社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

更新履歴情報と静的情報を用いて同一機能を実装しているクラス群を抽出する手法の提案

檜皮　祐希† 　松下　誠† 　井上　克郎†

† 大阪大学 大学院情報科学研究科　〒 560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{y-hikawa,matusita,inoue}@ist.osaka-u.ac.jp

あらまし　ソフトウェアの保守において，ソフトウェア理解や再利用のために特定の機能が実装されている箇所を特定することが必要になることがある．その際に用いられる手法として Feature Location という研究がある．Feature Location の既存手法は，静的手法，動的手法，更新履歴情報を用いた手法の 3 つに分けることができるが，各々の手法にはそれぞれ異なる問題点があり，ソフトウェアの機能によっては適切に抽出できない場合がある．本研究では，既存手法で用いられている情報のうち，更新履歴情報と静的情報を組み合わせることによりあるクラスと同一の機能を実装しているクラス群を抽出する手法を提案する．具体的には，まず，更新履歴情報を用いて同一機能を実装しているクラス群の抽出を行い，その抽出結果に対して，静的情報を用いて更新履歴情報から誤って抽出したクラス群の除去，及び，更新履歴情報で抽出できなかったクラス群の補完を行う．また，オープンソースソフトウェアに対して手法の適用実験と評価を行い，手法の有用性を示した．

キーワード　ソフトウェア理解，ソフトウェア再利用，同一機能実装クラス群の抽出，更新履歴情報，静的情報

Feature location using static infomation and revision history

Yuki HIKAWA†, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1-3. Machikaneyama-cho, Toyonaka, Osaka, 560-8531, Japan
E-mail: †{y-hikawa,matusita,inoue}@ist.osaka-u.ac.jp

Abstract　In the software maintenance or development, we have to find the part which is implemented one feature to the software comprehension or the software reuse, and Feature Location is the technique which find the part of the source code which is implemented one feature. In this paper, we propose the feature location approach using static infomation and revision history. We perform an experimental study for the open source software to evaluate our approach. The result shows that our approach is useful.

Key words　software comprehension, software reuse, feature location, revision history, static infomation

1. はじめに

ソフトウェア保守プロセスは，ソフトウェア理解と実際の修正作業に分けることができる．修正作業には，発見されたバグの修正や要求の機能拡張が含まれるが，実際に修正を行う際には，ソフトウェアについて理解しておく必要がある．しかし，ソフトウェアの大規模化や複雑化により，ソフトウェア全体を理解するのには多大な時間やコストがかかるため困難な作業となる．そこで，ソフトウェア全体を理解するのではなく，これから修正する機能に関連する部分のみを理解することができれば，ソフトウェア理解にかかるコストを大きく削減することができる [4]．そこで，理解するべき修正に必要な特定の機能に関連する部分を特定する必要がある．しかし，機能に関連する部分の特定にはソフトウェア全体の構成を知る必要があり，大規模かつ複雑なソフトウェアで行うのは困難である [8]．

一方，ソフトウェア開発においては，大規模かつ高品質なソフトウェア製品を短期間で開発することが求められている．そうした要求を実現する方法の 1 つとして，再利用がある [6]．再利用は，過去に作られたソフトウェア部品を利用して開発を進める方法で，再利用を行うことにより，高品質なソフトウェアを短期間で効率良く開発することができる．しかし，実際にある機能を実装する際に過去に作った他のソフトウェアの一部を再利用する際，再利用に必要なその機能を実装している部分を特定する必要があるが，ソフトウェア理解と同様に困難な作業

である．

このように，機能が実装されている部分を特定することが必要になるが，困難な作業である．そこで，ある機能が実装されている部分を効率的に特定する手法として，Feature Location と呼ばれる手法が研究されている [1]〜[3], [7], [9].

Feature Location の手法としてさまざまな手法が提案されており，それらの手法は静的手法，動的手法，更新履歴情報を用いた手法の 3 つにわけられるが，それぞれの手法ごとに異なる問題点が存在し，抽出する機能によって適切に抽出できない場合がある．

そこで本研究では，あるクラスと同一の機能が実装されている部分を特定する手法として，既存手法のうち，更新履歴情報を用いた手法に，静的手法を組み合わせて用いる手法を提案する．具体的には，まず，更新履歴情報を用いてあるクラスと同一の機能が実装されているクラス群を抽出する．そして，静的情報を用いて更新履歴情報から誤って抽出したクラス群の除去，及び，更新履歴情報からは抽出できなかったクラス群の補完を行う．この手法により，それぞれの問題点を補い，機能抽出の精度を上げることを目指す．

以降，2. 節では Feature Location の関連研究を紹介する．3. 節では提案手法について説明する．4. 節では 3. 節で説明した提案手法に対して行った適用実験について述べ，5. 節で実験結果の考察を述べる．最後に，6. 節でまとめと今後の課題を述べる．

2. 関 連 研 究

効率的にソフトウェアの持つ特定の機能がソースコード中でどの箇所に実装されているかを特定する手法として，Feature Location についての研究がある [1]〜[3], [7], [9]．Feature Location を用いることで，保守作業や開発作業において，特定の機能を実装している箇所を特定するコストを削減する事ができる．

既存の Feature Location の手法は大きく静的手法，動的手法，更新履歴情報を用いた手法の 3 つに分けることができる．

静的手法では，対象とするソフトウェアのソースコードを静的に解析し，モジュールのコールグラフを作成し，そのグラフを解析することで特定の機能を実装している箇所を抽出する [1], [9]．しかし，静的手法はいずれもソースコードの解析結果を元にして特定の機能を実装している箇所を見つけ出すため，実行経路が動的に決定されるオブジェクト指向プログラムにそのまま適用する事はできない．動的束縛は，オブジェクトが起動するメソッドを変数の型ではなく実行時の実体をもとに決定するため，実際に呼び出されるメソッドがソースコードを解析しただけではわからないためである．

動的手法では，あらかじめテストケースを用意しておき，そのテストケースを入力として実行したときの呼び出されたモジュールの履歴を解析することで特定の機能を実装している箇所を抽出する [2], [3]．しかし，動的手法は最初にテストケースを作る必要があり，また，テストケースの品質に抽出の精度が左右されてしまうため，慎重にテストケースを作成する必要があり，テストケース作成のコストが膨大になる．また，ソフトウェアを何度も実行する必要があるため，大規模なソフトウェアに適用する場合，非常に時間がかかる．

更新履歴情報を用いた手法では，版管理システム等での開発履歴情報を用いて，同時更新の傾向を解析することにより特定の機能を実装している箇所を抽出する [7]．しかし，更新履歴情報を用いた手法は，開発履歴情報を蓄えてある必要があり，版管理システム等を使って開発履歴を管理していないソフトウェアには適用することができなく，管理のされ方によっても精度が大きく変わってしまう．また，開発履歴のみを解析し，ソースコードには全く依存しないため，たまたま他の機能と同時に更新されたが，実際には機能とは関係のないモジュールが多く抽出されてしまう可能性がある．

3. 提 案 手 法

本節では，更新履歴情報と静的情報を用いてある特定のクラスと同一の機能を実装しているクラス群を抽出する手法について述べる．以下では，同一機能を抽出する際に最初に用いるある特定のクラスを開始クラスと呼ぶ．

更新履歴情報としてクラス間で同時に更新される割合を表した更新傾向グラフを用いる．ソフトウェア開発において同じ機能を実装しているクラスは同時に更新されることが多いと考えられるため，同時更新の傾向を解析することにより同一機能を実装しているクラスを特定できる．また，静的情報としてクラス間の呼び出し関係を表したコールグラフを用いる．呼び出し関係という直接的な関係を解析することにより同一機能を実装しているクラスを特定できる．本手法では，まず，更新傾向グラフで開始クラスと同時更新傾向が強いクラス群を抽出し，その抽出結果に，コールグラフ上で特定の呼び出し関係のあるクラス群を抽出し補完することにより開始クラスと同一の機能を実装しているクラス群を抽出する．

本手法は，以下の 4 つの手順から構成されている．

（1） 開始クラスの決定
（2） 更新傾向グラフ，コールグラフの作成
（3） 更新傾向グラフからの抽出
（4） 抽出結果とコールグラフの比較

以下，これら 4 つについて詳しく説明する．

3.1 開始クラスの決定

抽出したい機能のクラス群の中の 1 つのクラスを開始クラスとして決定する．開始クラスの決定は，機能から連想されるキーワードを用いた検索などによって行う．

3.2 更新傾向グラフ，コールグラフの作成

更新履歴情報として用いる更新傾向グラフと静的情報として用いるコールグラフを作成する．

3.2.1 更新傾向グラフの作成

更新傾向グラフとは，クラス間で同時に更新される傾向を表現するグラフである．ここでは，同じコミット作業により更新されたとき，同時に更新されたと考える．グラフの頂点はクラスを表現する．頂点間の辺は有向辺であり，辺の始点が頂点 c_1，終点が頂点 c_2 であるとき，同時に更新される傾向を表現するメトリクスである $confidence_{c_1,c_2}$ という値を持っている．

表 1　confidence 値の例

	A	B	C	D
A	1.0	0.75	0.5	0.25
B	1.0	1.0	0.33	0
C	0.5	0.25	1.0	0.75
D	0.33	0	1.0	1.0

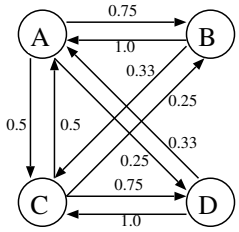

図 1　更新傾向グラフの例

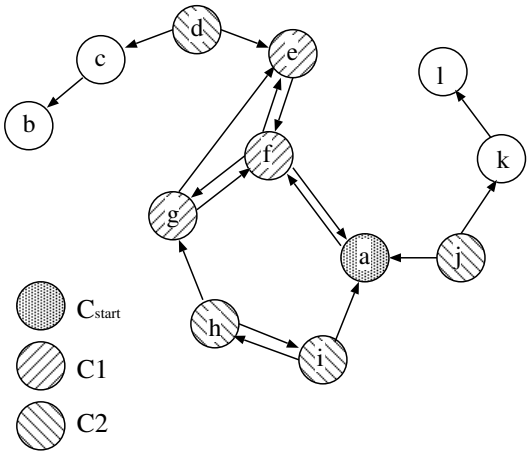

図 2　更新傾向グラフからの抽出例

confidence メトリクス [10] は，ある 2 つのクラス c_1，c_2 について，c_1 が更新されたとき，c_2 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．ソフトウェア中の全クラス集合を C とし，任意のクラス $c_1, c_2 \in C$ について，

$$confidence_{c_1,c_2} = \frac{c_1\text{と } c_2\text{が同時に更新された回数}}{c_1\text{が更新された回数}} \quad (1)$$

更新傾向グラフの作成では，まず，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，更新時のコメントの情報を用いて同時に更新されたクラスを求める．そして，求めた同時更新情報を用いて，クラス間の confidence 値を求め，グラフとして表現する．

更新傾向グラフの作成の具体例を挙げる．クラス A，B，C，D が次のような順に更新されたとする．

（1） A，B，C を同時に更新する
（2） A，B を同時に更新する
（3） C，D を同時に更新する
（4） A，B を同時に更新する
（5） A，C，D を同時に更新する
（6） C，D を同時に更新する

このとき，各クラス間の confidence の値は表 1 のようになる．表 1 の各セルは $confidence_{\text{行, 列}}$ を表している．そして，図 1 でこれを更新傾向グラフとして表現した．

3.2.2　コールグラフの作成

コールグラフとは，クラス間の呼び出し関係を表現したグラフである．グラフの頂点はクラスを表現する．頂点間の辺は有向辺であり，辺の始点が頂点 c_1，終点が頂点 c_2 であるとき，クラス c_1 内のメソッド，あるいは変数はクラス c_2 内のメソッド，変数を呼び出している．

ソースコードを静的に解析し，クラス間の呼び出し関係を抽出し，グラフとして表現する．

3.3　更新傾向グラフからの抽出

3.2.1 節で作成した更新傾向グラフからクラス群の抽出を行う．

開発や保守の段階において，1 つの機能を実装，修正する際には同時に関係するクラスについて作業するため，同時に更新されたクラスは同一機能を実装していることが多いと考える．そこで，同一機能を実装しているクラス群として，開始クラスと同時更新されることが多いクラス群，すなわち，confidence の値が大きいものを抽出する．

抽出方法は，あらかじめ閾値 E_{th} を決め，以下のような手順で行う．開始クラスを c_{start} とする．

（1） $\forall c_i \forall c_j confidence_{c_i,c_j} < E_{th}$ となる有向辺を削除する
（2） c_{start} から到達可能な頂点集合 $C1$ を求める．
（3） c_{start}，$C1$ の各頂点のいづれかに到達可能なクラス群 $C2$ を求める．
（4） c_{start}，$C1$，$C2$ の和集合を抽出クラス群とする．

閾値については，対象とするソフトウェアによって更新のされ方等が異なり閾値自体も異なってくるため，抽出を行う利用者によって抽出された数を見たり，抽出結果をコールグラフと比べることにより判断する．

更新傾向グラフからの抽出の具体例を図 2 を用いて示す．図 2 は更新傾向グラフから閾値以下の有向辺を削除したグラフである．開始クラス a から到達可能な頂点は e, f, g でこの 3 つが $C1$ となる．次に，a, e, f, g のいづれかの頂点に到達可能な頂点は d, h, i, j でこの 4 つが $C2$ である．したがって，このグラフから抽出されるクラス群は $\{a, d, e, f, g, h, i, j\}$ となる．

3.4　抽出結果とコールグラフの比較

3.2.2 で作成したコールグラフを用いて 3.3 節で更新傾向グラフから抽出したクラスについて誤って抽出したクラスの除去，及び，抽出できなかったクラスの補完を行う．

3.4.1　誤って抽出したクラスの除去

ここでは，3.3 節の更新傾向グラフからの抽出では抽出されたが，コールグラフ上では同一機能を実装していないとみなせるクラス群を除去する．除去にはコールグラフ上の開始クラスからの抽出クラス距離を用いて行う．クラス c_1 と c_2 間の抽出クラス距離 d_{c_1,c_2} はコールグラフを無向辺と考えたときの c_1 と c_2 のパスの間に抽出されていないクラスがいくつあるかの最小値を表すものと定義する．例えば，図 3 の d と f の抽出クラス距離 $d_{d,f}$ は頂点 d と f の間に抽出されていないクラス

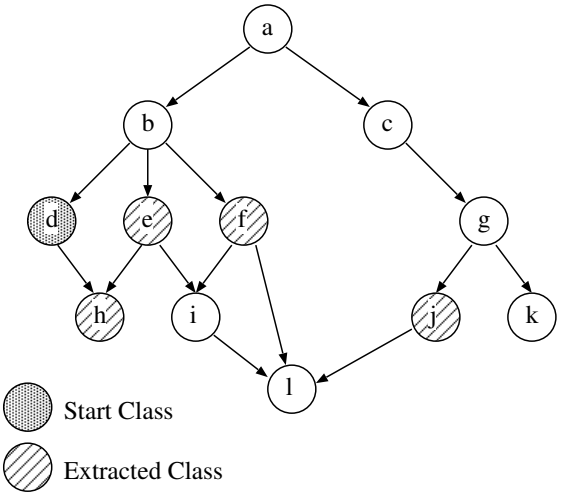

図 3　誤って抽出したクラスの除去例

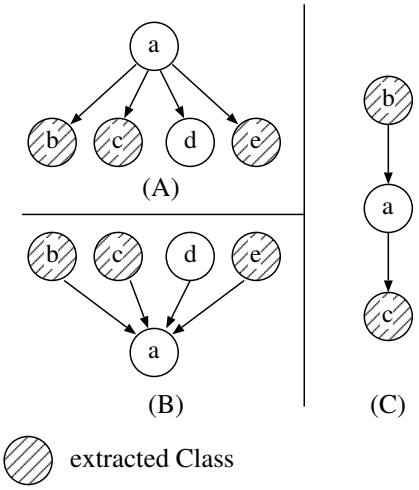

図 4　抽出できなかったクラスの補完例

b が存在するので，$d_{d,f} = 1$ となる．d と e の抽出クラス距離 $d_{d,e}$ は頂点 d と e の間は $d \to h \to e$ というパスを通ればすべてのクラスが抽出されているので，$d_{d,e} = 0$ となる．

この抽出クラス距離を用い，開始クラス c_{start} と $c \in E$（E は抽出クラス集合）について $d_{c_{start},c} \geqq 2$ となる c を除去する．

図 3 の例では，開始クラスは d，抽出クラス群 $\{e, f, h, j\}$ であるので，開始クラス d と抽出クラス群の各クラスとの抽出クラス距離を求めると，$d_{d,e} = 0$，$d_{d,f} = 1$，$d_{d,h} = 0$，$d_{d,j} = 2$ となる．したがって，抽出クラス距離が 2 となった j を除去する．

3.4.2　抽出できなかったクラスの補完

ここでは，3.3 節の更新傾向グラフからの抽出で抽出されなかったクラス群をコールグラフを用いて補う．

まず，次の条件のいずれかを満たすクラスを抽出する．

- 呼び出しているクラスのうち 3 つ以上のクラスが抽出されているクラス
- 呼ばれているクラスのうち 3 つ以上のクラスが抽出されているクラス
- 呼び出しているクラスのいずれかが抽出されており，かつ，呼ばれているクラスのいずれかが抽出されているクラス

図 4(A) のクラス a のように呼び出しているクラスのうち 3 つ以上が抽出されているクラスは，その機能を統括しているようなクラスと考えられる．図 4(B) のクラス a のように呼ばれているクラスのうち 3 つ以上が抽出されているクラスは，その機能に必要なライブラリ的な機能を実装していると考えられる．図 4(C) のクラス a のように呼び出しているクラスのいずれかが抽出されており，かつ，呼ばれているクラスのいずれかが抽出されているクラスは，実際にその機能の一部を実装していると考えられる．これらの更新傾向グラフからは抽出できなかったが，実際にはその機能の一部を実装していると考えられる 3 種類のクラス群を抽出クラス群に加える．

最後に，さらに呼び出してるクラスと呼ばれているクラスのすべてが抽出されてるクラスは，その機能に何らかの関係があると考えられるので抽出する．

これらの手順で最終的に抽出された結果を同一機能を実装しているクラス群とする．

4. 適 用 実 験

本手法の妥当性を検討するために，Java を対象に本手法を実装したツールを作成し，そのツールを用いてオープンソースソフトウェアの XBrowser [11] を対象に適用実験を行った．XBrowser は，Java によって記述されたウェブブラウザであり，規模は，クラス数 171 クラス，行数 28881 行，総更新回数 165 回，総リビジョン数 1388 個である．

実験では，XBrowser に実装されている機能のうちブックマーク機能とヒストリ機能の抽出を行った．ブックマーク機能は，何度も訪れる Web サイトの URL を記録しておく機能で，ユーザが指定したページをブックマークとして保存することができ，ユーザはブックマークとして保存したページをブックマークリストから選択することで，そのページに簡単にアクセスできるという機能である．ヒストリ機能は，ユーザが過去に閲覧したページの履歴を保持しておき，過去に開いたことのあるページへ戻ったり，その後，戻る前のページに戻ることを可能にする機能である．

それぞれの機能に対して，静的情報，更新履歴情報単独での抽出と提案手法を比較するために次の 3 種類の抽出を行った．

- 更新履歴情報のみを用いて，グラフ作成後，3.3 節の更新傾向グラフからの抽出のみを行った．
- 静的情報のみを用いて，グラフ作成後，3.4 節のコールグラフからの抽出のみを行った．ただし，提案手法では抽出クラスが 1 つのとき静的情報からの補完を行っても，ほとんど抽出できないため，抽出できなかったクラスの補完で,「呼び出しているクラスのうち 3 つ以上のクラスが抽出されているクラス」を「呼び出しているクラス」,「呼ばれているクラスのうち 3 つ以上のクラスが抽出されているクラス」を「呼ばれているクラス」と変更し，抽出した上で,「呼び出しているクラスのいずれかが抽出されており，かつ，呼ばれているクラスのいずれかが抽出されているクラス」を抽出するという手順に変更した．

表 2　ブックマーク機能の抽出結果

	正解クラス	抽出クラス	該当クラス	適合率	再現率
更新履歴情報のみを用いての抽出	34	10	10	1.00	0.29
静的情報のみを用いての抽出	34	32	16	0.50	0.47
提案手法	34	29	25	0.86	0.74
更新傾向グラフから抽出		10	10	1.00	——
誤って抽出したクラスの除去		0	0	0.00	——
抽出できなかったクラスの補完		19	15	0.79	——

表 3　ヒストリ機能の抽出結果

	正解クラス	抽出クラス	該当クラス	適合率	再現率
更新履歴情報のみを用いての抽出	30	15	6	0.40	0.20
静的情報のみを用いての抽出	30	31	13	0.42	0.43
提案手法	30	27	14	0.52	0.47
更新傾向グラフから抽出		15	6	0.40	——
誤って抽出したクラスの除去		2	0	0.00	——
抽出できなかったクラスの補完		14	8	0.57	——

- 提案手法の手順をすべて行った．

今回の実験では，開始クラスをブックマーク機能は，“Bookmark” というキーワードに一致した xbrowser.bookmark.BookmarkFinder クラス，ヒストリ機能は，“History” というキーワードに一致した xbrowser.history.HistoryFinder クラスとした．また，更新傾向グラフからの抽出で用いる閾値は，ブックマーク機能，ヒストリ機能共に，$E_{th} = 0.95$ とした．

実験に対する評価は，クラスごとにブックマック，ヒストリの各機能を実装しているかをソースコードから判定し，その上で，適合率，再現率を用いて評価を行った．手法によって抽出されたクラスの数を抽出クラス数，ソースコードから実際に機能を実装していると判断したクラスの数を正解クラス数，手法によって抽出されたクラスのうち実際に機能を実装していると判断したクラスの数を該当クラス数とすると，適合率，再現率の計算は次の式で行う．

$$\text{適合率} = \frac{\text{該当クラス数}}{\text{抽出クラス数}} \qquad \text{再現率} = \frac{\text{該当クラス数}}{\text{正解クラス数}} \qquad (2)$$

ブックマーク機能の実験結果を表 2 に，ヒストリ機能の実験結果を表 3 に示す．

5. 考　　察

3.1 節の開始クラスの決定方法として，今回は機能名を用いてクラス名のキーワード検索を行った．今回の方法以外に用いることが可能な方法としては，ソースコード部品の検索システムなどを用いるということが考えられる．ソフトウェア部品検索システムとは，ソフトウェア部品特有の性質を利用した検索システムで，クラス名を用いたキーワード検索より機能の特徴を反映した開始クラスを選ぶことができる．

今回の適用実験では，ある程度の同一機能を実装しているクラス群を抽出できた．しかし，提案手法は 2 つの情報を用いており，両方の情報が必要になる．例えば，更新回数が著しく少ないソフトウェアに対しては，更新履歴情報からの抽出ができず，静的手法からの抽出はできるものの精度が下がってしまう．

次に各機能についての考察を述べる．

5.1　ブックマーク機能

更新履歴情報のみで抽出した場合は，適合率は 1.0 となり完全なものの再現率は低くなっている．これは，同一機能を実装しているクラスが必ずしも全体で同時に更新されるわけではないため，同時に更新されることが少なかったクラスを取りこぼしてしまった．また，1 つのブックマーク機能でも更にいくつかの細かな機能に細分化され細かな機能の単位で更新が行われたため，細かな機能の単位で抽出されたためだと考えられる．

静的情報のみで抽出した場合は，多くのクラスを抽出することができ，再現率は更新履歴情報のみより高いものの，ブックマーク機能とは関係のないクラスも多くとってしまい適合率が低くなってしまった．これは，今回の手法では，そのクラスを呼び出しているクラスとそのクラスに呼び出されいてるクラスをすべて抽出してしまうが，実際には別の機能を呼び出したり別の機能に呼ばれるすことも多いので機能とは関係のないクラスも抽出してしまったからだと思われる．

提案手法全体を行った場合は，更新履歴情報のみ，静的情報のみに比べ，適合率，再現率ともに高い値になった．

手順の各段階での抽出結果を見てみると，提案手法の更新履歴情報を用いる部分まででは，すべて実際に機能を実装しているクラスを抽出できているものの，実際に機能を実装しているクラスのうちの一部分しか抽出することができなかった．しかし，静的情報を用いることにより更新履歴情報から抽出できなかったクラスの補完を行うことで，同時更新傾向が低く抽出できなかったクラス群の多くを抽出することができ，再現率を飛躍的に上げることができた．

3.4.2 節で，3 つ以上の抽出クラスから呼ばれている，あるいは呼んでいるとした．これは，その機能を実装している複数のクラスと接続している場合，同じ機能を実装していると考えられるためだが，2 つとした場合はそのクラスの他の機能と関係している場合も考えうるということからクラス数を 3 とした．このように設定した結果，誤って抽出することなく，同一機能

を実装しているクラスを抽出できた．しかし，どのソフトウェアでも 3 という値を用いることができるかは今後検証していく必要があると考える．

また，提案手法を行った結果，単独で手法を適用するより再現率は高くなったものの，まだ機能を実装しているクラスを完全に抽出することはできていない．そのため，適合率をあまり下げることなく再現率を上げるためにコールグラフを用いて抽出できなかったクラスを補完していく方法を改良していく必要があると考える．

5.2 ヒストリ機能

ヒストリ機能の抽出も，ブックマーク機能同様，提案手法の適合率，再現率が最大となった．

しかし，ブックマーク機能に比べ，適合率，再現率ともに低く，特に再現率は 50%以下となってしまった．この原因として考えられるのは，更新履歴情報からの抽出がうまくいかず，さらに，コールグラフを用いての誤って抽出したクラスの除去もあまり効果的にできなかったためである．更新履歴情報からの抽出がうまくいかなかった原因は，ヒストリ機能は，開いたページの履歴を取得する際，他の多くの機能と直接関係しており，開発や保守においてヒストリ機能を実装，修正する際，同時に他の機能を実装しているクラスも更新することが多く，ヒストリ機能を実装していないクラスとの間の confidence も高くなったため抽出されてしまったためと考えられる．さらに，コールグラフから誤って抽出したクラスの除去では，除去することができた 2 つともヒストリ機能を実装しておらず正確に除去できたものの，9 個ある誤って抽出したクラスのうち 2 個しか除去することができなかった．これは，更新履歴情報から誤って抽出されたクラスと開始クラスの抽出クラス距離が近いものが多かったため本手法での除去では効果的に除去することができなかったためと考えられる．

また，十分に誤って抽出したクラスを除去できなかったため，補完する際，誤って抽出したクラスがノイズとなり，そのクラスの周辺の別の機能を実装しているクラスを抽出してしまったため，適合率も下げてしまうことになった．

今後，更新傾向グラフから誤って抽出することが少なく，機能実装クラスを多く抽出できるように工夫し，また，抽出クラス距離以外のメトリクスも用いることによりさらに多くの誤って抽出したクラスの除去を行えるようにしたい．

6. ま と め

本研究では，あるクラスと同一の機能が実装されている部分を特定する手法として，更新履歴情報を用いた手法に静的手法を組み合わせて用いる手法を提案した．具体的には，更新履歴情報として，クラス間の同時更新の傾向を表現する更新傾向グラフを用い，同時に更新されるものが多いクラスを抽出した．さらに，静的情報として，クラス間の呼び出し関係を表現するコールグラフを用い，誤って抽出したクラスの除去と抽出できなかったクラスの補完を行った．また，提案手法について適用実験を行い，静的手法と更新履歴を用いた手法を組み合わせた結果，手法を単独で実行するときより高い適合率，再現率で特定の機能を実装している箇所を特定できることを確認した．

今後の課題として以下のことがあげられる．

- 更新傾向グラフからの抽出で，誤って他の機能を実装しているクラスも抽出してしまい，補完を効果的に行うことができなかった．更新傾向グラフからの抽出手法について改良していき，精度を上げていきたい．
- 誤って抽出されたクラスの除去の精度が悪く，更新履歴情報からの抽出で誤ってクラスを抽出してしまった場合，適合率と再現率を大きく下げることになった．そこで，抽出クラス距離以外のメトリクスを用いることでより効果的に除去を行えるように考えていきたい．
- 提案手法では，適合率は高かったものの，再現率はあまり高くなかった．そこで，抽出できなかったクラスの補完の方法として，現在の 3 つの条件以外に適当な条件がないかを検討していきたい．
- 現在はクラス単位での抽出のみしか実装していないため，メソッド単位やパッケージ単位での抽出も実装していきたい．

文　献

[1] K. Chen and V. Rajlich: “Case Study of Feature Location Using Dependence Graph”, In *Proceedings of the International Workshop on Program Comprehension (IWPC’00)*, pp.241–249, 2000.

[2] T. Eisenbarth, R. Koschke, and D. Simon: “Locating Features in Source Code”, *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING (Vol. 29, No. 3)*, pp.210–224, 2003.

[3] A. D. Eisenberg and K. D. Volder: “Dynamic Feature Traces: Finding Features in Unfamiliar code”, In *Proceedings of the International Conference on Software Maintenance (ICSM’05)*, pp.337–346, 2005.

[4] K. Erdos and H. M. Sneed: “Partial comprehension of complex programs(enough to perform maintenance)”, In *Proceedings of the International Workshop on Program Comprehension(IWPC’98)*, pp.98–105, 1998.

[5] A. E. Hassan and R. C. Holt: “Using Development History Sticky Notes to Understand Software Architecture”, In *Proceedings of the International Workshop on Program Comprehension (IWPC’04)*, pp.183–192, 2004.

[6] I. Jacobson, M. Griss, and P. Jonsson: “Software Reuse”, Addison-Wesley Pub, 1997.

[7] 楠田: “メソッドの同時更新履歴を用いたクラスの機能別分類法”，大阪大学 大学院情報科学研究科 修士学位論文，2006.

[8] J. J. Marciniak: “Encyclopedia of Software Engineering”, John Wiley & Sons, 1994.

[9] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang: “SNIAFL: Towards a Static Non-Interactive Approach to Feature Location”, In *Proceedings of the International Conference on Software Engineering (ICSE’04)*, pp.293–303, 2004.

[10] T. Zimmermann, S. Diehl, and A. Zeller: “How history justifies system architecture (or not)”, In *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE 2003)*, pp.73–83, 2003.

[11] “XBrowser”. http://xbrowser.sourceforge.net/