部品間の関係変化を分析するツールにおける後継部品の追跡に関する考察

2010SE112 草野福美　2010SE191 澤本美澄　2010SE239 田西恭平
指導教員：横森励士

1　はじめに

ソフトウェア開発では，様々な変化への対応や，品質の向上，機能の向上を図るために長期にわたって保守活動が行われているものが多数存在し，内部の構造はしだいに複雑化し把握しにくくなる．ソフトウェアを構成する部品間の関係がどのように複雑化するのかを視覚化するツールが提案されているが，単純にファイル名とパッケージ名が一致している部品間の前後のバージョンで関係を追いかけるだけでは，途切れる部品が多く，変化を追跡するのに不十分であった．

本研究では，ファイル名やパッケージ名が更新前後で変化した部品に着目し，その部品の名前がどう変化したか，それらが後のバージョンの部品と対応付け可能かを実際のオープンソースソフトウェアを対象に調査する．内容がどう変わったかを調査し，どのような方法を採用すれば視覚化ツールにおいて十分な効果を発揮できるか考察する．これらの分析から，後継となる部品とみなす条件を設定し，それらの基準にもとづいてツール上で連続した部品として表示させる．これらの改良が視覚化ツールには必要不可欠であると考えられ，このような改良を行うことで，ツールを通じて開発者がソフトウェア開発における進捗情報をより的確に共有することができると考えられ保守活動における作業の支援につながると考えられる．

2　背景技術

2.1　ソフトウェアの成長と視覚化ツール

長期間にわたって開発されたアプリケーションは，不具合の対応，機能の追加などを通じて多くの修正が行われている．それらの修正によって，時間と共にソフトウェアの構成はより複雑化していく．私たちの研究室では，開発を通じてソフトウェアの部品間の関係がどのように変化していくかをグラフとして視覚化するツールを，Clone Tool[3] および Use Relation Viewer（URV）[4] として提案した．Clone Tool[3] は，ソフトウェアの複数のバージョン間のファイルを分析して，コードクローンの変化を視覚的に表示するツールである．CCFinder の分析結果として生成されるテキストファイルを読み込み，コードクローンの変化を表やグラフとして出力する．Use Relation Viewer[4] は，Java ソフトウェアのクラス間の利用関係を分析するツールである．Classycle の分析結果として生成される xml ファイルを読み込み，表やグラフとしてファイル間の利用関係を，図 1 のように視覚化する．ソフトウェアの複数のバージョン間のファイルを分析して，利用関係の差分の変化を示すこともできる．これらのツールでは，ソースコードの入手が可能な一方でリポジトリの情報が何らかの理由で入手できない場面でも，分析可能とすることを想定し，入力を各バージョンのソースコードとしている．

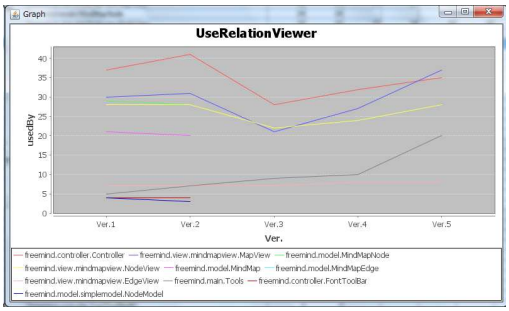

図 1　Use Relation Viewer の画面

3　研究について

3.1　既存ツールの問題点

現状の部品間の関係を視覚化するツールでは，ファイル名およびパッケージ名が完全一致した場合のみ，履歴中で同一の部品とみなしている．同一の機能を保有しているクラス同士であっても，ファイル名や存在するパッケージが一致しなければ別のクラスとみなされ，関係の変化を追跡することができない．グラフとして表示した場合，例えばパッケージを移動した時点で追跡できなくなるので，図 1 のようにグラフの線が途切れて表示されてしまう．

3.2　研究の目的

本研究では，実際のオープンソースソフトウェアの開発履歴を対象に，前のバージョンのクラス群と一致しなかったクラス群の分析を行う．分析では，その直後のバージョンにおいてクラスで記述されていた内容がどのクラスに存在するかを調査し，それぞれのクラスの後継の部品となるべきクラスを推定する．分類ごとに内容の一致率を調査し，どのような形で後継となる部品を推定すれば十分な効果を発することができるかを考察する．

4　評価実験

以下の項目を分析することを目的として，本研究の評価実験で調査する．さらに，それぞれの項目について詳細な分析を行うために，それぞれに小項目を追加する．

1. パッケージ名の変更されたクラスはどの程度存在するか．
   (a) パッケージ名の変更はどのように行われている

か.

(b) パッケージ名の変更の場合，どれだけ更新前後で中身が一致しているのか.

2. クラス名の変更はどの程度存在するか.
   (a) クラス名の変更はどのように行われているか.
   (b) クラス名の変更の場合，どれだけ更新前後で中身が一致しているのか.
3. ファイル名の変更以外の理由により追跡ができなくなっているクラスはどの程度存在するか.
   (a) ファイル名の変更以外の理由により追跡ができなくなっているクラスは，どのように変更されているか.

4.1 評価実験の方法

あるソフトウェアの開発履歴を対象として，以下の作業を行う．これにより，連続したバージョン間で追跡できなかったファイルを調査する．

1. いくつかの連続したバージョン群を選び，ソースコードを取得する.
2. 連続する２つのバージョンを抽出し，ファイル名，パッケージ名が次のバージョンに存在しなかったものを抽出する.
3. 抽出したファイルから，クラス名が一致するファイルが次のバージョンに存在するかを調査する.
4. クラス名が一致したファイルが存在しなかった場合，同一のパッケージ内から，類似したクラス名のものが存在するかどうかを調査する.
5. 類似したクラス名のものが存在しなかった場合，パッケージ名，クラス名の両方に着目し，名前が類似したファイルが存在するかどうかを調査する.
6. 上記３つの手順について，後継となる部品である可能性の高いファイルが発見できた場合，ソースコードを比較し，評価を行う.
7. 以上の手順を繰り返し，最終的に後継となる部品とみなすことのできるファイルが存在しなかった場合，後継となる部品はなしと判定する.

このように分類したクラス群について，後継となるクラスが存在するクラスについてそれぞれに diff を用いて類似度を計算する．なお，類似度とは，ソースコード P と P の後継の部品と想定されるソースコード Q について，(Q に含まれる P の行数 +P に含まれる Q の行数)/（P の行数＋ Q の行数）で定義したものである．内容がコピーされた部品ほど，高い値を示すと考えられる.

4.2 評価実験における結果の分析方法

後続のバージョンでパッケージ名，クラス名が一致する部品が確認できなかった部品について，表 1 で分類した．パッケージ名の変更にあたる項目 A，B については，移動した理由についてもそれぞれ調査を行い，次のように調査した．それぞれの項目について，例も併せて示す.

表 1 データの分類項目

記号	項目
A	パッケージ・クラス名が両方変更
B	パッケージ名のみ変更
C	クラス名のみ変更
D	後継の部品なし

- 所属パッケージが親になった
  例：test.example.cname → test.cname
- 所属パッケージが子になった
  例：example.cname → test.example.cname
- パッケージ名が途中で変わった
  例：test.example.cname → package.example.cname
- パッケージ名の最後の部分が変わった
  例：test.example.cname → test.example.cnames
- パッケージ名の途中と最後の部分の両方が変わった
  例：test.example.cname → package.example.cnames

クラス名の変更にあたる項目 A，C については，変更内容について調査を行い，次のように分類した．こちらもそれぞれの項目について，例も併せて示す.

- クラス名の単語が一部違うものに変化した
  例：　TestExample → PackageExample
- クラス名に新しい単語が付け足された
  例：　Example → TestExample
- クラス名が単純に変更された
  例：　Example → Examples

後継となる部品がない場合は D として分類した上で，類似した機能を持つクラスがないか調査を行い，さらに以下のように分類した.

- 別のクラスに統合された
- 完全に消滅した

4.3 評価実験の対象

SourceForge で公開されている開発プロジェクトを対象として，評価実験の対象プロジェクトを選出した．Java を開発言語として，5 バージョン以上のソースコードが入手できるという条件のもとで，次の 6 プロジェクトを分析対象とした.

- Freemind[*1]
  マインドマップ作成ソフトウェア.
- Hodoku[*2]
  数独（ナンバープレイス）のためのソフトウェア.

[*1] http://sourceforge.net/projects/freemind/
[*2] http://sourceforge.net/projects/hodoku/

- JgraphT[*3]
  Java におけるグラフを構築するためのライブラリ.
- JavaGeom[*4]
  ジオメトリアプリケーション用の Java ライブラリ.
- JXPlorer[*5]
  JavaLDAP クライアント.
- JIPT[*6]
  Java の画像処理のツールキット.

表 2 追跡できなかったクラスの個数

プロジェクト名	次のバージョンで追跡できなかったクラス数	A	B	C	D
Freemind	61	0	31	6	24
Hodoku	57	0	43	0	14
JgraphT	155	6	120	4	25
JavaGeom	113	0	78	0	35
JXPlorer	22	0	0	0	22
JIPT	16	0	0	0	16

4.4 評価実験の結果

前節で選んだ 6 つのプロジェクトに対して，後継となる部品の推定を行った．結果を表 2 に示す．パッケージ名のみ変わるケースが特に多く見られた一方で，クラス名の変更は多くないとわかった．また，どのプロジェクトでもある一定の部品は後継となる部品を確認できなかった．

次に，さらに細かく分類した結果と，追跡可能部分について分類ごとに平均類似度を測定した結果を表 3 に示す．プロジェクトごとに平均類似度が大きく異なることが見受けられる．Freemind では，パッケージ名の変更によるものが高い類似度を示している．しかし，Hodoku, JavaGeom では類似度は極端に低い数値を示している．対象のプロジェクトを通じて，パッケージの最後の部分が変更されたケースは高い類似度を示している．クラス名の変更によるものでは，各プロジェクトそれぞれ平均類似度がパッケージ名の変更によるものと比べて低い数値を示している．クラス名の変更によるものの中では，名前の単語が変化したものが最も高い数値を示した．また，後継となる部品が存在しなかったものもいくつか存在が確認されている．

5 考察

評価実験の結果から，以下の結果を得ることができた．

1. パッケージ名の変更されたクラスはどの程度存在するか.
   →対象のプロジェクトでの追跡できなかったクラスは，ほとんどがパッケージ名の変更が原因であった.
   (a) パッケージ名の変更はどのように行われているか.
   →プロジェクトごとにばらつきはあるが，パッケージの途中部分が変更したものが最も多く確認された．Freemind では，階層の変化を伴ったパッケージ名の変更はなかったが，そのほかのプロジェクトではパッケージの階層が変化していた．特に，JgraphT と JavaGeom では，パッケージが親や子になる変更の方が多くなっていた.
   (b) パッケージ名の変更の場合，どれだけ更新前後で中身が一致しているのか.
   → Freemind，JgraphT では平均でも半分以上一致する場合が多く，クラスによっては完全に一致したものも存在した．その一方で，Hodoku と JavaGeom では一致率が低く，平均で 1 %未満となり，プロジェクトによって差が大きく開いた．どのプロジェクトでも類似度が最も高くなるのはパッケージ名の変更の場合であった.
2. クラス名の変更はどの程度存在するか.
   →クラス名の変更はパッケージの移動より少ない頻度で見られた.
   (a) クラス名の変更はどのように行われているか.
   →プロジェクトごとにばらつきがあること，色々なパターンが混在していることがあった．Freemind では 3 つのパターンがそれぞれ同じ頻度で見られ，JgraphT では，名前の単語の付け足しが最も多く，名前の単語の一部の変化が最も少ないというばらつきが見られた.
   (b) クラス名の変更の場合，どれだけ更新前後で中身が一致しているのか.
   →名前の一部の単語が変化した場合が最も高い類似度を示した．全体的にパッケージ名の変化の場合と比較して一致度は低い．JgraphT では名前の変化で一致度が 0 という極端な結果が見られた.
3. ファイル名の変更以外の理由により追跡ができなくなっているクラスはどの程度存在するか.
   →どのプロジェクトでも存在が確認された．後継部品が存在しないもののみのプロジェクトもあった.
   (a) ファイル名の変更以外の理由により追跡ができなくなっているクラスは，どのように変更されているか.
   →ほとんどが完全に消滅していたが，ほかのクラスに統合されたクラスの存在も確認された.

評価実験を行った結果，追跡できなかったクラスはパッケージ名の変更が原因となっているものが大多数であった．実際に追跡できなかったクラスを次のバージョンのクラスと比較したところ，関連したファイルがまとめて違うパッケージに移動されたことで大量のクラスが追跡でき

[*3] http://jgrapht.org/
[*4] http://geom-java.sourceforge.net/
[*5] http://jxplorer.org/
[*6] http://sourceforge.net/projects/jipt/

表 3 それぞれのクラスの分類結果

	Freemind		Hodoku		JgraphT		JavaGeom		JXPlorer		JIPT	
項目	数	平均類似度	数	平均類似度	数	平均類似度	数	平均類似度	数	平均類似度	数	平均類似度
パッケージ名の変更によるもの (A，B)	31	0.592	43	0.001	130	0.476	78	0.093	0	-	0	-
パッケージが親になった	0	-	5	0	12	0.205	0	-	0	-	0	-
パッケージが子になった	0	-	0	-	38	0.290	69	0.086	0	-	0	-
パッケージの途中部分が変更された	3	0.466	38	0.001	71	0.591	8	0.047	0	-	0	-
パッケージの最後部分が変更された	12	0.276	0	-	7	0.869	1	0.989	0	-	0	-
パッケージの途中部分，最後部分が両方変更された	16	0.674	0	-	3	0.333	0	-	0	-	0	-
クラス名の変更によるもの (A，C)	6	0.268	0	-	12	0.130	0	-	0	-	0	-
名前の単語の一部が変化した	2	0.491	0	-	2	0.471	0	-	0	-	0	-
名前に単語が付け足された	2	0	0	-	6	0.103	0	-	0	-	0	-
名前が変化した	2	0.313	0	-	4	0	0	-	0	-	0	-
後継となる部品が存在しなかったもの (D)	24		14		4		35		22		16	
他のクラスに統合された	6		0		7		0		0		0	
完全に消滅していた	18		14		16		35		22		16	

なくなっている例が多く存在した．新しい所属パッケージが親パッケージになったものや，子パッケージになった例もいくつか存在した．クラス名については，1 つの変更の影響は，名前が変更されたファイル自身にのみ起こるが，パッケージ名の変更は，1 つの変更が起こると，そのパッケージに属する全てのファイルに影響を起こしてしまうので，クラス名よりも，パッケージ名の変更により追跡できなくなったクラスが多く発生したと考えられる．

類似度を調査するとプロジェクトごとに大きく異なり，全体的に 4 割以上の比較的高い数値を割り出すプロジェクトもあれば，1 割以下の低い数値がほとんどを占めるプロジェクトも存在した．しかし，同じプロジェクト内では，類似度が全体的に高いか，全体的に低いかのどちらかになる傾向があったので，名前による追跡が有効なプロジェクトとそうでないプロジェクトで追跡の有無を選択することができれば有効であると考えられる．

5.1 関連研究

ソフトウェアの更新前後において，更新前，更新後の部品群（クラス群）がどのように対応をしているかを分析するための研究は数多く存在している．文献 [1] によると，それらのマッチングに関する研究のサーベイを行っており，手法を大きく分けて 7 つに分類している．文献 [1] によると，一番単純な，ソフトウェア部品の中に存在する固有名詞，動詞などの名前を用いてマッチングを行う方法である Entity NameMatching で，多くのバージョン間でのマッチングに対して，一番基本的であるが十分な性能を示していると述べている．本研究の分析結果でも，後継となる部品の存在したクラスのうち，パッケージ名のみの変化が全体の多くを占めており，部品間の関係変化を追跡するツールにおいても，パッケージ名を吸収することが後継となる部品の追跡に対して有効であると考えられる．

Danny らの研究 [2] では，リファクタリングを通じてソフトウェア内部構造が変化することに対応することも目的として，それぞれの構成要素やリファクタリング操作を考慮して，更新のヒストリーが保たれるような SCM(Software Configuration Management) システムを提案した．本研究では，各バージョンのソースコードのみが得られることも分析の前提としているが，リポジトリの情報も入手可能な場合，これらの手法をもとに，より正確な追跡が行えることが予想できる．

6 まとめ

本研究では，ファイル名かパッケージ名が更新前後で変化した部品について，それらの部品がどう変化したかを調査した．結果として，パッケージ名の変更の場合は，更新前後で中身が比較的変化しないこと，プロジェクトごとに傾向が異なることなどを確認した．今後，後継となる部品とみなす条件を設定し，ツールに反映させることで，ツールが示す情報がより有益になり，情報の共有が効果的に行えるようになることを通じて保守作業を支援することができることと考えられる．

参考文献

[1] M.Kim,D.Notkin: ”Program Element Matching for Multi Version Program Analyses,” in processing of the 2006 international workshop on mining software repositories, P.58-64,2006.

[2] D.Dig,K.Manzoor,T.Ngayen,R.Johnson: ”Refactoring-aware Configuration Management for Object-Oriented Programs,” Proceeding of the 29th International in Conference on Software Engineering, P.427-436,2007.

[3] 松坂太智，岡田直希，坂下智紀 : ”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，”南山大学情報理工学部 2012 年度卒業論文要旨集，2012.

[4] 亀井雄佑，木下裕太郎，前原一機 : ”バージョン間の利用関係の変化を提示するシステムの試作，”南山大学情報理工学部 2012 年度卒業論文要旨集，2012.