履歴情報を用いた同一ユースケース実装クラス群抽出法の提案

Extracting implementation Classes of Use Case by Using Development Histories

楠田 泰三* 川口 真司† 松下 誠‡ 井上 克郎§

Summary. Understanding software that is about to maintain is a first step of software maintenance, but it takes lots of costs to achieve. Software visualization is one of the key technologies to help developers understanding software. However, visualizing huge software systems makes huge maps, and it is hard to grasp. In this paper, we propose a method to find out a set of classes that may compose a use-case of target software. With this method, developers could be focus a small portion of software that may help to reduce costs of software understandings. We also show a example to validate our method.

1 導入

ソフトウェアの保守作業はソフトウェア全体のコストの大部分を占めるようになってきている [3]．また，ソフトウェアの保守作業の多くは保守対象となるソフトウェア理解に費やされている [7]．そこで，保守対象となるソフトウェアの理解を支援するための手法がこれまで研究されてきた．

ソフトウェアの理解支援の一つとして，可視化に関する研究が行われている [6]．ソフトウェアの可視化とは，ソフトウェアの構造やふるまいを図やグラフを用いて表現を行う技術である．ソースコードよりも抽象度の高い図やグラフを用いてソフトウェアを表現する事で，ソフトウェアの複数のクラス，関数といったエンティティを同時に理解したり，それらエンティティ間の関係を理解したりすることを支援することを目的としている．

しかし，オブジェクト指向開発では開発が進むにつれてクラス数が増加していくため，大規模なオブジェクト指向ソフトウェア全体を可視化できたとしても，出力された図も大規模なものになるため，理解が容易とはならないという問題がある．一方，開発者にとってそのソフトウェアの全てを同時に理解する場合は多くない．例えば，ソフトウェア中のある不具合を修正する場合，その不具合が発生するユースケースを実装している部分を理解すればよい．すなわち，ある時点で着目した部分のみが詳細にわかれば良いことが多い．

そこで本研究では，大規模なソフトウェアから同一のユースケースを実装しているクラス群を抽出する手法の提案を行う．具体的には，ソフトウェア中で頻繁に同時に更新されるクラス群を，同一ユースケースを実装しているクラス群として抽出する．これは，同一ユースケースを実装している箇所は同時に更新されることが多いと考えられるためである．また，本手法を用い，抽出したクラス群を可視化することで，要素数の少ない図を出力することができソフトウェア理解を支援できる．

以降 2 節では本研究の関連研究の紹介を行う．3 節では手法について述べ，4 節では検証実験について述べる．5 節では本研究に関する考察を述べ，6 節で結論を述

*Taizo Kusuda 大阪大学大学院情報科学研究科

†Shinji Kawaguchi 大阪大学大学院情報科学研究科

‡Makoto Matsushita 大阪大学大学院情報科学研究科

§Katsuro Inoue 大阪大学大学院情報科学研究科

べる．

2 関連研究

ソフトウェアからユースケース実装箇所を抽出する研究として，Lucca らの研究がある [4]．この研究では，メソッドのコールグラフを作成し，グラフ上でユーザからの入力を持つメソッドからユーザへの出力を持つメソッドへの経路を見つけ，その経路上のメソッドを持つクラスを一つのユースケース実装クラス群と考えている．しかし，ソフトウェアが大きくなると入力を持つメソッドから出力を持つメソッドまでの経路が非常に多くなり，多数のユースケースが抽出されるという問題がある．

ソフトウェアの同時更新の情報をソフトウェア開発に役立てる研究としては，Zimmermann らが行っている研究がある [8]．この研究では，モジュールやディレクトリをまたいで頻繁に同時に更新が行われた場合，カプセル化が適切に行われていないと判断し，そのような同時更新が起きている箇所をソフトウェアの再構成箇所として提案を行っている．また，この研究では再構成箇所の提案を行うために，同時更新傾向の強い 2 つのファイル，関数などのソフトウェアエンティティを求めているが，本研究では 2 つ以上エンティティを同時に扱い，同時更新傾向が強いクラス群を求めている．

3 提案手法

本節では，同一ユースケースを実装している箇所を可視化することを目標として大規模なオブジェクト指向ソフトウェアから同一ユースケース実装クラス群を抽出する手法について述べる．本手法では，版管理システムによって保持されている履歴を用いて，同時に更新される傾向が強いクラス群を同一のユースケースを実装しているクラス群として抽出する．本手法は 1) 同時更新情報の抽出，2) 同時更新の強さを表すメトリクスの計算，3) 更新傾向グラフの作成，4) クラスタ化更新傾向グラフの作成，5) 同時更新傾向の強いクラス群の抽出という 5 つの作業から構成される．以下，この 5 つの作業について詳しく説明をする．

3.1 同時更新情報の抽出

本研究では，同じコミット作業により更新されたクラスを同時に更新されたクラスと考える．同じコミット作業で更新されたクラスを求めるには，版管理システムのリポジトリに蓄積されている更新者，更新日時，更新時のコメントなどの情報を用いる．

3.2 同時更新の強さを表すメトリクスの計算

次に，求めた同時更新情報を用いて，同時更新の強さを求める．同時更新の強さとは，任意の 2 つのクラスについて，その 2 つのクラスがどの程度同時に更新される傾向にあるかを数値で表現したものである．本研究では，そのような同時更新の強さを表現するメトリクスとして，support，confidence という二つのメトリクス [8] を用いる．以下では，この二つのメトリクスについて述べる．

3.2.1 support メトリクス

support メトリクスは，ある 2 つのクラス対 e_1, e_2 について，その 2 クラスが同時に更新された回数を表現する．ソフトウェア中の全クラスの集合を E とし，ソフトウェアの i 番目の更新により更新されたクラスの集合を $G_i \subseteq E$ とする．このとき任意のクラス $e_1, e_2 \in E$ について，

$$support_{e_1,e_2} = |\{G_i | e_1 \in G_i \wedge e_2 \in G_i\}|$$

と定義する．定義より，$support_{e_1,e_1}$ はクラス e_1 がこれまでの開発において更新された回数を表現する．

3.2.2 confidence メトリクス

confidence メトリクスは，ある 2 つのクラス順序対 e_1, e_2 について，あるクラスが更新されたとき，別のあるクラスがどの程度の割合で更新されるか，ということを表現する．$confidence$ メトリクスを以下の式で定義する．任意のクラス $e_1, e_2 \in E$ について，

$$confidence_{e_1,e_2} = \frac{support_{e_1,e_2}}{support_{e_1,e_1}}$$

3.3 更新傾向グラフの作成

次に，support，confidence メトリクスを用いて，同時更新傾向を表現するグラフである更新傾向グラフを作成する．グラフの頂点はクラスを表現する．頂点間の辺は向きを持っており，辺の始点が頂点 e_1，終点が頂点 e_2 であるとき，($support_{e_1,e_2}$, $confidence_{e_1,e_2}$) という値の組を持っている．

3.4 クラスタ化更新傾向グラフの作成

前節で作成した更新傾向グラフのクラスタリングを行い，クラスタ化更新傾向グラフを作成する．クラスタ化更新傾向グラフの各ノード，エッジは更新傾向グラフとほぼ同じものであるが，加えて各ノードはどのクラスタに含まれているかという情報を保持している．

クラスタリングは，クラスタ内に含まれるクラス間の support，confidence の値ができるだけ高くなるようにする．

更新傾向グラフをクラスタリングするにあたり，まず support の値により更新傾向グラフの辺にフィルタリングを行う．つまり，support 値として閾値 S_{th} 以下の値を持っている辺を，更新傾向グラフから除去する．

次に，support 値を用いてフィルタリングされた更新傾向グラフのクラスタリングを行う．各クラスタはクラスを表現する頂点の集合となっている．クラスタリングは，全てのクラスタ間の近さが閾値 W_{th} より小さくなるまでクラスタ間の近さが近いものから順にクラスタを結合していく，という方法で行う．クラスタリングの手順は以下の通りである．

1. グラフ中の一つの頂点のみを含むクラスタを各頂点についてそれぞれ一つずつ作成する．それらクラスタをクラスタ集合 L に加える．
2. $C_i \in L, C_j \in L$ なる C_i, C_jについて $W_{C_i,\,C_j}$ が最大となる C_i, Cj を求める．
3. $W_{C_i,\,C_j} \geq W_{th}$ であれば
   3.1 $L \leftarrow L - C_i - C_j$
   3.2 $L \leftarrow L \cup \{C_i \cup C_j\}$
   3.3 2 に戻る
4. $W_{C_i,\,C_j} < W_{th}$ であれば
   4.1 L の各要素を一つのクラスタとして出力

ここで，C_iを i 番目のクラスタ，クラスタ集合 $L = C_1, C_2, ..., C_n$ とする．また，2 つのクラスタ C_1, C_2 の近さ $W_{C_1,\,C_2}$ は以下の式で計算される．

$$W_{C_1,\,C_2} = \frac{1}{|C_1| \times |C_2| \times 2} \sum_{e_1 \in C_1} \sum_{e_2 \in C_2} (confidence_{e_1,e_2} + confidence_{e_2,e_1})$$

すなわち，クラスタ間の近さはクラスタをまたぐクラス間の $confidence$ 値の平均と定義する．

3.5 同時更新傾向の強いクラス群の抽出

前節で作成したクラスタ化更新傾向グラフを描画し，グラフ中の各クラスタが，適切な大きさになっているかを調べる．一つのクラスタに含まれるクラスの数が多過ぎたり，多くのクラスタに含まれるクラスの数が少な過ぎたりした場合は，閾値

を設定しなおして再びクラスタ化更新傾向グラフを作成する．グラフ中の各クラスタが適切な大きさになっていると判断した場合は，グラフ中の各クラスタに含まれるクラス群を同時更新傾向の強いクラス群として抽出する．

4　検証実験

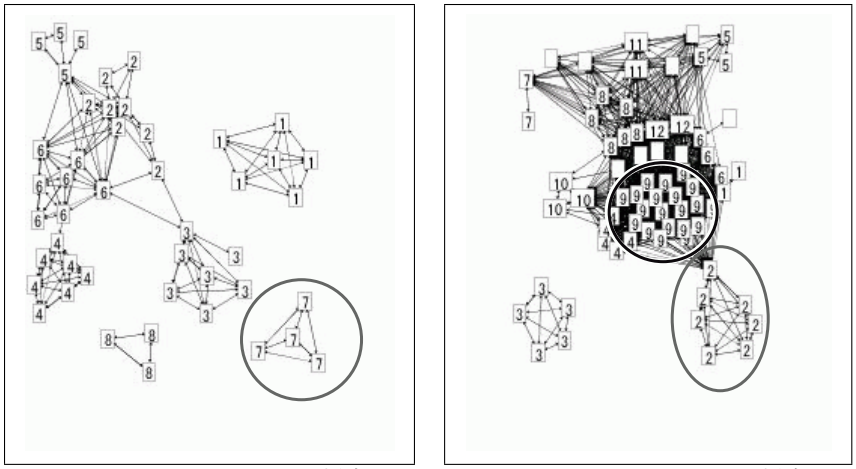
本手法の妥当性を確かめるために，検証実験を行った．本実験では，対象とするソフトウェアに対し本手法を適用し，抽出されるクラス群が同一ユースケース実装クラス群となっているかどうか検証を行う．

なお，本実験では，クラスタリングを行った時，各クラスタに対してクラスタ番号を適当に割り当てる．クラスタ化更新傾向グラフを描画する際はクラスがどのクラスタに属しているかをクラスタ番号を用いて表現する．また，グラフには辺が全く接続されていないノードの描画は行わない．

実験対象としては我々の研究室で開発された CVS リポジトリ閲覧，検索システム CREBASS(Cvs REpository Browse And Search System) [5] を用いた．

CREBASS は，CVS のリポジトリを対象として，Web ブラウザを通しての内容の閲覧，および開発者が必要とする情報の取得のための開発支援環境の構築を目指している．CREBASS の規模は 70 クラス，7513 行，これまでの総更新回数は 140 回である．このソフトウェアが実装している主なユースケースとして,「リビジョン情報の検索」「プロジェクトデータの変遷の折れ線グラフによる表示」「プロジェクト内のファイルのログの表示」などがある．

このソフトウェアに対して，本手法を適用した．support の閾値 $S_{th} = 2$ とし，クラスタリングの際に用いる閾値 W_{th} を変化させながらクラスタ化更新傾向グラフを描画した結果，W_{th} を 0.1 とするのが妥当であると判断した．$W_{th} = 0.1$ の時のクラスタ化更新傾向グラフを図 1(a) で示す．



(a) $S_{th} = 2, W_{th} = 0.1$ の場合　(b) $S_{th} = 0, W_{th} = 0.5$ の場合

図 1　CREBASS の更新傾向グラフ

ここで，図 1(a) 中のクラスタ番号 7 に含まれる 4 つのクラスに着目する．この 4 つのクラスの役割を開発者に聞いたところ，このクラス群は，前述した CREBASS のユースケースの 1 つである，ユースケース「プロジェクトデータの変遷の折れ線グラフによる表示」を実装しているクラス群であるということがわかった．このユースケースは，プロジェクト内にあるファイルの行数，編集量，コミットを行った開

発者の累計をそれぞれ折れ線グラフにして表示するというもので，抽出した 4 個のクラスは表示する折れ線グラフのデータごとに対応した 3 個のクラスと，それらの共通の処理を行う 1 個のクラスとなっていた．

本手法により，ユースケース「プロジェクトデータの変遷の折れ線グラフによる表示」を実装しているクラス群を抽出する事ができたが，他のユースケース，例えば「リビジョン情報の検索」を実装しているクラス群を抽出することはできなかった．そこで，このユースケースを実装しているクラス群を開発者に列挙してもらい，それらのクラスの開発履歴を調査した．その結果それらのクラスそのものの更新回数が 2 回しかないことがわかった．そのため，これらのクラスは support 値によるフィルタリングを行った結果，更新傾向グラフ中で孤立ノードとなってしまい，抽出できなかったものと考えられる．

そこで，このクラス群を抽出するために，$S_{th} = 0$，つまり更新傾向グラフの support 値のフィルタリングを行わずに，クラスタ化更新傾向グラフの作成を行い，同時更新傾向の強いクラス群を抽出することを試みた．クラスタリングの際に用いる閾値 W_{th} は 0.5 が妥当であると判断した．$W_{th} = 0.5$ の時のクラスタ化更新傾向グラフを図 1(b) に示す．実験の結果，図 1(b) 中のクラスタ番号 2 のクラス群が，ユースケース「リビジョン情報の検索」を実装しているクラス群と合致していることがわかった．

また，図 1(b) 中のクラスタ番号 9 のクラス群に着目すると，このクラス群はユースケース「プロジェクトデータの変遷の折れ線グラフによる表示」を実装しているクラス群を含むクラス群となっていたが，このクラス群には当該に含まれないクラス群も含まれていることがわかった．この結果から，support 値を用いた更新傾向グラフのフィルタリングは，クラスタリングにより同一ユースケース実装クラス群を抽出する際に有効であると考えられる．

CREBASS を用いた実験により，本手法を用いて同一ユースケース実装クラス群を抽出することは十分に可能である事がわかった．ただし，適切にクラス群の抽出を行うためには，着目するクラスの更新回数にあわせて，support 値によるフィルタリングを行うか否かを設定する必要があることもわかった．

5 考察

5.1 ユースケースとその実装クラス群の更新傾向

本手法では，同じユースケースを実装しているクラス群は同時に更新される傾向が強い，と仮定している．これは，ソフトウェアの開発を行う際には複数のユースケースの実装を並列的に行っているわけではなく，あるユースケースの実装が完了した後，別のユースケースの実装を行うというプロセスで開発を行っていくと考えるためである．実験対象として用いた CREBASS の開発者もユースケースを一つずつ実装していくというプロセスで開発を行っていた．よって，ユースケースを実装しているクラス群を抽出する際に同時更新の情報を用いることは有用であると考える．

しかし，クラスの更新回数が他のクラスに比べて非常に少ない場合もあり，履歴情報のみを用いてそのようなクラスを抽出するのは難しい．よって，同一ユースケース実装クラス群の全てを抽出するためには，履歴情報以外の情報の解析，例えばソースコードの静的解析の手法を組み合わせて用いる必要があると考える．

5.2 *support* 値によるフィルタリングの閾値 S_{th}

本手法では，更新傾向グラフからクラスタ化更新傾向グラフを作成する際，*support* 値によるフィルタリングを行っている．関係の無い作業を複数個行った後に一度にコミットを行った場合，それら関係の無い作業によって更新されたクラスも同時に更新が行われたものとする．そのような偶然同時に更新が行われたクラス対をクラ

スタリングの際に考慮しないようにするために，*support* 値によるフィルタリングを行う．このように複数の作業を行う事によって偶然同時に更新されたクラス間の *support* は，一つの作業内で更新されたクラス間の *support* より低く，1，2 程度である [8]．そこで，実験では *support* の閾値 S_{th} として 2 を採用することにした．しかし，更新回数が非常に少ないクラス群を抽出する際には，フィルタリングの閾値 S_{th} を 0 または 1 とする必要がある．

5.3 クラスタリングの閾値 W_{th}

クラスタ化更新傾向グラフを作成する際，クラスタ間の近さの閾値 W_{th} を変化させる事で出力されるクラスタの個数が変化する．仮に W_{th} が小さすぎる場合，一つのクラスタに含まれるクラスの数が多くなり別のユースケースを実装しているクラスが同じクラスタに含まれる確率が高くなる．逆に W_{th} が大きすぎる場合，一つのクラスタに含まれるクラスの数が少なくなり，同一のユースケースを実装しているクラス群が別々のクラスタに含まれることになる．そのため，W_{th} を適切に決める必要がある．本実験では，W_{th} を変化させながらその結果のクラスタ化更新傾向グラフを描画するということを繰り返して適切な W_{th} を求めている．しかし，適切な W_{th} はある程度ソフトウェアの更新の形態に依存する値であるため，ソフトウェア全体の更新回数や一度のコミットで更新されるクラスの数などを用いて W_{th} を計算する手法について今後考えていく必要がある．

6 結論

本研究では，ソフトウェアから同一ユースケースを実装しているクラス群を抽出し，そのクラス群のみを可視化することで要素数の少ない理解しやすい図を出力する手法を提案した．具体的には，同一ユースケース実装クラス群を抽出する方法として，ソフトウェアの開発履歴を用いて同時更新傾向の強いクラス群を同一ユースケース実装クラス群と考え，これを抽出する方法を考えた．また，本手法の妥当性を検証するため適用実験を行った．適用実験の結果から，本手法を用いる事で同一ユースケース実装クラス群を抽出することは可能であるが，そのためにはソフトウェアの更新の形態に合わせて適切に閾値を設定する必要があるということもわかった．

今後は，同一ユースケース実装クラス群を抽出する手法についてさらに研究を行っていくとともに，この手法を用いて抽出したクラス群を実際にどのように可視化するかについても考えたい．

参考文献

[1] Brian Berliner.: CVS II:Parallelizing Software Development.

[2] Ahmed E. Hassan and Richard C. Holt: Using Development History Sticky Notes to Understand Software Architecture. *Proceedings of the 12th IEEE International Workshop on Program Comprehension. pp.183-192,2004*

[3] R. L. Glass.: We Have Lost Our Way. *System and Software. pp.111-112,1992*

[4] Giuseppe Antonio Di Lucca, Annna Rita Fasolino and Ugo De Carlini.: Recovering Use Case models from Object-Oriented Code: a Thread-based Approach. *Proceedings of the Seventh Working Conference on Reverse Engineering. pp.108-117,2000*

[5] 中山崇, 松下誠, 井上克郎, “関数の変更履歴と呼び出し関係に基づいた開発履歴理解支援システム”, **電子情報通信学会技術研究報告** *pp.7-12,2004*

[6] Blaine A. Price, Ian S. Small, and Ronald M. Baecker.: A Taxonomy of Software Visualization. *Proceedings of the Hawaii International Conference on System Sciences. pp.597-606,1992*

[7] T. A. Standish.: An Essay on Software Reuse. *IEEE Transactions on Software Engeineering. pp.494-497,1984*

[8] T. Zimmermann, S. Diehl, and A. Zeller.: How history justifies system architecture (or not). *Proceedings of the International Workshop on Principles of Software Evolution. pp.95-105,2003*