システム保守のためのコードクローン検出に関する検討
A Study on Code Clone Detection for Software Maintenance

安藤未緒† 前田和昭‡ 高橋道郎‡
Mio Ando† Kazuaki Maeda‡ Michio Takahashi‡
† 中部大学大学院　経営情報学研究科　前期博士課程
‡ 中部大学　経営情報学部　経営情報学科
† Graduate School of Business Administration
and Information Science, Chubu Univ.
‡ Department of Business Administration
and Information Science, Chubu Univ.

要旨
情報システムの大規模化・複雑化に伴い,システムの保守作業に要する時間は増大し，多くの費用が必要となってきている．本論文では，システム保守を困難にしている要因の一つであるコードクローンに着目し，コードクローン検出技術の調査結果を報告する．さらには,入手可能なコードクローン検出ツールを使った経験について述べ，新しいツールの可能性について検討する．

1. はじめに

近年の情報システムの大規模化・複雑化に伴い，企業における既存システムの保守・デバッグ作業に要するコストは増大している．この保守作業を困難にする要因の一つにコードクローンがある．コードクローンとは，再利用を目的として，既存コードのコピー&ペーストや，コード生成ツールを使った自動生成などによって生まれた「ソースコード中の同一または類似した部分」と考えて良い．

コードクローンはソフトウェアを作成する上で避けて通ることはできない存在である．しかし，全てのコードクローンが問題となるわけではなく，また，コードクローンのタイプによって保守作業が違ってくる可能性もある．そのため，コードクローンを発見するだけではなく，それがどのようなタイプなのかを分析することが重要である．また，コードクローンの分布状況を分析し，何らかの視覚化ツールを提供できれば，システム保守のための助けとなることが期待できる．

2. コードクローン検出

コードクローンについての定義は未だ明確になっていない．論文[1]には,

A code clone is a code portion in source files that is identical or similar to another.

と述べられている．また，文献[2]では,

Code clones are segments of code that are similar according to some definition of similarity.

と述べられている．これらの記述から，コードクローンの定義は類似度に依存しているため，類似度の定義が変われば，異なったコードクローンが検出されることが分かる．

Bellon らの論文[3]では，コードクローンを以下のように分類している．

・タイプ１：空白やタブの有無，括弧の位置などを除いて，完全に一致するコードクローン．
・タイプ２：変数名や関数名などの識別子のみが異なるコードクローン．
・タイプ３：タイプ２に文の挿入・削除・変更が行われたコードクローン．

また, これまでに提案されたコードクローン検出技術は, その検出単位から以下のように分類できる[4].

<u>行単位の検出</u>

複数の連続した行同士を比較することでコードクローンを見つける．連続して一致している行が閾値以

上のときに，コードクローンと判断する．行単位の検出では，行の途中から始まるコードクローンの検出など，行以外の範囲での検出ができない．また，空白や改行位置を変更した場合，コードクローンとして判断できないことがあるため，前処理として，空白や改行位置の整形を行う必要がある．

字句単位の検出

複数の連続した字句同士を比較することでコードクローンを見つける．連続して一致している字句の列が閾値以上のときに，コードクローンと判断する．ソースコードを字句解析し，字句の列に変換してから比較を行うため，空白や改行位置の違いによって検出結果が変わることがない．しかし，字句単位の検出では構文情報を利用しないため，構文を無視したコードクローンが検出される可能性がある．

抽象構文木を用いた検出

抽象構文木の部分木同士を比較することでコードクローンを見つける．ソースコードを構文解析し，抽象構文木を構築してから比較を行うため，空白や改行位置の違いによって検出結果が変わることがない．また，検出されたコードクローンは，プログラムの構文上意味のあるものになる．部分木の比較には，ハッシングを使って共通部分式を見つけ出す方法[5]を使うことが多い．

プログラム依存グラフを用いた検出

ソースコードに対して意味解析を行うと同時に，プログラム依存グラフを構築し，そのグラフ中の部分グラフ同士を比較することでコードクローンを見つける．プログラム依存グラフを使えば，変数や式などの間の依存関係が分かるため，他の検出方法では判断できないコードクローンを見つけ出すことができる．

コードクローン検出に関する研究は，1990 年代から継続して行われ，これまでに数多くの研究論文が発表されている．発表済みの研究論文リストを知りたい場合は，アラバマ大学の Web サイト[6]を参照すると良い．また，検出ツールを使って実験した結果をまとめた論文[3]やサーベイ論文[4][7]では，検出技術に関する比較が効果的にまとめられている．

3. 代表的なコードクローン検出ツール

ここでは，代表的なコードクローン検出ツールとして，CCFinder と CloneDR について述べる．

3.1. CCFinder

CCFinder では，字句単位の検出を行う．コードクローン検出のために，接尾辞検索アルゴリズムを用いる．検出速度が速く，100 万行程度のソースコードに対して数分で検出が可能である．また，コードクローンの分布状況を視覚的に表示するための機能が提供されている．検出前に，閾値となる最小字句数を設定することにより，検出するコードクローンの大きさを調整できる．対象とするプログラミング言語は，C/C++，COBOL，Java，FORTRAN である．

検出の流れは以下のようになっている[1]．

- 入力されたソースコードを字句列に変換する．
- パターンマッチングと変形ルールを用いて前処理を行う．具体的には，名前空間の除去，ユーザ定義名の置換，テーブル初期化部分の除去，モジュール区切りの認識などである．
- 前処理を終えた字句列から接尾辞木を構築する．
- 構築した接尾辞木の中に，根から葉までの経路が同じとなる複数の字句列が存在するとき，コードクローンと判断する．

3.2. CloneDR

CloneDR では，抽象構文木を用いた検出を行う．抽象構文木を構築し，その部分木からハッシュ値を計算し，同一のハッシュ値の部分木のみを比較することで検出を高速化している．また，部分的に異なっているコードクローンも検出することが可能であり，検出したコードクローンを自動的に等価なサブルーチンやマクロに置き換えることも可能である．CloneDR では，検出すべきコードクローンの類似度

を自由に調整できる．対象としているプログラミング言語は，C++，COBOL，Java である．

3.3.　検出ツールの比較

CCFinder の改良版である CCFinderX（入手先[9]）と，CloneDR（入手先[10]）を使い，同じソースコードを対象としてコードクローン検出を実行し，その結果を比較する．

使用マシン：　Apple 社 iMac（CPU：Intel Core 2 Duo 2.66GHz，メモリ：　2GB）

オペレーティングシステム：Microsoft Windows XP Professional SP3

対象ソースコード：JRuby バージョン 1.0.1（入手先[11]），ファイル数：700，全行数：138,065

CCFinderX

コードクローン検出を行った結果，1,537 個のクローンセットが検出された．クローンセットとは，類似したコードクローンをまとめて一単位としたものである．検出結果は，散布図，ファイルテーブル，クローンセットテーブル，ソーステキストで表示できる．CCFinderX の実行例を図 1 に示す．図 1 の右側が散布図であり，コードクローンがどのように分布しているかをグラフで表現している．縦軸・横軸ともに，ソースコード内の字句列を同じ順番で並べたもので，灰色の垂直線・水平線がファイルの境界を表す．検出したコードクローンは，グラフ上で該当する座標位置に線や点を使って表す．　図 1 の左上に表示されているタブをクリックすると，ファイルテーブルとクローンセットテーブルを切り替えることができる．ファイルテーブルでは，対象となったソースコードのファイル一覧を示し，ファイルごとに含まれるクローンセット数が表示される．クローンセットテーブルでは，検出されたコードクローンの一覧を示し，それぞれのコードクローンの長さが表示される．ソーステキストでは，指定したファイルのテキストを表示し，検出したコードクローンを別の色で示す．CCFinderX では，２つのファイルを同時に表示しながら，画面上でコードクローンを確認することができるものの，３つ以上のファイル間に存在するコードクローンのテキストを一度に確認することができない．そのかわり，特定のコードクローンが存在するファイル数などの数値データを提示する．

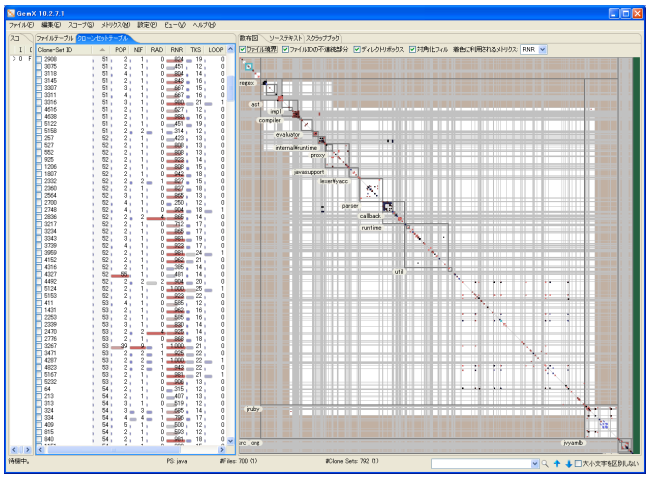

図 1　CCFinderX の実行例

CloneDR

コードクローン検出を行った結果，6,108 個のクローンセットが検出された．検出結果は，HTML 文書として書き出される．CloneDR の実行例を図２に示す．出力された HTML 文書の目次には，検出対象ソースコードのファイル数や行数，行ごとのコードクローン数，検出できたクローンセットの一覧が載っている．そこからのリンクを辿って，図２にある各クローンセット情報が掲載されたページを表示できる．CCFinderX がファイルを中心としてコードクローンを表示することに対して，CloneDR はクローンセットを中心としてコードクローンを表示する．特定のクローンセットに対する各コード片が，図２の下部のように表示される．CCFinderX とは異なり，

Clone Instance (Click to see clone)	Line Count	Source Line	Source File
1	10	48	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/ZSuperNode.java
2	10	50	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/ZArrayNode.java
3	10	56	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/ArrayNode.java
4	11	50	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/BlockNode.java
5	10	46	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/RetryNode.java
6	10	46	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/RedoNode.java
7	10	47	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/DStrNode.java
8	6	52	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/DSymbolNode.java
9	10	52	C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/DXStrNode.java

Clone Instance 1 Line Count: 10 Source Line: 48 File: C:/Documents and Settings/ando/My Documents/jruby-src-1.0.1/jruby-1.0.1/src/org/jruby/ast/ZSuperNode.java

```
public ZSuperNode(ISourcePosition position) {
    super(position, NodeTypes.ZSUPERNODE);
}

/**
 * Accept for the visitor pattern.
 * @param iVisitor the visitor
 **/
public Instruction accept(NodeVisitor iVisitor) {
    return iVisitor.visitZSuperNode(this );
}
```

図 2　CloneDR の実行例

検出結果をテキストだけで表示しているものの，全体がまとまっていて見やすく工夫されている．

4. 検討

二つの検出ツールを使用した経験を踏まえ，コードクローンの表示形式について検討する．CCFinderX は，散布図，ファイルテーブル，クローンセットテーブル，ソーステキストといった表示形式が関連し合い，ファイルを中心にして，いろいろな形式でコードクローンを表示する．しかし，ソーステキストを眺めるときには，各ファイルのソーステキストをそのまま表示しているため，コードクローンの該当部分だけを抜き出して比較することができない．CloneDR では，クローンセットごとにまとめた検出結果が書き出されるため, 検出したコードクローンに該当する部分だけを集中して確認することができる．しかし，ファイルに含まれるコードクローンの割合といったファイルごとの情報は表示されない．

このように，コードクローン検出には複数の技術があり，また，検出結果の表示にも複数の形式があることから，コードクローン検出結果を保管するデータ形式を標準化し，標準化されたデータ形式から多様な表示を可能にするツールを提供すべきであろう．

コードクローン検出技術は，行単位の検出から始まり，字句解析と構文解析，さらに静的解析を進めてプログラム依存グラフを用いた検出へと進展していった．静的解析をより深く進めれば，より多様なコードクローンの検出が可能になる．しかし, Java や C#などに代表される最近のプログラミング言語は, プログラミングのための記述能力が豊富であり，必要とするプログラミング言語の仕様にしたがって，静的解析のためのツールを開発するコストは大きい. また, 短期間に言語仕様が拡張されるケースも多々見られるため，それに追従して静的解析のためのツールも保守が必要となる．

以上から，開発コストがなるべく大きくならないコードクローン検出技術を実現するために，行単位と字句単位を土台として，構文情報を有効に取り入れ，複数のプログラミング言語に対応可能で，しかも，検出結果を標準化されたデータ形式で書き出すことで多様な表示が可能になるような，コードクローン検出ツールが望ましいのではないだろうか.

参考文献

[1] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue, “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code,” IEEE Transactions on Software Engineering, Vol.28, No.7, 2002, pp.654-670.

[2] Rainer Koscheke, “Identifying and Removing Software Clones,” Software Evolution, Springer-Verlag, 2008, pp.15-36.

[3] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, Ettore Merlo, “Comparison and Evaluation of Clone Detection Tools,” IEEE Transactions on Software Engineering, Vol.33, No.9, 2007, pp.577-591.

[4] 肥後芳樹，楠本信二，井上克郎, “コードクローン検出とその関連技術”, 電子情報通信学会論文誌, Vol.J91-D, No.6, 2008, pp.1465-1481.

[5] A.V.エイホ，M.S.ラム，R.セシィ 他，コンパイラ −原理・技法・ツール−, サイエンス社, 2009.

[6] Code Clones Literature, < http://students.cis.uab.edu/tairasr/clones/literature/> (accessed 2009-11-14).

[7] Chanchal K. Roy, James R. Cordy, Rainer Koschke, “Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach,” Science of Computer Programming, Vol.74, Issue 7, 2009, pp.470-495.

[8] Ira D. Baxter, Andrew Yahin, Leonardo Moura, et al., “Clone Detection Using Abstract Syntax Trees,” 14th IEEE International Conference on Software Maintenance, 1998, pp. 368-377.

[9] CCFinder Official Site, <http://www.ccfinder.net/> (accessed 2009-11-14).

[10] Semantic Designs, Inc: Clone Doctor, <http://www.semdesigns.com/Products/Clone/> (accessed 2009-11-14).

[11] JRuby.org :: Home <http://www.jruby.org/> (accessed 2009-11-14).