

コードクローン検出法

S161053

前田欣耶

目次

- 1、コードクローンの簡単な説明
- 2、コードクローンの定義
- 3、検出のための粒度
- 4、クローン発見マッチングアルゴリズム
- 5、手法の実装と実例
- 6、リファクタリング
- 7、コードクローンの評価用メトリクス
- 8、コードクローンの出現頻度
- 9、まとめ

1、コードクロンの簡単な説明

- コードクロンとはプログラムテキスト中の同一、あるいは類似したコードの断片

```
341:    dwFrameGroupLength = 1;
342:    for(dwCnt = 2; dwCnt <= 64; dwCnt *= 2)
343:    {
344:        if(((ulOutRate / dwCnt) * dwCnt) !=
345:            ulOutRate)
346:        {
347:            dwFrameGroupLength *= 2;
348:        }
349:    }
```

```
784:    frameGroupLength = 1;
785:    for (cnt = 2; cnt <= 64; cnt *= 2) {
786:        if (((rate / cnt) * cnt) != rate)
787:            frameGroupLength *= 2;
788:    }
```

図 1 変数名等が変更されているコードクロンの例

- コードクローンがプログラム中に存在していると修正、変更が難しくなる。(めんどい。)

対処法

・コードクロンの文書化：コードクローンに関する情報(どこに、どんな)が存在するかが文書化されている。
だが、全て修正、変更することには変わらないので実用的でない。
(めんどい。)

・**コードクロンの自動検出**：コードクローンを自動で検出する。
様々な手法で検出システムが提案されている。

- コードクローンを検出するためのツールには以下の特性が求められている

- ・大規模プログラムへの適用可能性：現実的な時間とメモリで検出できるか。

- ・コードクローンの識別：数多くのコードクローンが検出されるので何らかの特徴づけをして、フィルタリングする。

- ・コードクローンの有意性：現実的に意味を持たないコードクローンを取り除く。(例えば、2，3行程度のコードをコードクローンとみなさない)

- ・拡張性：コードクローンの検出対象のプログラミング言語に深く依存しない

2、コードクロンの定義

- ある系列(コード)中に存在する、二つの部分系列 α 、 β が等価だとする
 - $C(\alpha, \beta)$ と書き、 α は β のクローンであるという。
 - α 、 β の組をクローンペアという。
 - $C(\alpha, \beta)$ は、反射、推移、対象率を満たす。
 - α を含む同値類を α のクローンクラスという。

- ある系列(コード)中に存在する、二つの部分系列 α 、 β が等価だとする $\rightarrow C(\alpha, \beta)$
 - それぞれの部分系列 α' 、 β' に対して $C(\alpha', \beta')$ が成り立つ
 - ある系列中に存在する、二つの部分系列A、Bの部分系列が α 、 β である時、 $C(A, B)$ でなくかつ $C(\alpha, \beta)$ である時、 α と β を極大クローンペアという。
 - ある系列中に含まれる極大クローンペアを全て発見することをクローン発見問題といい、これを目的としている。
 - プログラムテキストが対象の基本だが、後で紹介する検出方法はHTMLや英文などにも適用可能。
(最適化は必要)

```
341:     dwFrameGroupLength = 1;
342:     for(dwCnt = 2; dwCnt <= 64; dwCnt *= 2)
343:     {
344:         if(((ulOutRate / dwCnt) * dwCnt) !=
345:            ulOutRate)
346:         {
347:             dwFrameGroupLength *= 2;
348:         }
349:     }

784:     frameGroupLength = 1;
785:     for (cnt = 2; cnt <= 64; cnt *= 2) {
786:         if (((rate / cnt) * cnt) != rate)
787:             frameGroupLength *= 2;
788:     }
```

図 1 変数名等が変更されているコードクローンの例

- 「等価」とは...
完全に同じものを等価としたい。
だが、空白、コメントの挿入,削除、識別子の名前変更
も等価とみなしたい。
なので字句、行、文レベルでの変換操作をする。

3、検出のための粒度

- 文字

プログラムテキストの構成要素を文字列とみなして、パターンマッチングを行う方法。

これは手間がかかりすぎるので不可能。

- 字句

字句解析を行った後、その字句を要素とした系列に対して、クローン発見問題を解く方法。

字句解析は、空白や改行の削除ができる。特定の種類の字句(識別子や定数)を一つの字句に固定化する。

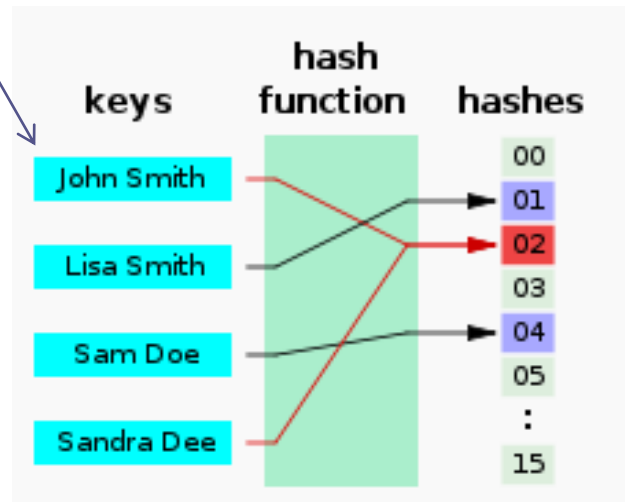
よってコードクローンとして認識できる。

- ・行

プログラムテキストの各行をハッシュ関数を用いて比較的小さな要素列(ハッシュ値の列)に圧縮。そのハッシュ値の列を対象としてクローン発見問題を解く方法。

ただし、あらかじめ空行の削除、改行位置の整形をしないといけない。

プログラムテキストの各行

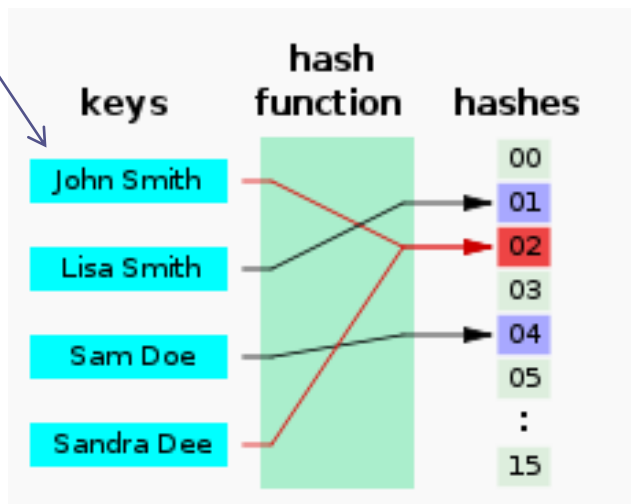


- ・ 文

プログラムテキスト中の文を取り出し、ハッシュ関数で一つの要素にしてクローン発見問題を解く方法。

文の認識をするために字句解析、構文解析をしないといけないが、空白や改行に影響を受けない。

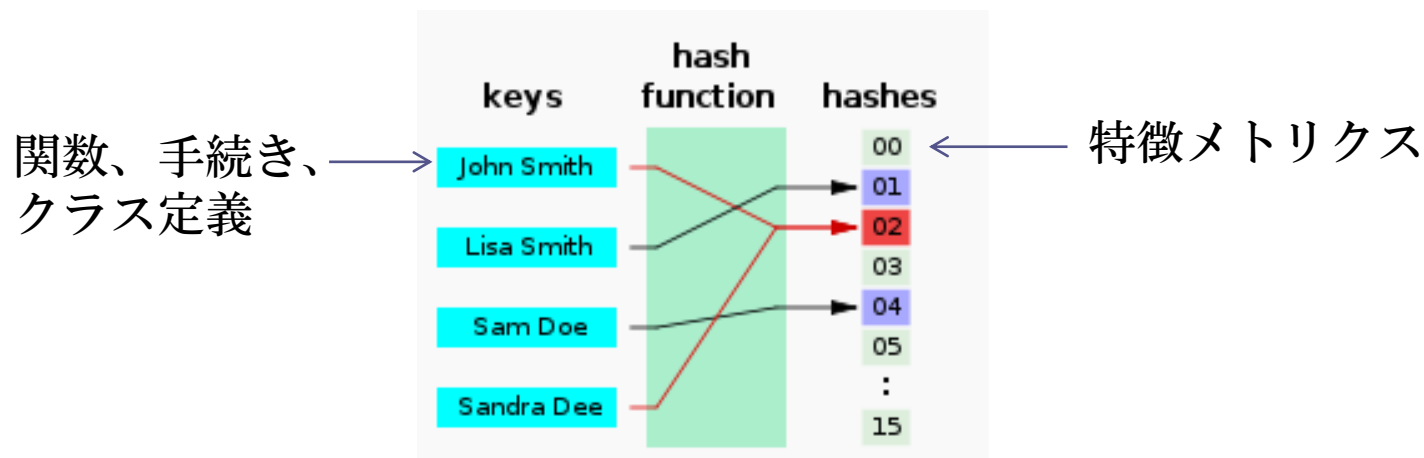
プログラムテキストの文



- 関数、手続き、クラス定義

プログラムテキスト中の関数、手続き、クラス定義を一つの要素とし、対になる要素を見つけることで、クローン発見問題を解く方法。

その中の一部がコードクローンになっていても見つけられない。



大規模なプログラムもコンパクトに系列に変換できるが、どんな特徴メトリクスを選択するかというのは難しい。

4、クローン発見マッチングアルゴリズム

- 表検索

対象となる系列となる各要素を縦と横に並べ、縦と横の要素が等しい場合に*そうでない場合に空白を埋める。

クローンは右下に下る線分として現れる。

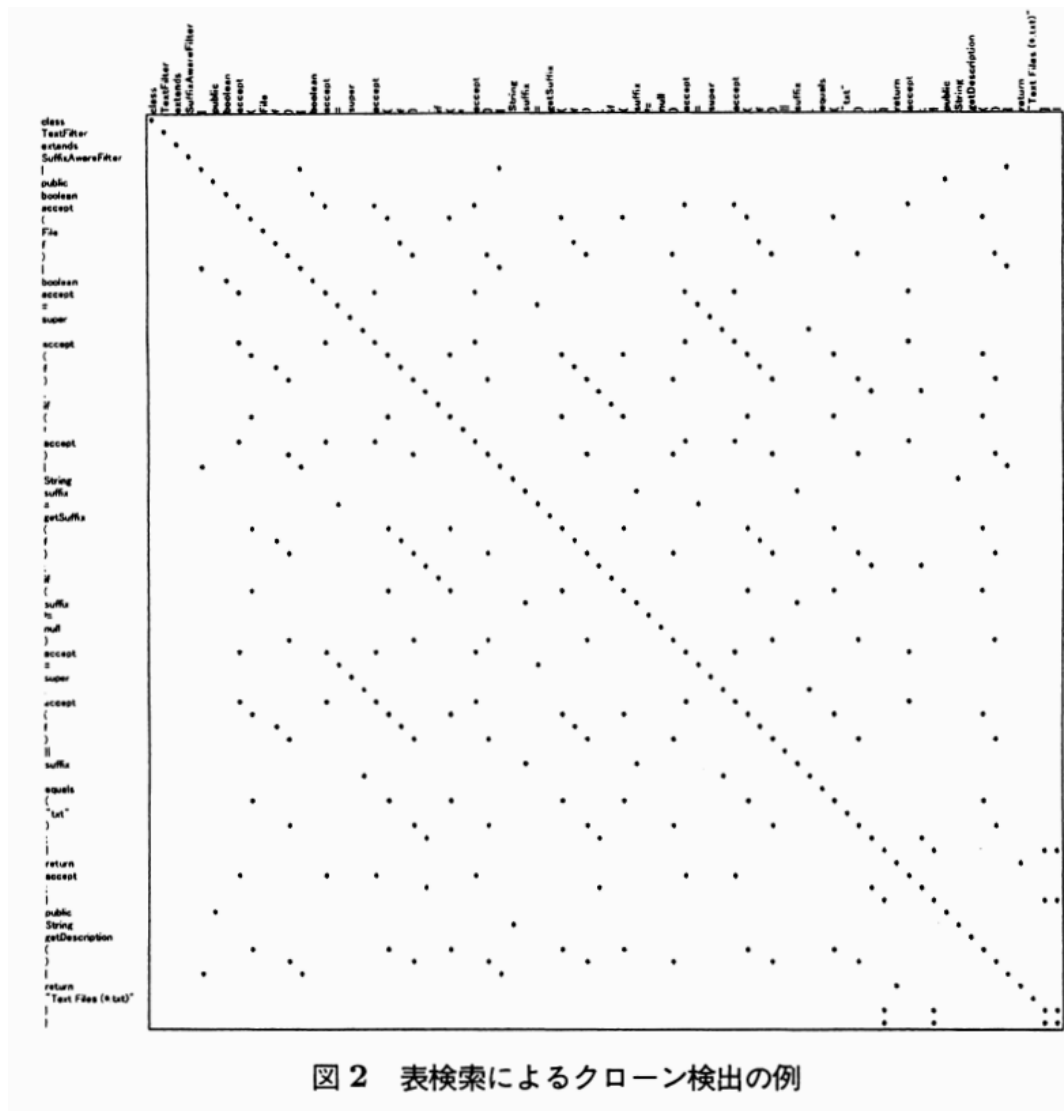
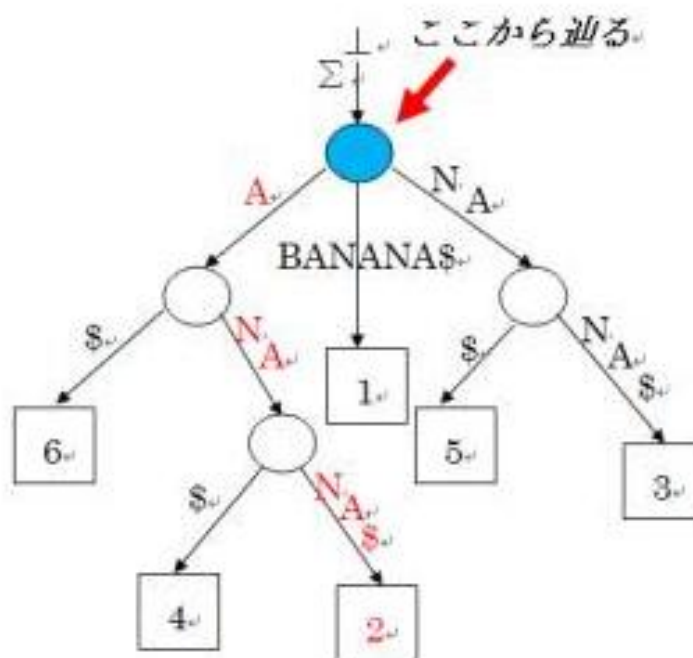


図 2 表検索によるクローン検出の例

- サフィックス木探索

サフィックス木に系列Sの全クローン情報を含む。二つの葉が共通の内部頂点を持つとき、**二つの葉はクローン**である。**この木を全探索**することでクローン発見問題を解く。

1	BANANA\$
2	ANANA\$
3	NANA\$
4	ANA\$
5	NA\$
6	A\$



サフィックス木の例

- ・ソーティング

メトリクスを用いた抽象化を行った場合、メトリクス値の要素をソートして等価なものを探す方法

5、手法の実装と実例

表 1 コードクローン検出ツールの比較

ツール名	検出方式 (粒度, マッチングアル ゴリズム, 計算複雑さ)	スケーラビリティ (プログラムテキストの規模, CPU, メモリ, 検出時間)	プログラミング言語への依存性 { 適用可能な言語 }
Dup [1] [2]	行, サフィックス木, $O(n)$	1.1MSLOC R3000 40MHz, 256MB RAM, 7 分	小 { C }
Duploc [7]	行, 表検索, $O(n^2)$	460KSLOC G3 230MHz, 100MB RAM, 6.5 時間	小 { C, COBOL, Python, Smalltalk }
CCFinder [11]	字句, サフィックス木, $O(n)$	10MSLOC PIII 650MHz, 1GB RAM, 68 分	中 { C, C++, Java, COBOL }
CloneDR [5]	AST の節点, 独自 (本文参照) $O(n^2)$	100KSLOC, (不明), 600MB RAM, 2 時間	大 { C, COBOL, Java }
SMC [3] [4]	メソッド ソーティング及び表検索 $O(n)$	215KSLOC, Pentium-Pro 180MHz, 64MB RAM, (不明)	大 { Java }

- Dup

前処理...ユーザ定義の識別子をパラメータ化。

- Duploc

前処理...空白、コメント、プログラム中に非常に多く出現する文字列を取り除く。

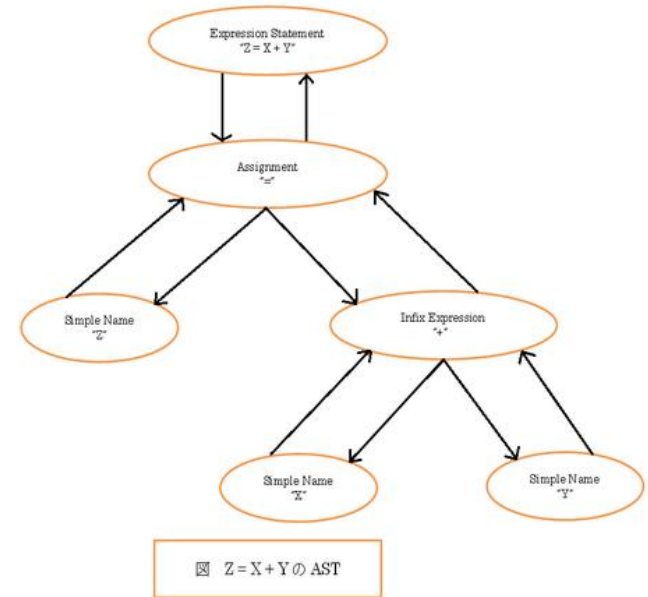
GUIを備えており、コードクロンの散文図を表示しそれに対応するプログラムテキストを編集する機能を持つ。

リファクタリングを目的とするシステムMOOSEの開発を計画中。

• CloneDR

AST...抽象構文木。通常の構文木から言語に意味のないものを取り除いたもの。

ASTの接点のペアをそれぞれ比較する。また構文解析によって、プログラミング言語に基づいた処理を行える。



- SMC

まず、特徴メトリクスによってコードクローンと思しきメソッドを見つける。次に表探索。その検出されたクローンペアのメソッドを、特徴(何が異なるか)により18種類に分けることができる。

- CCFinder

筆者らが2000年開発したツール。簡易な変形ルールを用いて、インデントや複文が変更されたプログラムテキストからコードクローンを検出することができる。

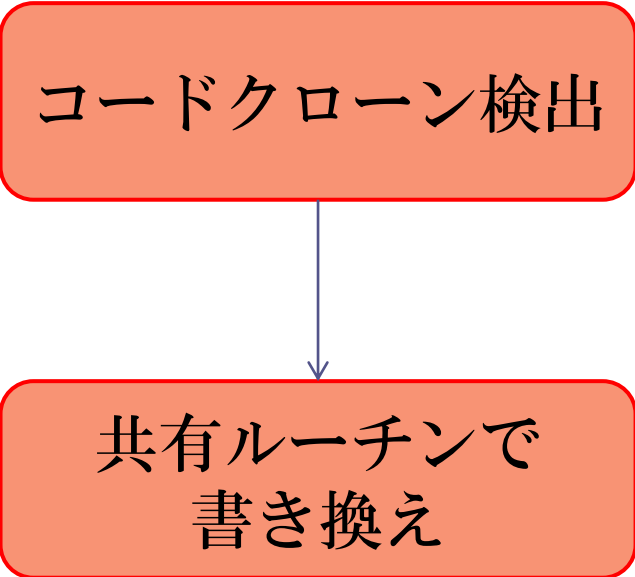
6、リファクタリング

まず、コードクローン検出。

次に、コードクローンを削除したい。
共有ルーチンで書き換え。

だが、書き換え前と同じふるまいをすることを保証する手法は確立されていない。

また、コードクローンを修正する際、コードクローンの存在を確認しておく必要がある。修正し忘れたコードクローンが存在すると、不具合が直っていないという事態がおきるから。

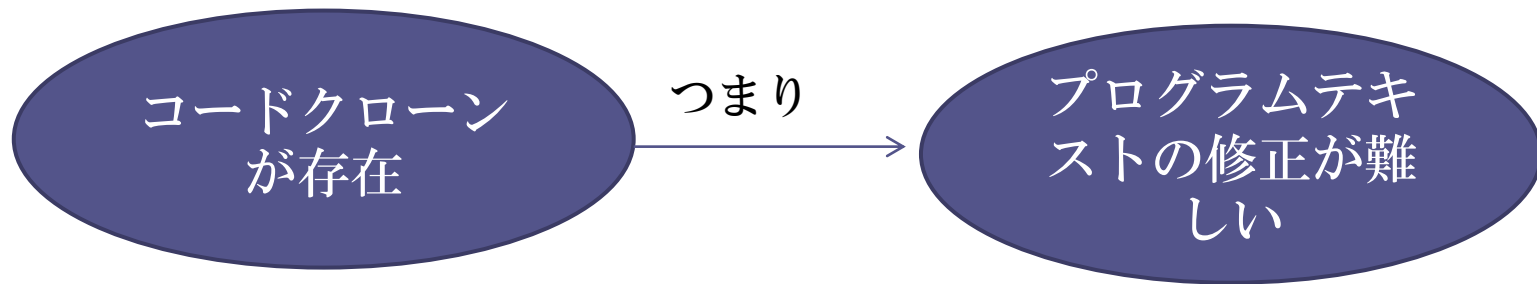


```
graph TD; A[コードクローン検出] --> B[共有ルーチンで書き換え]
```

コードクローン検出

共有ルーチンで
書き換え

7、コードクローンの評価用メトリクス



クローンを一種の保守性を表すメトリクスとして用いる立場がある。

8、コードクロンの出現頻度

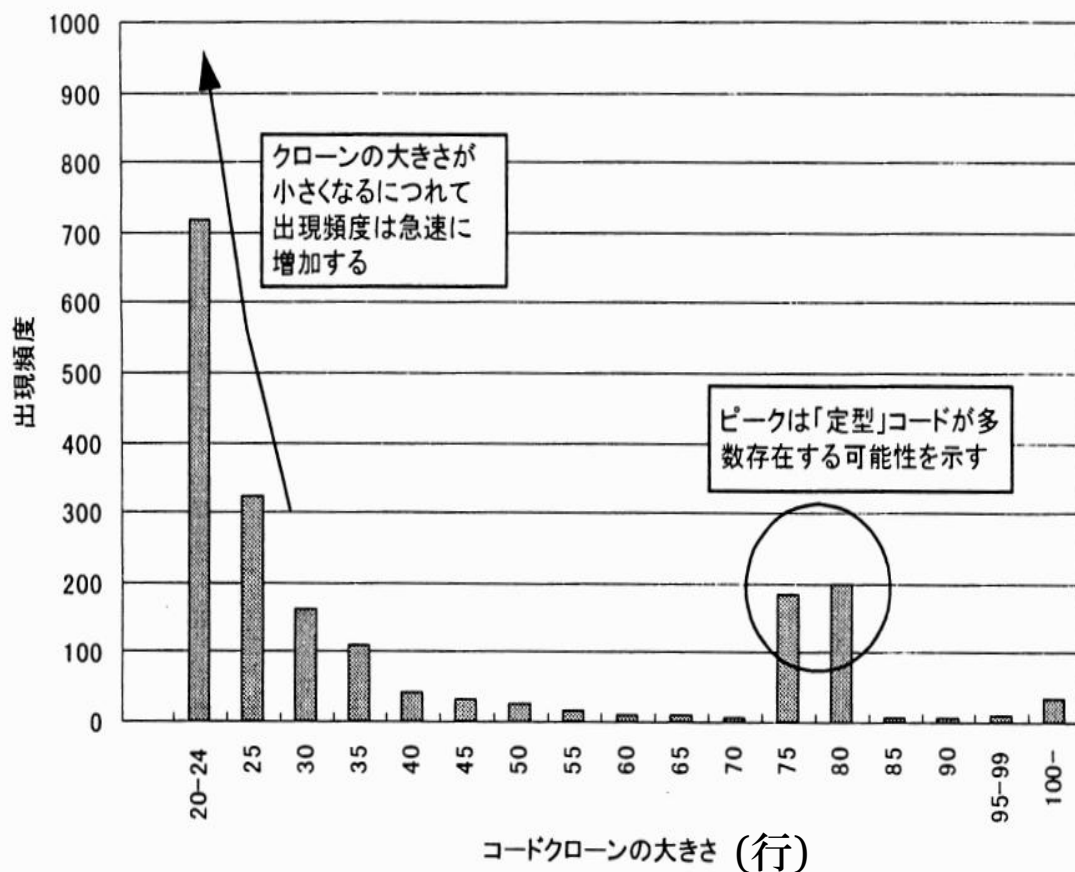


図3 コードクロンの典型的なヒストグラム

- 研究の目的に応じて検出する最小行数を決めている。

- 75,80でピークに。例えばテンプレートに少し修正を加えてコードを書く場合がこれに該当するであろう。

9、まとめ

- コードクローンとは、プログラムテキスト中の同一、あるいは類似したコードの断片。
- コードクローンを検出し、修正することが課題。
- 「字句、行などのレベルで検出」 → 「マッチング」 → 「書き換え、修正」という流れ。