術を用いた検出

抽象構文木を用いた検出

- 前処理として抽象構文木を構築
- 抽象構文木上の同形の部分木をコードクローンとして検出



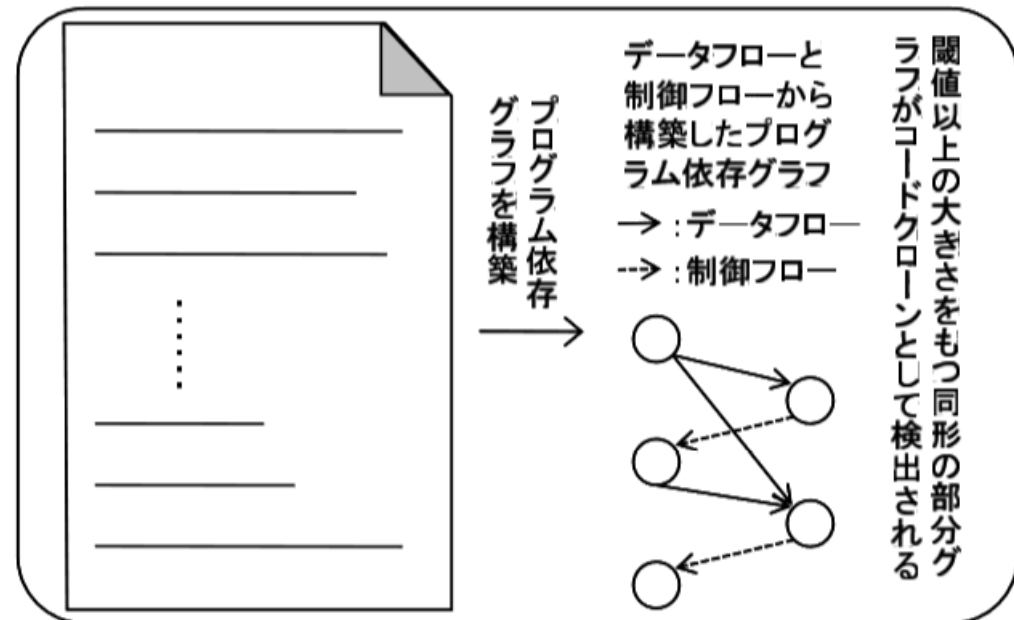
(c) 抽象構文木を用いた検出

検出技術

- 行単位の検出
- 字句単位の検出
- 抽象構文木を用いた検出
- プログラム依存グラフを用いた検出
- メトリックスやフィンガープリントなど、
そのほかの検出技術を用いた検出

プログラム依存グラフを用いた検出

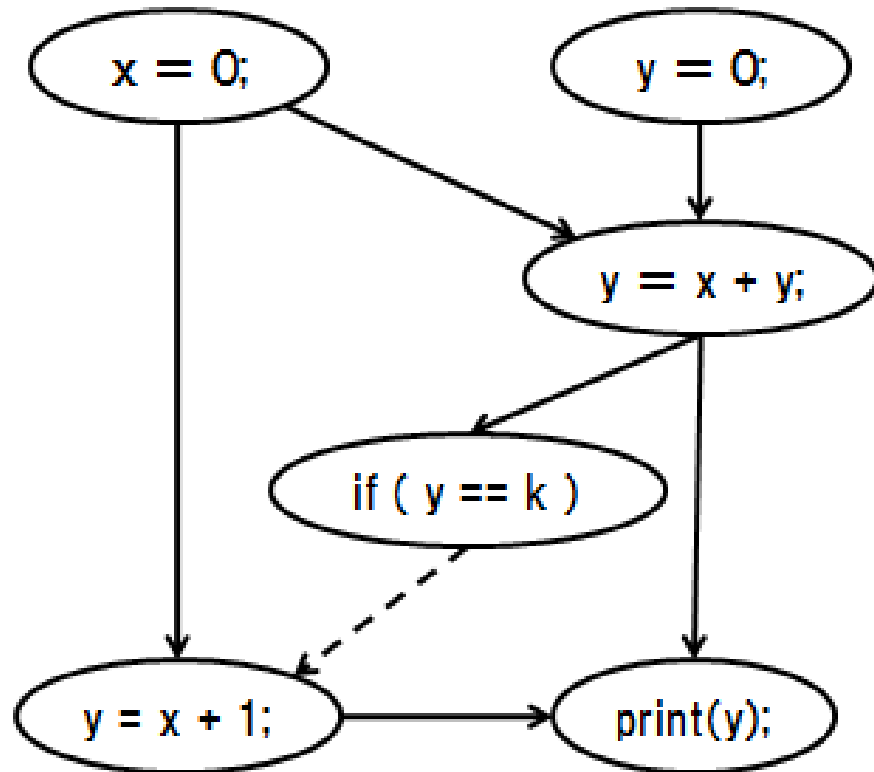
- 前処理として要素を頂点, 依存関係を有向辺とするグラフ (プログラム依存グラフ) が構築
- グラフ上の同形の部分グラフがコードクローンとして検出



(d) プログラム依存グラフを用いた検出

例：プログラム依存グラフ

```
x = 0;  
y = 0;  
y = x + y;  
if ( y == k ) {  
    y = x + 1;  
}  
print(y);
```



```

    fp3 = lookaheadset + tokensetsize;
    for (i = lookaheas(state) ; i < k ; i++){
bb      fp1 = LA + i * tokensetsize;
bb      fp2 = lookaheadset;
bb      while (fp2 < fp3)
bb          *fp2++ |= fp1++;
    }

```

(a) コード片 1

```

    fp3 = base + tokensetsize;
    ...
    if (rp) {
        while ((j = *rp++) >= 0) {
            ...
bb          fp1 = lookaheadset;
bb          fp2 = LA + j * tokensetsize;
bb          while (fp1 < fp3)
bb              *fp1++ |= *fp2++;
        }
    }

```

(b) コード片 2

図 3 順序入れ換わりコードクローン

```

    ...
bb  tmpa = UCHAR(*a),
##  tmpb = UCHAR(*b);
bb  while (blanks[tmpa])
bb      tmpa = UCHAR(*++a);
##  while (blanks[tmpb])
##      tmpb = UCHAR(*++b);
bb  if (tmpa == '-') {
        tmpa = UCHAR(*++a);
        ...
    }
##  else if (tmpb == '-') {
        if (...UCHAR(*++b)...) ...

```

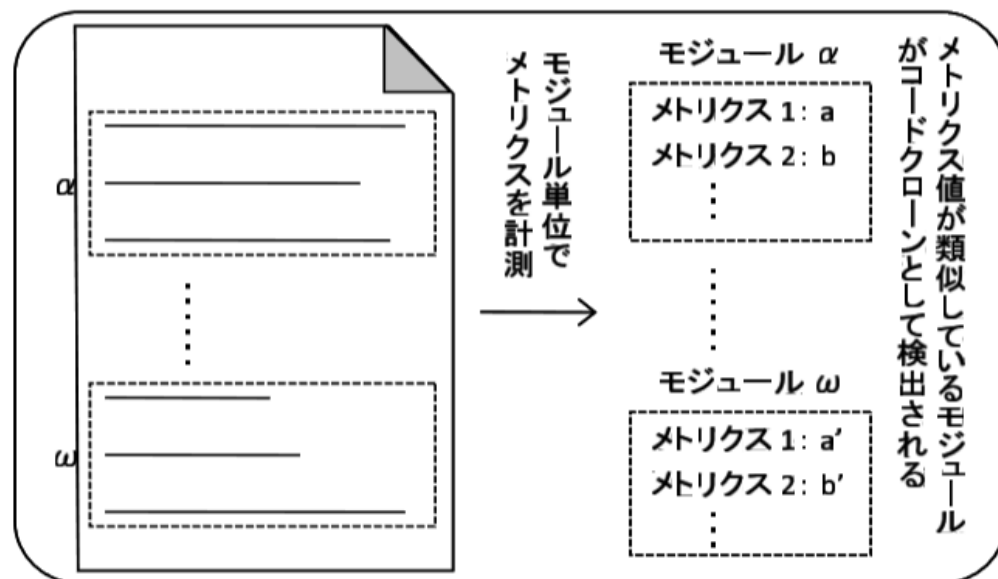
図 4 巻き付きコードクローン

検出技術

- 行単位の検出
- 字句単位の検出
- 抽象構文木を用いた検出
- プログラム依存グラフを用いた検出
- メトリックスやフィンガープリントなど、
そのほかの検出技術を用いた検出

メトリックスやフィンガープリントなど その他の技術を用いた検出

- プログラムのモジュールに対してメトリックスを計測し、その値の一致、または近似の度合を検査することによって、そのモジュール単位でのコードクローンを検出



(e) メトリックスを用いた検出

コードクローンの分析, 可視化

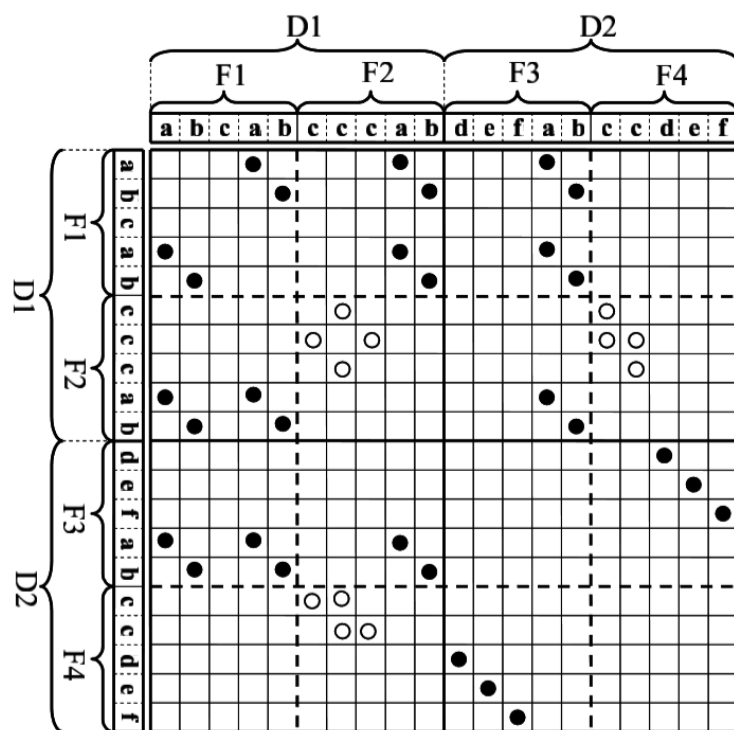
分析, 可視化を行う前に

- 調査の必要がないコードクローンの存在が障害
- コードクローンのフィルタリングは必須

分析, 可視化を行う前に

- コードクローンの非繰り返し度を表すメトリックス (RNR) を使用したフィルタリング
- コードクローンと関数や構造体などのプログラムの要素の対応付けを行いフィルタリング

Geminiの散布図



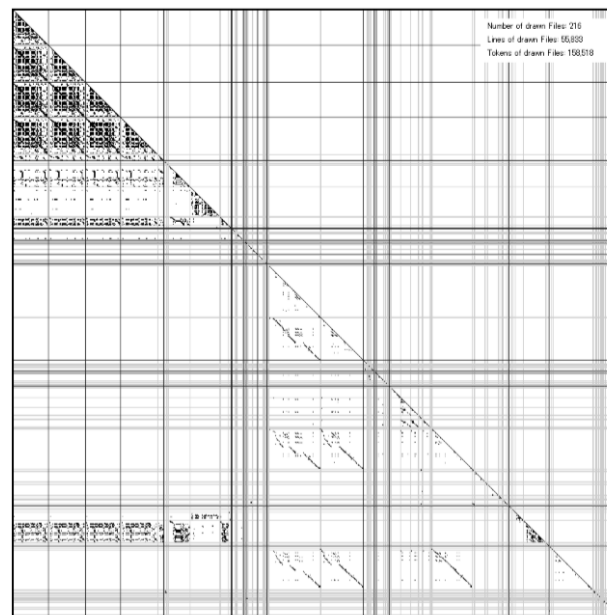
F1, F2, F3, F4 : ファイル

D1, D2 : ディレクトリ

● : RNRが閾値以上のコードクローンに含まれるトークン

○ : RNRが閾値未満のコードクローンに含まれるトークン

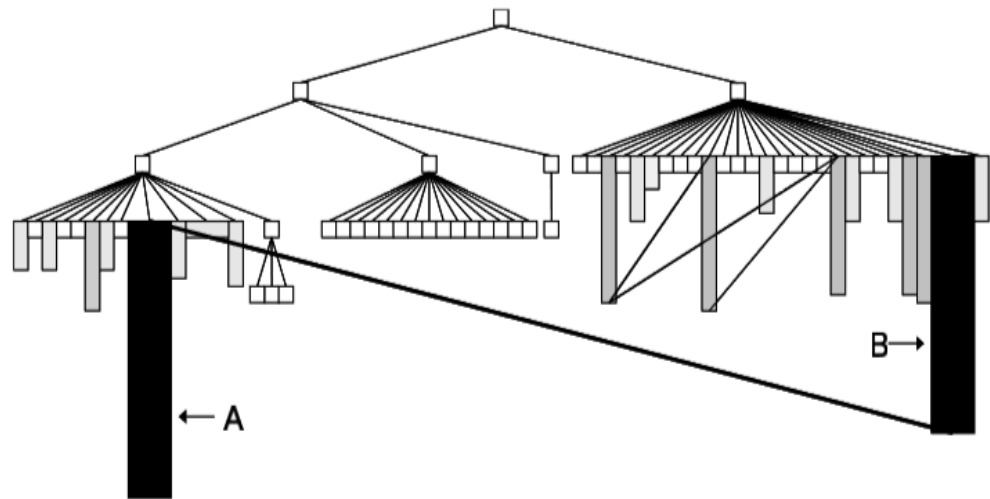
(a) 散布図のモデル



(b) 散布図を用いた可視化の例

Polymetric View

- 様々な粒度で抽象化
- ファイル内のコードクローン量やファイル間の共有コードクローン量など



Polymetric View を用いた可視化の例

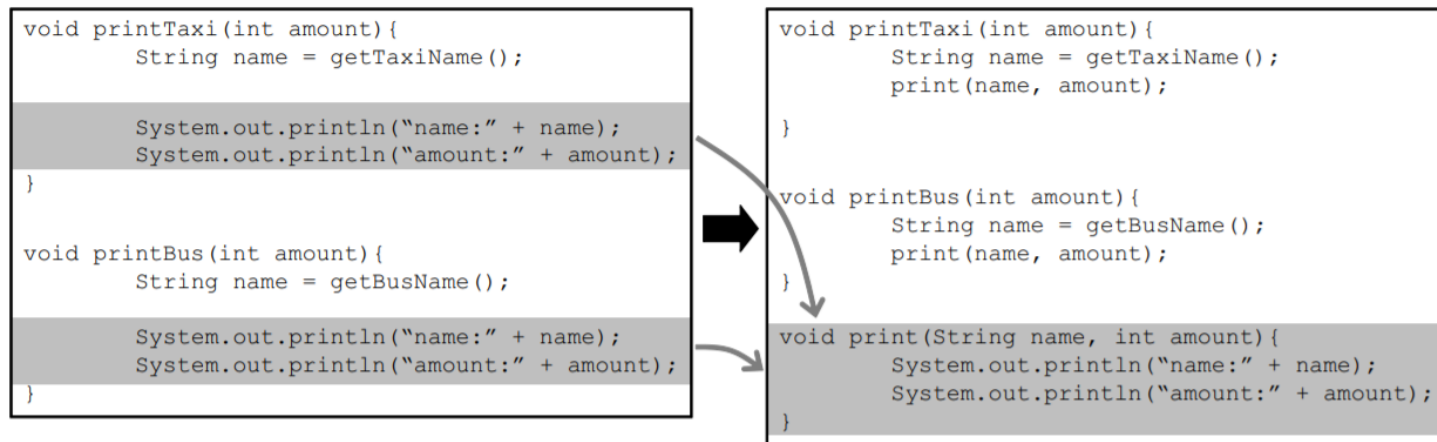
その他

- 複数のバージョンからコードクローンを検出して、その遷移の様子を表示するツール
- 複数のコンピュータ上でコードクローン検出を分散して行い、大規模ソースコード集合からの効率的なコードクローン検出及び可視化

コードクローンの集約

コードクローンの集約 (リファクタリング)

- 互いにコードクローンになっているコード片群を一つのモジュールにまとめること



リファクタリングの例

コードクローンの修正支援

- リファクタリングすべきでないものや抽象化機構が存在しない



- コピーとペーストの履歴の保存, コードクローンを検出し自動で書き換え

まとめ

まとめ

- コピペは一概に悪いとは言えない
- 重複するコードは可能な限り集約
- コードクローンに関する技術は難しい！！！！

参考

検出技術の分類	検出手法	検出可能なコードクローン			対応言語
		タイプ 1	タイプ 2	タイプ 3	
行単位の検出	Johnson の手法 [10]	○	-	-	不特定多数
	Ducasse らの手法 [11], [12]	○	-	-	不特定多数
	Baker の手法 [13]~[15]	○	○	-	不明
	Wettel ら手法 [16]	○	-	○	不特定多数
字句単位の検出	Prechelt らの手法 [17]~[19]	○	○	-	C/C++, Java, C#,Scheme
	Kamiya らの手法 [6]	○	○	△ ^(注1)	C/C++, Java, COBOL など
	Li らの手法 [20]	○	○	○	C/C++
	Basit らの手法 [21]	○	○	○	Java
抽象構文木を用いた検出	Baxter らの手法 [5]	○	○	-	C/C++, Java, Ada, C#など
	Koschke らの手法 [22]	○	○	-	C
	Jiang らの手法 [23]	○	○	○	C,Java
	Wahler らの手法 [24]	○	○	-	XML で表現されたソースコード
プログラム依存グラフを用いた検出	Komondoor らの手法 [25]	○	○	○	C/C++, Scheme
	Krinke の手法 [26]	○	○	○	C
メトリックスを用いた検出	Mayrand らの手法 [27]	○	○	○	C/C++
	Kontogiannis らの手法 [28], [29]	○	○	○	C

表 1 コードクローン検出技術とコードクローンタイプ

参考

検出手法	検出数	適合率	再現率	F 値
Kamiya らの手法 [6]	1,128	72	72	72.0
Baxter らの手法 [5]	84	100	9	16.5
Mayrand らの手法 [27]	278	63	19	29.2
Prechelt らの手法 [18]	131	82	12	20.9
Aiken の手法 [43]	120	73	10	17.6

Burd らの比較：各手法により検出されたコードクローン
の数の数，適合率，再現率

おまけ



肥後 芳樹 (正員)

平 14 阪大・基礎工・情報中退. 平 18 同大学院博士後期課程了. 平 19 阪大・情報・コンピュータサイエンス・助教. 博士(情報科学). コードクローン分析・リファクタリングに関する研究に従事. 情報処理学会, IEEE 各会員.



楠本 真二 (正員)

昭 63 阪大・基礎工・情報卒. 平 3 同大学院博士課程中退. 同年同大・基礎工・情報・助手. 平 8 同大講師. 平 11 同大助教授. 平 14 阪大・情報・コンピュータサイエンス・助教授. 平 17 同学科教授. 博士(工学). ソフトウェアの生産性や品質の定量的評価, プロジェクト管理に関する研究に従事. 情報処理学会, IEEE, JFPUG 各会員.



井上 克郎 (正員)

昭 54 阪大・基礎工・情報卒. 昭 59 同大学院博士課程了. 同年同大・基礎工・情報・助手. 昭 59~61 ハワイ大マノア校・情報工学科・助教授. 平元阪大・基礎工・情報・講師. 平 3 同学科・助教授. 平 7 同学科・教授. 博士(工学). 平 14 阪大・情報・コンピュータサイエンス・教授. ソフトウェア工学の研究に従事. 情報処理学会, 日本ソフトウェア科学会, IEEE, ACM, 各会員.