バージョン間のコードクローン関係の変化を提示するシステムの試作

2009SE160 松坂太智　2009SE223 岡田直希　2009SE241 坂下智紀
指導教員：横森励士

1　はじめに

ソフトウェア開発においては，ソフトウェアの高品質高機能な状態を保つために，日々，保守活動が行われている．このような保守活動では，不具合の修正以外にも性能などの品質向上や，これから生じる機能拡張への対応などが求められている．ソフトウェアの保守を困難にする要因の 1 つとして，コードクローンが指摘される．コードクローンとは，既存のソースコード (の一部) を複製することによって作られたコードを指す．コードクローンが存在する部分に対して修正を加えた場合，他のコードクローンへの修正が必要かどうかを検討しなくてはならないなど，コードクローンの存在により保守活動が複雑化しやすくなることが知られている．効率的に保守活動を行うには，コードクローンの観点からの分析が必要不可欠であるが，現状のシステムは 1 つのソフトウェア内にどのようにクローンが存在するかに着目したもので，ソフトウェアの修正によりどのようにクローン関係が変わったかを見せるという観点は存在しない．

本研究では，クローン関係が開発を通じてどのように変化したかについて全体的な情報を提示するツールを提案する．バージョンごとに分析されたクローン情報を読み込み，クラスやパッケージ単位でのクローン関係の変化を表やグラフで表示する．実際の適用例を元に表示された結果の表やグラフがどのように役に立つかを説明する．これらを通じて，本ツールが直接の開発者だけでなく，開発に参加する人全体で広く情報を共有するのに役立つことを確認する．

2　背景技術

2.1　コードクローン

コードクローンとは，新しい機能を追加するなどプログラミングを行う時，既存のソースコードの一部を複製することによって作られたコードである [1]．コピーした既存のコードを一部修正することで，新しい機能を追加するというアプローチは，一時的にはその場しのぎとしてプログラミング作業の軽減につながるが，不具合のある既存のコードにコードクローンが存在する場合，すべてのクローンに対して修正の必要性を検討する必要がある．その結果，このように作られたコードクローンは保守作業を困難にする．コードクローンの分析ツールとして CCFinder，視覚化して表示するためのツールとして Gemini が存在し，以下これらのツールについて説明する．

2.2　CCFinder

CCFinder[1] とは，コードクローンを検出するツールである．ソースコードをトークン単位で直接比較することによって，トークンの種類が連続して一致する部分をコードクローンとして検出する．トークンの種類をもとに判定するので，CCFinder は変数名や関数名などの異なるコード片も，コードクローンとして検出する．また，CCFinder の検出処理は非常に高速であり，数百万行規模のシステムを実用的な時間でコードクローンを検出することができる．

2.3　Gemini

Gemini[2] とは，CCFinder が出力したテキストファイルを読み込み，検出したコードクローンの位置やクローン関係をグラフなどで視覚的に表示するツールである．この Gemini では，単一のシステムにおけるコードクローン関係についての視覚化をクローン散布図やメトリクスグラフを用いて実現している．

2.4　EPM

EPM[3] は，リアルタイムでのプロジェクト管理を目的とした開発データの自動収集分析システムである．現在広く普及している開発支援フリーウェア (CVS，Mailman，GNATS 等) と連携することによって開発履歴データを自動的に収集し，それらの情報を元に一貫性のあるデータに基づいて分析結果をグラフや表として表示する．これにより，直接の開発者だけでなく，開発に参加する人全体が広く進捗情報を共有するのに役立つ．

3　コードクローン視覚化ツールの提案

3.1　既存のツールの問題点

ソフトウェア保守活動において，ソフトウェア開発の進捗状況を定量的なデータを用いて随時把握し，管理を行うことは極めて重要である．このようなツールとして EPM があり，1 つのソフトウェアシステムの各時点における情報を見せることで管理を行うことを提案している．ただし，EPM ではソースコードに対しての分析が LOC のみであるなど，機能が不十分である．また，コードクローン関係の分析で行われる Gemini や CCFinder といった既存のツールは，1 つのソフトウェアシステムに対する分析としては有益であるが，複数バージョンを考慮する分析は行っていない．情報を広く共有することを目的とした複数バージョンを考慮したコードクローン関係の分析ツールを新たに開発し，クローン関係の変化を見せる環境を構築することが必要である．

3.2　提案するツール

本研究では，ある 1 つのソフトウェアシステムの中にあるクローン関係が開発を通じてどのように変化したかを分析するツールを提案する．それぞれのバージョン毎に

クローン関係を分析し，必要な情報を読み込んだ上で，それぞれのクラスに存在するクローン関係をまとめる．結果は，バージョン間，パッケージ内部間，ファイル間などの粒度で表やグラフとして表示を行う．このようなツールを用いることで，コードクローンの分布の変化状況を直観的に把握することが可能となり，例えばリファクタリングの必要性が出てきた場合などの判断基準を広く共有するのに役立つ．

4　提案するコードクローン視覚化ツールの構成

4.1　ツールの全体像

提案するツールの全体の構成は図 1 のようになっている．処理の流れを以下に示す．

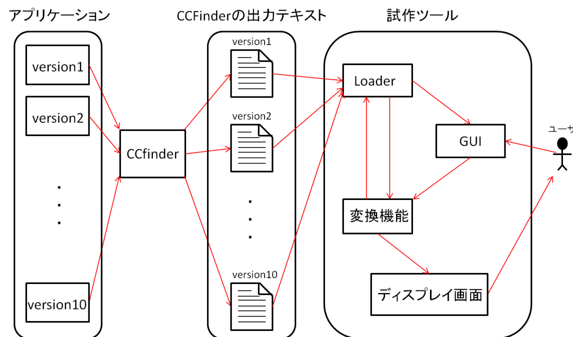

図 1　イメージ図

1. ソフトウェアをバージョン毎に CCFinder で解析する．
2. 出力された各情報を試作ツールで読み込む．
3. Loader で必要なクローン情報を読み，データを集計する．
4. ユーザーの GUI 操作で変換機能から Loader にアクセスして必要な情報を参照する．
5. Loader から参照した情報を変換機能で変換しディスプレイ画面にコードクローンを視覚化したものを表示する．

4.2　情報の抽出

CCFinder で解析されたテキスト情報は図 2，図 3 のように出力される．図 2 は，解析情報の前半部分を示したものである．#vertion では使用した CCFinder のバージョンが記述されている．#format，#langspec，#option には CCFinder 実行時の設定情報がまとめられている．今回，この情報は必要ないので，抜き出しは行っていない．次に#begin{file discription} から#end{file discription} の間では，全ての Java ファイルについての記述がされている．1 つの Java ファイルに対して，グループ ID，ファイル ID，プログラム行数，トークンの数，ファイルパスが表示されている．必要な情報として，各 Java ファイルに割り当てられた (グループ，ファイル)ID とファイルパスの 2 つを抜き出す．
図 3 は，解析情報の後半部分を示したものである．

```
#version: ccfinder 7.3.2
#format: classwise
#langspec: JAVA
#option: -b 30
#option: -e char
#option: -k 30
#option: -r abcdfikmnoprsuv
#option: -c w-fg
#option: -y
#begin{file description}
0.0      643      1724
C:¥CCFinder¥target¥Jetrix710¥0.1.0¥src¥java¥net¥jetrix¥Channel.java
0.1      125      220
C:¥CCFinder¥target¥Jetrix710¥0.1.0¥src¥java¥net¥jetrix¥ChannelManager.java
0.2      260      573
C:¥CCFinder¥target¥Jetrix710¥0.1.0¥src¥java¥net¥jetrix¥Client.java
        ・
        ・
        ・
etProtocolTest.java
#end{file description}
#begin{syntax error}
#end{syntax error}
```

使用しているCCfinderのバージョン及び,実行時の設定の表示
解析対象のファイル群の情報
(グループ,ファイル)ID・ファイル行数・トークン数
ファイルパス
変換している際に発生した構文エラーの位置情報で、ここに書き出された行は，クローン検出の対象外となっている.

図 2　CCFinder における出力結果 1

#begin{clone} から#end{clone} の間ではクローンの情報が書き出される．この中に#begin{set}，#end{set} が記述されており，これは 1 つのクローンセットの情報をまとめたものである．ここではグループ ID，ファイル ID，コードクローンの開始行，開始カラム，開始トークン，終了行，終了カラム，終了トークン，繰り返し処理ではないトークン数というフォーマットで書かれている．図の場合では (グループ，ファイル)ID が 0.3，0.5，0.127 のファイルがクローン関係にある．これらのクローン情報を抜き出す．

```
#begin{syntax error}
#end{syntax error}
#begin{clone}
#begin{set}
0.3      42,13,22      67,34,62   3
0.3      50,13,32      73,43,72   0
0.5      53,13,24      77,43,64   2
0.5      56,13,26      78,46,66   0
0.127    46,13,21      70,41,61   3
0.127    55,13,35      79,26,75   0
#end{set}
#begin{set}
    ・
    ・
#end{set}
#end{clone}
```

クローンセットの情報
クローンセット
ID 0.3のファイル 42行目から67行目
ID 0.3のファイル 50行目から73行目
ID 0.5のファイル 53行目から77行目
ID 0.5のファイル 56行目から78行目
ID 0.127のファイル 46行目から70行目
ID 0.127のファイル 55行目から79行目

図 3　CCFinder における出力結果 2

4.3　表示例

コードクローン関係の有無に関わらず，バージョン間でパッケージ，ファイルが新しくできたか消えたかの変化を表・グラフで提示 (図 6〜図 9) しつつ，1 つのシステムにおけるバージョン間のコードクローン関係の変化を次の観点から表示する．いずれの場合も，分析結果はコードクローン関係を持つクラスの対 (クローンクラス対) の数を示している．

- バージョン間のクローンクラス対の数の変化
  それぞれのバージョンのソフトウェアに存在するクローンクラス対の総数の変化を表・グラフで提示 (図 4，図 5)．
- パッケージ内部のクローンクラス対の数の変化

それぞれのパッケージ内にその両端が存在するクローンクラス対の総数の変化を表・グラフで提示 (図 6，図 7)．

- ファイルのクローン関係の変化
  それぞれのクラスにおいて，クローン関係にあるクラスの数の変化を表・グラフで提示 (図 8，図 9)．

5　コードクローン視覚化ツールの適用例と評価

実際に Java で書かれたソフトウェアのソースコードを分析し，ツールによって描かれた情報がどのように役に立つかを考察する．バージョン間，パッケージ内部間，ファイル間の 3 つの視点からコードクローン関係の分析結果を紹介する．今回，解析対象のプログラムにしたものは Hadoop という，大量のデータを手軽に複数のマシンに分散して処理できるオープンソースの分散処理プラットフォームである．Hadoop の branch-0.1 をバージョン 1，branch-0.2 をバージョン 2，branch-0.3 をバージョン 3，branch-0.4 をバージョン 4，branch-0.5 をバージョン 5 として適用を行った．

5.1　バージョン間のクローンクラス対の数の変化

version	クローン総数
version.1	30
version.2	85
version.3	105
version.4	126
version.5	169

図 4　バージョン間の表

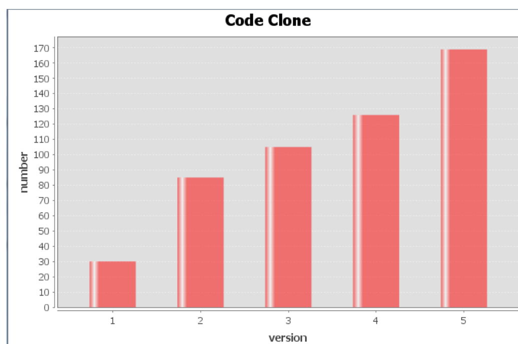

図 5　バージョン間のグラフ

この図 4 の表からは，バージョンが上がるごとにクローンが増加していることがわかる．この表を参照した図 5 のグラフでは，クローンクラス対の総数が大きく増えたバージョンとしてバージョン 1・2 の間，バージョン 4・5 の間が確認できる．また表からも，バージョン 1 からバージョン 2 は 55，バージョン 4 からバージョン 5 は 43 と，大きくクローンクラス対の総数が増えたことがわかり，この時の変更内容に対してより詳細な分析が必要なことに気づける．このように，バージョン間の視点の分析から始めることで，どのバージョンから分析を行った方が効率的かを速やかに判断することができる．

5.2　パッケージ内部のクローンクラス対の数の変化

Package Name ▲	ver.1	ver.2	ver.3	ver.4	ver.5
\java\org\apache\hadoop\tools	-1	-1	0	-1	-1
\java\org\apache\hadoop\util	0	0	0	0	0
\test	9	38	43	48	48
\test\org	9	38	43	48	48
\test\org\apache	9	38	43	48	48
\test\org\apache\hadoop	9	38	43	48	48
\test\org\apache\hadoop\dfs	0	1	1	1	1
\test\org\apache\hadoop\fs	0	3	6	6	6
\test\org\apache\hadoop\io	3	3	3	3	4
\test\org\apache\hadoop\ipc	1	1	0	0	0
\test\org\apache\hadoop\mapred	1	1	1	3	2
\test\org\apache\hadoop\record	-1	13	13	13	13
\test\org\apache\hadoop\record\test	-1	13	13	13	13

図 6　パッケージ内部間の表 (一部)

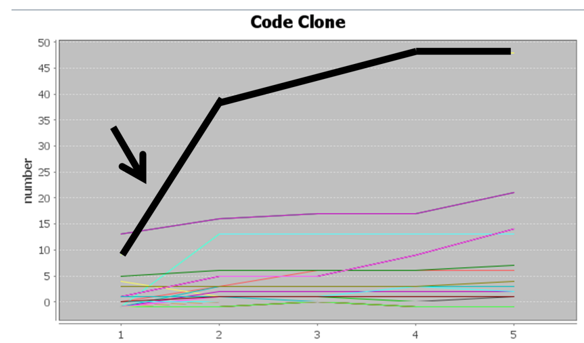

図 7　パッケージ内部間のグラフ

この図 6 の表では，パッケージ名でソートした状態の表の一部を示している．例えば，test というパッケージ内ではクローンクラス対の総数がバージョン 1 では 9，バージョン 2 は 38，バージョン 3 は 43，バージョン 4，バージョン 5 は共に 48 であることがわかる．また，0 という数値はそのパッケージ内にコードクローン関係がないことを指し，-1 という数値はそのパッケージ自体がそのバージョンには存在しないことを指す．この表を参照した図 7 のグラフでは，このソフトウェアはバージョンが上がると共に，ある 1 つのパッケージだけが極端に数値が大きくなっていることがわかる．表で確認すると，\test\org\apache\hadoop のパッケージ内で多くのクローンクラス対が急激に増えていることがわかる．このパッケージのより詳細な分析が必要なことがわかる．このように，パッケージ内部間の視点の分析を取り入れることで，具体的にどの部分でクローンクラス対が増えたかがわかる．どのパッケージから分析を行った方が効率的かを速やかに判断することができ，その後からのファイル間の分析などの作業効率化につながる．

5.3　ファイルのクローン関係の変化

具体的に\test\org\apache\hadoop 内を分析した．この図 8，図 9 の数値の 0 という数値はそのファイルにコードクローン関係がないことを指し，-1 という数値はそのファイル自体がそのバージョンには存在しないことを指す．図 8 の表を参照した図 9 のグラフでは，直観的に多くのファイルでコードクローン関係の数が増えていること

File Name	version.1	version.2	version.3	version.4	version.5
FSImage.java	-1	-1	0	-1	-1
UniqApp.java	-1	-1	0	-1	-1
FileContext.java	-1	1	1	1	1
TestFileSystem.java	6	11	13	15	18

図 8 ファイル間の表 (一部)

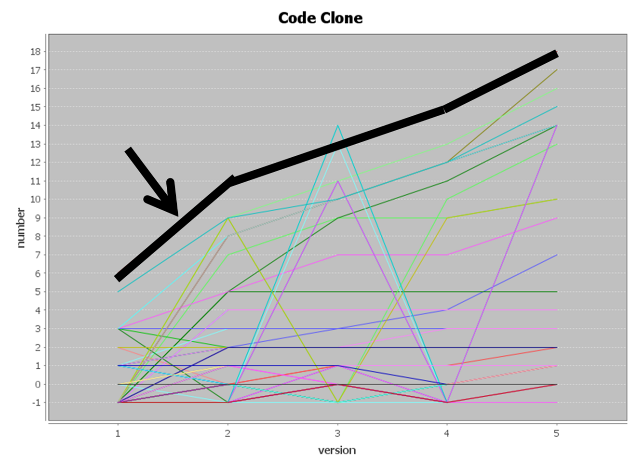

図 9 ファイル間のグラフ

がわかる．また，大半のバージョンで一番多くコードクローン関係を持つファイルがあることがわかる．このファイルを表で調べると図 8 にもある TestFileSystem.java というファイルであることがわかり，TestFileSystem.java をより詳細に調査する必要があることに気づくことができる．このように，ファイル間のコードクローン関係に対し，既存のツールにはなかったバージョン間での変化を提示することで，具体的にどのクラスのどのバージョンの変更でクローンクラス対が増えたかが直感的にわかり，より効率的に問題点を見つける”きっかけ”となる．

6 今後の課題

6.1 フィルタリング機能の実現

コードクローン情報の中に，調査の必要があるコードクローンとそうでないものが混在している．例えば，プログラミング言語に依存しているコードクローンは，コードクローンとして分析する価値は低い．言語依存のコードクローンの例としては，連続した変数宣言やメソッド呼出，switch 文の連続した case エントリ，典型的な for 文の処理ルーチンなどが挙げられる．フィルタリングの機能を実現して，これらのコードクローンを解析結果から除外することで精度の高い分析結果が得られると考える．今後フィルタリングの機能の第一歩として，ユーザが手動で指定したクローンクラス対を表やグラフから削除する機能をつける予定である．

6.2 同一部品の判定機能の実現

現在は，ファイルパスからソフトウェア部品名を推定し，複数バージョン間の同一の部品の判定を行っている．バージョンの前後でソフトウェア部品の配置場所の変更などがあった場合でも，中身がほぼ一致している部品を同一の部品とみなし，追いかけられるようにする機能を実現することで，途中の開発方針の変更によってクラス配置の方針変更があった場合でもそれらを柔軟に取り入れることができると考えられる．ただし，クラス配置の方針変更により，クラスの役割自体も変わる場合も考えられ，機能自体が必要かどうかを検討する必要がある．今後，多くの事例に実際に適用し，クローン関係が追跡できなくなった部品を解析し，それらに後継とすべき部品があるかどうかを調査し，どのような条件で後継部品を決定すべきかを調査する必要があると考えられる．

7 まとめ

本研究では，ソフトウェアの保守活動において，コードクローン関係が開発を通じてどのように変化したかについて全体的な情報を提示するツールを試作した．既存のツールにはない，バージョン間でクラスやパッケージ単位でのクローン関係の変化を表やグラフで表示する機能を実現した．そして，この提案したツールで実際の適用例を元に考察を行った．本研究の成果として，この提案したツールが，コードクローン関係の分析において，コードクローンの分布の変化を直観的に把握することを可能にし，直接の開発者だけでなくとも，開発に参加する人全体で広く情報を共有するのに役立つと考えられる．今後の課題として，フィルタリング機能や同一部品の判定方法の実現の検討が挙げられる．

参考文献

[1] T. Kamiya, S. Kusumoto, K. Inoue: ”CCFinder: Amultilinguistictoken-based code clone detection system for large scale source code,” *IEEE Transactions on Software Engineering*, vol.28, no.7, pp.654-670, 2002.

[2] Y. Ueda, Y. Higo, T. Kamiya, S. Kasumoto, and K. Inoue: ”Gemini:Code Clone Analysis Tool,” *Proceedings of 2002 International Symposium on Empirical Software Engine*, vol.2, no. 386, pp.31-32, 2002.

[3] 大平雅雄, 横森励士, 阪井誠, 岩村聡, 小野英治, 新海平, 横川智教: ”ソフトウェア開発プロジェクトのリアルタイム管理を目的とした支援システム,” 電子情報通信学会論文誌 D-I, vol.J88-D-I, no.2, pp.228-239, 2005.