修士学位論文

題目

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

指導教員
井上 克郎 教授

報告者
楠田 泰三

平成 18 年 2 月 13 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻 ソフトウェア工学講座

平成 17 年度 修士学位論文

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

楠田 泰三

内容梗概

ソフトウェアの保守作業にかかるコストの増大に伴ない，保守対象となるソフトウェアの機能やその構造を理解するための支援手法について研究が広く行われている．Feature Location と呼ばれる手法では，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し，保守作業において理解する必要のある箇所を特定することでソフトウェア理解のコストを軽減する．しかし，これまでに提案された Feature Location の手法は，オブジェクト指向ソフトウェアを対象とした場合正確な結果が得られず，また解析に時間がかかるため，あまり利用されていなかった．

本研究では，オブジェクト指向ソフトウェアを対象として，機能と実装箇所の対応を特定することを目指し，同じ機能を実装している箇所は更新が同時に行なわれる傾向が強いという仮定に基づいて，ソフトウェアの更新履歴を用いてクラスを機能別に分類する手法の提案を行う．本手法では，まず開発履歴情報を参照し，どのメソッドが同時に更新されてきたかという履歴を抽出する．次に，メソッド間の同時更新の傾向を表現したグラフである更新傾向グラフを作成する．最後に，更新傾向グラフに対しクラスタリングを行なうことで，同時に更新される傾向が強いメソッド群を特定し，その結果を基にクラスを分類する．また，オブジェクト指向言語 Java を対象に本手法を適用するためのシステムを作成した．さらに，作成したシステムを用いて 2 つのソフトウェアに対する適用実験を行い，本手法の有用性を確認した．

主な用語

ソフトウェア理解
Feature Location
オブジェクト指向
開発履歴情報

目 次

1 まえがき

近年，ソフトウェアの大規模化，複雑化に伴い，ソフトウェアの保守作業はますます困難になり，そのコスト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 [13]．ソフトウェアの保守作業は，ソフトウェアの出荷後に行なわれる機能の追加や修正のためのソフトウェアの変更作業を含んでいるが，ソフトウェアに対し変更を及ぼす際は対象となるソフトウェアの理解が必要不可欠となる．よって，ソフトウェアの理解は保守作業において非常に重要な作業であるが，保守対象となるソフトウェアが大規模であった場合，そのソフトウェアを完全に理解する事は非常に難しく，多大なコストを要する作業となる．そのため，ソフトウェア保守のコストの大部分を保守対象となるソフトウェア理解に費やすこととなる [22]．従って，ソフトウェア理解のコストを軽減することはソフトウェアライフサイクル全体のコストを軽減する事に繋がると考えられ，ソフトウェアの理解支援についての研究が行なわれている．

一方で，ソフトウェアの保守を行なう時にはソフトウェア全体に対する理解よりも，その保守の作業に応じたソフトウェアの一部の詳細な理解の方が重要であると言う事も考えられている [11]．この考えに基づき，Feature Location と呼ばれるソフトウェア理解支援手法が研究されている [5][8][9][32][33]．Feature Location とは，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し，保守作業において理解する必要のある箇所を特定することでソフトウェア理解のコストを軽減する．

しかし，Feature Location の手法は主に手続き型言語で記述されたソフトウェアを対象としており，動的に実行経路が決定するオブジェクト指向ソフトウェアにそのまま適用する事は難しい．ソースコードの解析結果を用いる静的手法の場合，正確な実装箇所が判断できず，実際にソフトウェアを実行してその結果を用いる動的手法の場合は解析に時間がかかるためである．

一方で，ソフトウェアを効率的に管理，開発を行なうために版管理システムというシステムを用いた開発が広く行なわれている．版管理システムは，リポジトリというデータベースを用いてソフトウェアプロダクト (ソースコード，ドキュメントなど) を管理する．リポジトリには最新のソフトウェアプロダクトだけでなく，そのソフトウェアプロダクトに対して行なわれてきた全ての更新の情報も保存されている．この更新の情報には，いつ，誰が，どの箇所を更新してきたかという情報が含まれている．

ソフトウェアの機能を追加，修正するためにはソースコードを更新する必要がある．版管理システムを用いてソフトウェアの開発を行なっている場合，機能の追加，修正作業で行なったソースコードの更新の情報はリポジトリに保存されている．また，機能の追加，修正作業でソースコード中で複数の箇所を更新する必要が生じた場合，それらの箇所は一度に更

新される可能性が高いと考えられる．

そこで本研究では，オブジェクト指向言語で記述されたソフトウェアの機能と実装箇所の対応を特定することを目指し，ソフトウェアのクラスを機能別に分類するための手法の提案を行なう．具体的には，ソフトウェア中で同じ機能を実装している箇所は同時に更新される傾向が強いと考え，同時に更新される傾向が強いクラス群をグループ化する．

以降 2 節ではソフトウェア保守とソフトウェア理解支援手法 Feature Location について述べる．3 節ではソフトウェア開発で用いられている版管理システムについて説明を行なう．4 節では本研究で提案する手法について詳しく説明を行なう．5 節では 4 節で説明を行なった手法を実装したシステムについて述べ，6 節では作成したシステムを用いた適用実験について述べる．7 節では本研究の関連研究について述べ，最後に 8 節でまとめと今後の課題を述べる．

2 ソフトウェア理解支援

2.1 ソフトウェア保守

ソフトウェア保守とは,“ 納入後，ソフトウェア・プロダクトに対して加えられる，フォールト修正，性能または他の性質改善，変更された環境に対するプロダクトの適応のための改訂 ”であると定義されている [18]．ソフトウェア保守は以下の 4 つの作業に分類することができる．

- 修正保守 (corrective maintenance)
  ソフトウェア・プロダクトに対して発見された障害の原因を特定し，障害に対処をするために行なうソフトウェア・プロダクトの改変.
- 適応保守 (adaptative maintenance)
  ソフトウェア・プロダクトが動作するために必要な OS，ハードウェアなどの周辺環境の変更が起きた際に，ソフトウェア・プロダクトを続けて使用可能なように維持するために行なうソフトウェア・プロダクトの改変.
- 改善保守 (perfective maintenace)
  性能または保守性を改善するために行なうソフトウェア・プロダクトの改変.
- 予防保守 (preventive maintenance)
  ソフトウェア・プロダクトのなかに潜む潜在的なフォールトが顕在化する前に，それを検出し，対処するために行なうソフトウェア・プロダクトの改変.

ソフトウェア保守は，ソフトウェアライフサイクルコストの大部分を占めているため，ソフトウェア保守のコストを下げることは非常に重要であると考えられている [13].

多くの場合ソフトウェアを適切に運用するため，開発と保守を同じチームが行なうが，保守のアウトソーシング (社外調達) も主要な産業になりつつある．例えば大企業では，非公開としたいビジネス中核となるソフトウェアを除いては，ソフトウェア保守を含めて，運用をアウトソーシングすることも多い．このような場合，開発者は通常コードを説明しなければならない場には居合わせないことが多く，変更も文書化されていないことも多い．したがって，保守者は，ソフトウェアに関して制限された理解しか得ることができず，ソースコードを自分で読まねばならない．文献 [22] では保守作業のおおよそ 40%から 60%は，改変すべきソフトウェアの理解に費やされていると指摘されている．したがって，保守作業の効率を高めること，さらにソフトウェアの理解を効率的に行なう事は，ソフトウェア工学における重要な課題の一つとなっている．

2.2 Feature Location

ソフトウェアの理解は重要な作業でありかつコストを要する作業であるために，ソフトウェア理解支援についての研究が広く行なわれているが，そのようなソフトウェア理解支援の効率的な手法の一つとして，Feature Location についての研究がある [5][8][9][32][33].

Feature Location とは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ことを目的としている．文献 [11] では，ソフトウェア保守を行なう上ではソフトウェア全体についての理解よりも，特定の保守の作業に応じたソフトウェアの一部の詳細な理解の方が重要である事が言及されている．例えば，ある機能についての修正保守を行なう場合を考えると，修正を行なう前に必要なのはソフトウェア全体ではなくその機能を実装している箇所に対する理解である．Feature Location を用いる事で，保守作業において理解する必要のある箇所を特定することでき，ソフトウェア理解にかかるコストを削減する事ができる．

Feature Location の手法は，静的手法，動的手法の 2 つの手法に分類できる．以下では，それぞれの手法の概要と問題点について述べる．

静的手法

静的手法は，対象とするソフトウェアのソースコードを解析し，手続き型プログラムにおける関数，またはオブジェクト指向プログラムにおけるメソッドの呼出グラフを作成して，一つの機能を実装している箇所を呼出しグラフのサブグラフとして与えるというものである．呼出グラフ上において接続されているどの関数，メソッドが一つの機能を実装しているものになっているかを決定する方法により，いくつかの手法が存在する．文献 [5] では，開発者自身が呼出グラフ上を辿り，グラフ上で接続されているノードが特定の機能を実装しているか否かを検討していくという手法を提案している．また，文献 [33] では自然言語で記述されたドキュメントに登場する単語と関数の識別子との類似度を求めて特定の機能を実装している関数群を抽出し，その結果を用いて呼出グラフからサブグラフの抽出を行なう手法を提案している．

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している．しかし，これらの手法は実行経路が動的に決定される動的束縛 (dynamic binding) を持つ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

動的手法は，実際にソフトウェアを様々な入力のもとで実行し，その時に呼び出される関数の系列を用いて，特定の機能を実装した箇所を見つける．文献 [32] では，ある機能を利用するいくつかのテストケースと，その機能を利用しないいくつかのテストケースを用いて，機能を実装している箇所を求めている．具体的には，ある機能 A を利用するテストケースにおいて一度でも実行された関数のうち，機能 A を利用しないテストケースでは一度も実行されないものを，機能 A を実装している関数であると考えている．文献 [10] では，テスト駆動開発を行なったソフトウェアを対象に，開発時に用いられたテストケースを用いて機能に関係するメソッドを特定している．テストケースの実行履歴から，メソッドの呼び出し関係を取得し，その呼び出し関係が現れる回数を元に特定の機能とメソッド呼び出し関係がどれほど関連しているかの計算を行なっている．これらの手法は実際の実行系列を用いるためにソースコードの内容に依存せず，動的束縛のような実行時の条件にも対応できるが，ソフトウェアを何度も実行する必要があるため，対象が大規模なものになると解析に非常に時間がかかるという問題がある．

3 版管理システム

本節では，開発履歴の取得のために使う版管理システムについて説明を行なう．

3.1 概要

版管理とは，主として以下の 3 つの役割を提供する機構である．

- プロダクトに対して施された追加・削除・変更などの作業を履歴として蓄積する．
- 蓄積した履歴を開発者に提供する．
- 蓄積したデータを編集する．

版管理手法を述べるにあたり，その基礎となるモデルが数多く存在する [1][6]．本節では，多くの版管理システムが採用している Checkout/Checkin モデルについて概要を述べる．

3.1.1 用語

ここでは，Checkout/Checkin モデルを説明するのに必要な用語の説明を行なう．

- リビジョン (revision)
  各プロダクト (ソースコード，ドキュメントなど) のある時点における状態．1 つのリビジョンには，その時点でのソースコードやリソースなどの実データと，リビジョン作成日時や作成時のメッセージログなどの属性データが格納されている．
- リポジトリ (repository)
  一つのプロジェクトの全てのリビジョンを格納する，データ格納庫．
- チェックアウト (checkout)
  リポジトリより特定のリビジョンのプロダクトを開発者の手元にコピーする操作．
- チェックイン (checkin)
  プロダクトに対して行なった変更内容をリポジトリに送り，新たなリビジョンを作成する操作．

3.1.2 The Checkout/Checkin Model

このモデルは，ファイルを単位としたリビジョンの制御に関して定義されている．リビジョン管理下にあるプロダクトはシステムに依存したフォーマット形式のファイルとしてリ

ポジトリに格納されている．開発者はそれらのファイルを直接操作するのでなく，各システムに実装されているオペレーションを介して，リポジトリとのデータ授受を行う．

このモデルでは，各開発者が以下の 3 つの作業を繰り返すことソフトウェアの開発を行なっていく．また，このプロセスを図 1 に示す．

1. リポジトリにアクセスし，必要なプロダクトをチェックアウトする．
2. チェックアウトしたプロダクトを手元で更新を行なう．
3. リポジトリにアクセスし，更新を行なったプロダクトをチェックインする．この作業により，リポジトリ内に新たなリビジョンが作成される．

3.2 現存の版管理システムの紹介

版管理システムと呼ばれるものは，多数存在する．

UNIX 系 OS では，多くの場合，RCS[30] や CVS[3][12] といったシステムが標準で利用可能となっている．ClearCase[2] のように商用のものも存在する．また，UNIX 系 OS だけではなく，Windows 系 OS においても，SourceSafe[24] や PVCS[23] をはじめ，数多く存在する．さらに，ローカルネットワーク内のみではなく，よりグローバルなネットワークを介したシステム [15] も存在する．

ここでは，それら版管理システムのうちいくつかを紹介する．

- RCS

  RCS[30] は UNIX 上で動作するツールとして作成された版管理システムであり，現在でもよく使用されているシステムである．単体で使用される他，システム内部に組み込み，版管理機構を持たせる場合などの用途もある．RCS ではプロダクトをそれぞれ UNIX 上のファイルとして扱い，１ファイルに対する記録は１つのファイルに行われる．

  RCS におけるリビジョンは，管理対象となるファイルの中身がそれ自身によって定義され，リビジョン間の差分は diff コマンドの出力として定義される．各リビジョンに対する識別子は数字の組で表記され，数え上げ可能な識別子である．新規リビジョンの登録や，任意のリビジョンの取り出しは，RCS の持つツールを利用する．

- CVS

  CVS[3][12][20][26] は RCS 同様，UNIX 上で動作するシステムとして構築された版管理システムであり，近年最も良く使われるシステムの１つである．RCS と大きく異なるのは，複数のファイルを処理する点である．また，リポジトリを複数の開発者で利

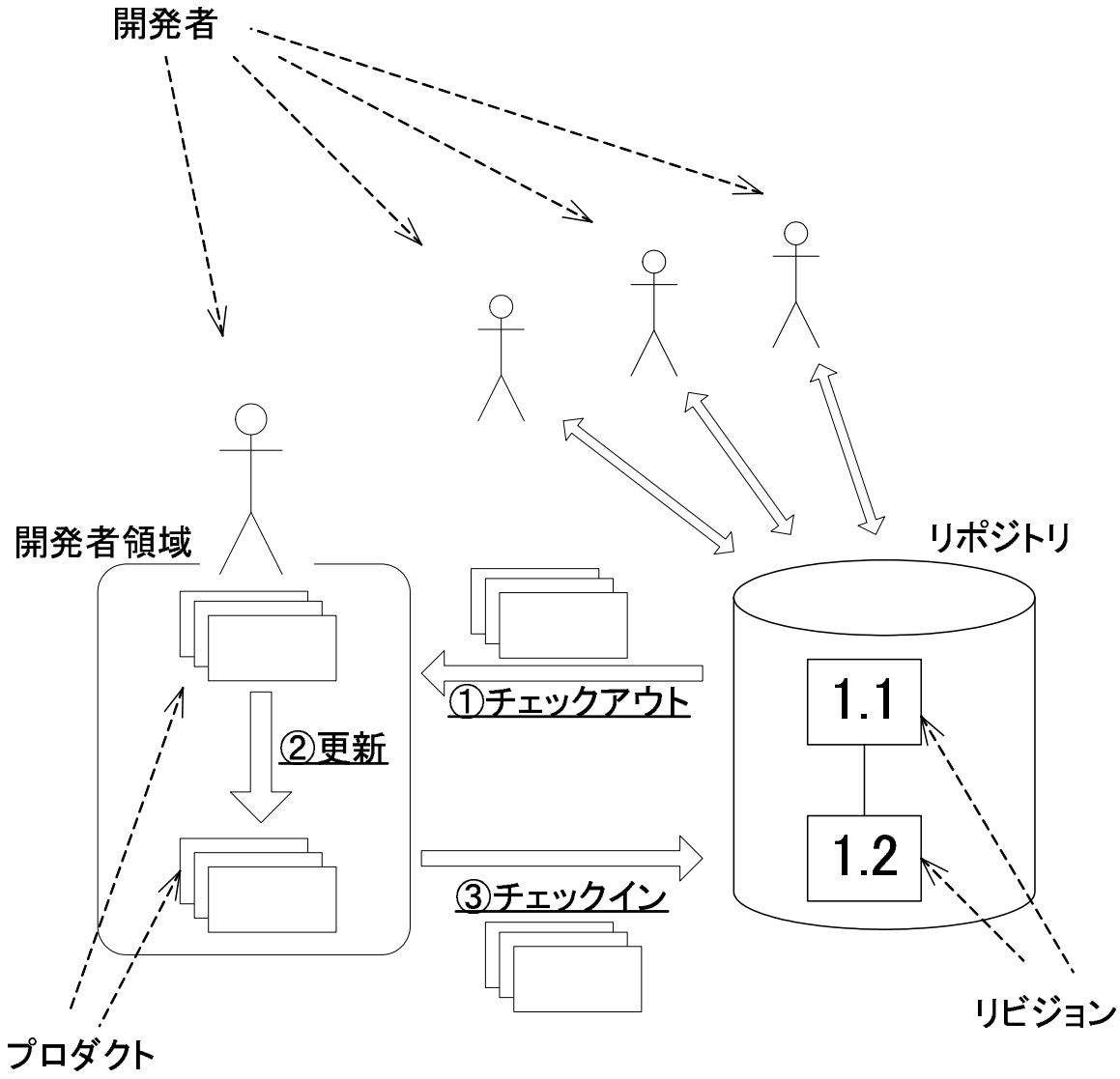

図 1: 版管理システムを用いたソフトウェア開発のプロセス

用することも考慮し，開発者間の競合にも対処可能となっている．さらに，ネットワーク環境 (ssh， rsh 等) を利用することも可能である為，オープンソースによるソフトウェア開発に用いられることが多い．その最たる例が，FreeBSD[28] や OpenBSD[29] 等のオペレーティングシステムの開発である．

- Subversion

  Subversion[31] は，CVS の持ついくつかの問題点を解決するために開発された版管理システムで，CVS の後継として注目されている．CVS ではプロダクトをファイル単位で管理していたが，Subversion では全てのプロダクトを一つのデータベースで管理する．Subversion では，CVS では考慮していなかったファイルの移動の処理を扱うことができたり，リポジトリ内のディレクトリを削除したりすることが可能になっている．またネットワークを介した利用を想定して開発されており，リポジトリへチェックインを行なう際は，新しいリビジョンを作成するのに必要な差分の情報だけが送られるため，効率が良い．

4　提案手法

本節では，ソフトウェアの開発履歴を用いてオブジェクト指向ソフトウェアのクラス群を機能別にグループ化していくための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開発履歴を用いて，同時に更新される傾向が強いメソッド群を特定し，その結果を基にクラスを機能別に分類する．本手法は以下の 6 つの作業からなる．また，本手法のおおまかな流れを図 2 で示す．

1. 同時更新情報の抽出
2. 外れ値の除去
3. 同時更新の強さを表すメトリクスの計算
4. 更新傾向グラフの作成
5. 更新傾向グラフのクラスタリング
6. 同時更新傾向の強いクラス群の抽出

以下，この 6 つの作業について詳しく説明をする．

4.1　同時更新情報の抽出

本研究では，ソフトウェアの更新の作業によって，メソッドの処理内容を記述している箇所が更新された場合，そのメソッドは更新されたものとして扱う．また，新たにメソッドが追加された場合も，メソッドの更新が行なわれたものとして扱う．

また，3.1.2 節の説明で述べたチェックインからチェックアウトまでの一連の作業を一回の更新作業と考える．通常，開発者は一回の更新作業でプロダクトの複数の箇所を更新する．一回の更新作業において更新が行なわれたメソッド群を同時更新されたメソッド群と呼ぶ．また，同時更新されたメソッド群が属しているクラス群を，同時更新されたクラス群と呼ぶ．

図 3 は，一回の更新作業において行なわれた更新と，その時の同時更新されたメソッド群を表現している．図 3 中の更新作業では，メソッド A.a1() の処理内容を記述している箇所が更新され，さらに新たにメソッド B.b2() が追加されている．よって，同時更新されたメソッド群は { A.a1() , B.b2()} となる．また，この時同時更新されたクラス群は { A , B } となる．

同時更新情報の抽出の作業では，対象とするソフトウェアの全ての更新作業について，同時更新されたメソッド群を求め，そのリストを作成する．

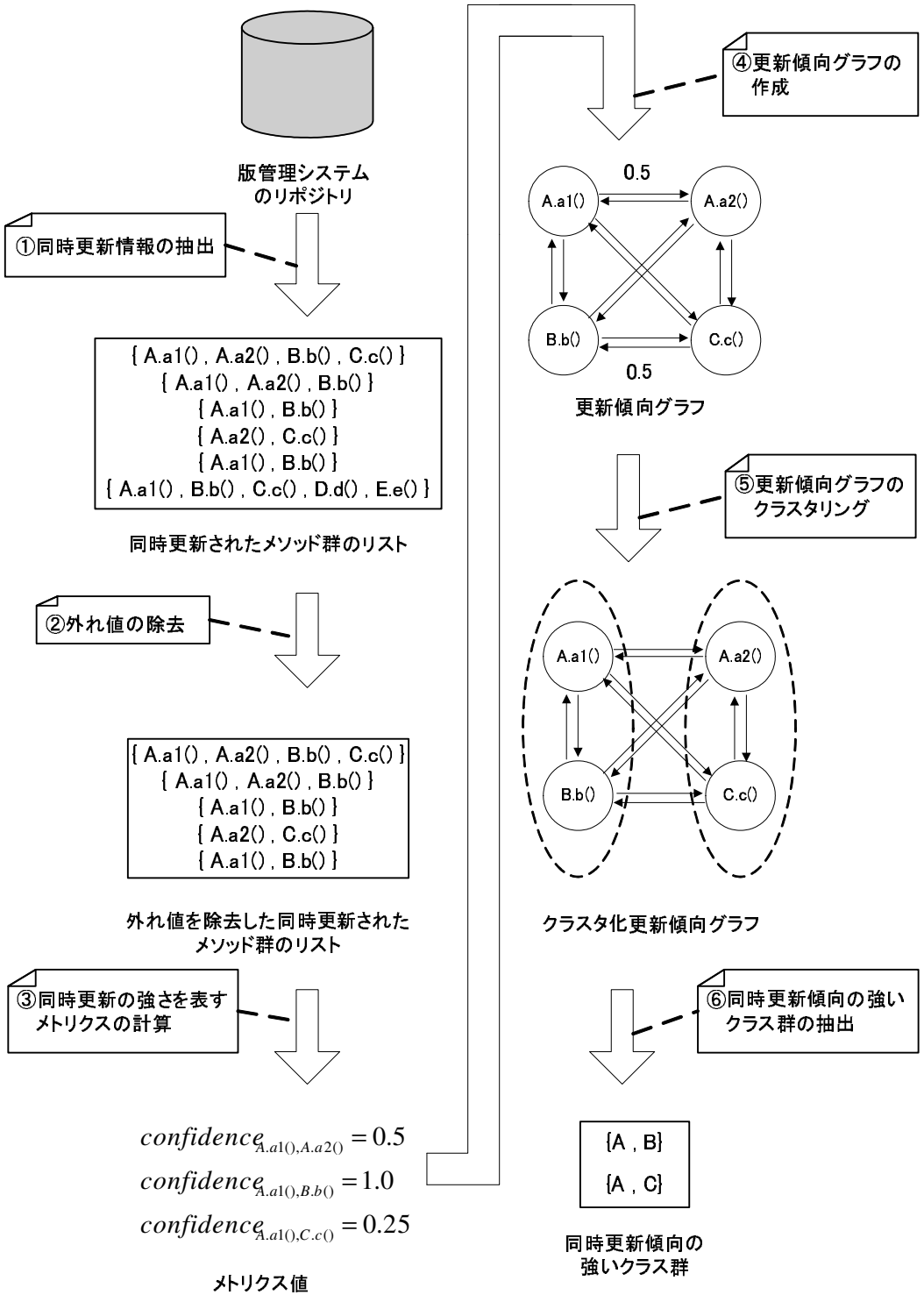
版管理システム
のリポジトリ
①同時更新情報の抽出
{ A.a1() , A.a2() , B.b() , C.c() }
{ A.a1() , A.a2() , B.b() }
{ A.a1() , B.b() }
{ A.a2() , C.c() }
{ A.a1() , B.b() }
{ A.a1() , B.b() , C.c() , D.d() , E.e() }
同時更新されたメソッド群のリスト
②外れ値の除去
{ A.a1() , A.a2() , B.b() , C.c() }
{ A.a1() , A.a2() , B.b() }
{ A.a1() , B.b() }
{ A.a2() , C.c() }
{ A.a1() , B.b() }
外れ値を除去した同時更新された
メソッド群のリスト
③同時更新の強さを表す
メトリクスの計算
confidence A.a1(),A.a2() = 0.5
confidence A.a1(),B.b() = 1.0
confidence A.a1(),C.c() = 0.25
メトリクス値
④更新傾向グラフの
作成
0.5
A.a1()
A.a2()
B.b()
C.c()
0.5
更新傾向グラフ
⑤更新傾向グラフの
クラスタリング
A.a1()
A.a2()
B.b()
C.c()
クラスタ化更新傾向グラフ
⑥同時更新傾向の強い
クラス群の抽出
{A , B}
{A , C}
同時更新傾向の
強いクラス群

図 2: 提案手法の流れ

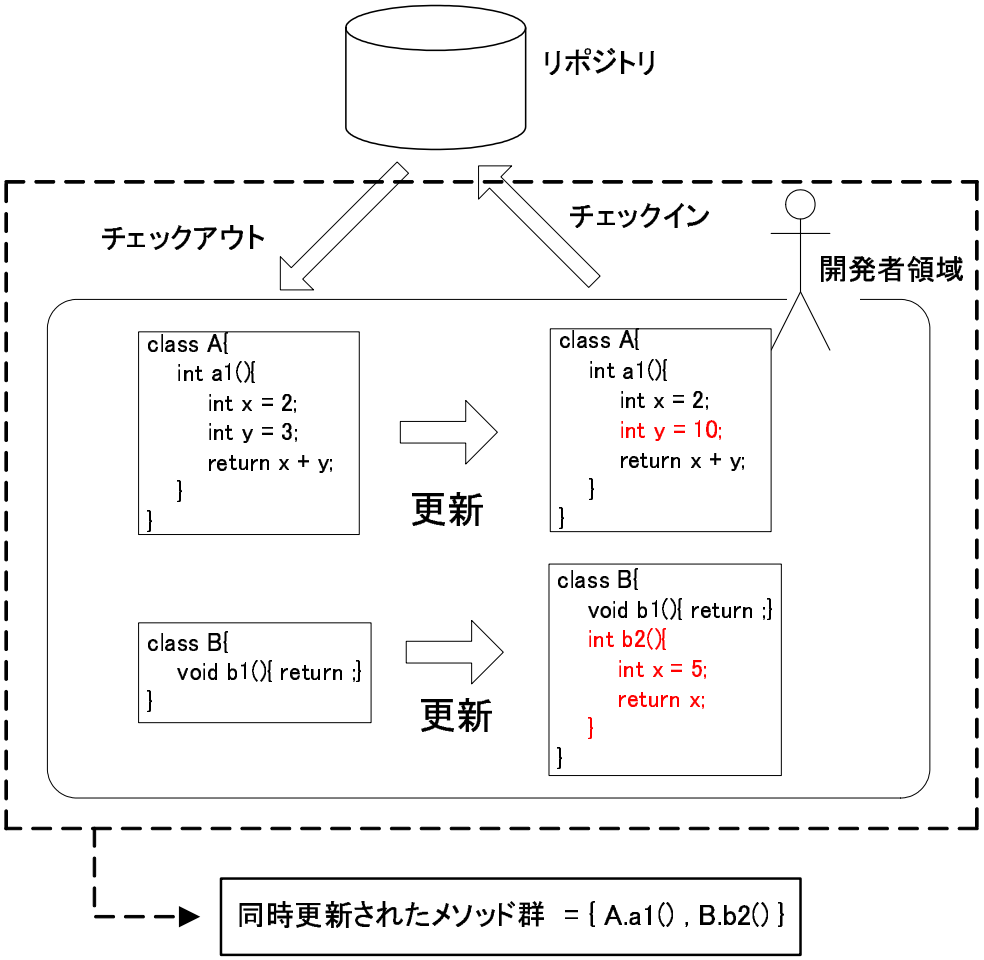

図 3: 同時更新されたメソッド群

4.2 外れ値の除去

本研究では，ソフトウェアの更新は機能単位で行なうことを仮定しているが，複数の機能の欠陥を修復した後にチェックインを行なった場合や，大規模なリファクタリングを行なう事により，同じ機能を実装しているクラス群でないクラス群が同時に更新されることも起こり得る．ここでは，そのような偶然同時に起きた更新を考慮しないようにするために，著しく多くのクラスが同時更新された場合に，その更新作業を以降の手順では考慮しないようにする．

同時更新されたクラスの数の分布はプロジェクトごとに異なるので，四分位点を用いた分析を行ない，外れ値を判定する．具体的な手順は以下の通りである．

ソフトウェアの全ての更新作業について，同時更新されたクラスの数を求め，その四分位点を値が小さい方から順に，Q_1, Q_2, Q_3 とする．そして，$Q_3 + (Q_3 - Q_1) \times 1.5$ を外れ値の閾値として定め，同時更新されたクラスの数が閾値より高い場合，その更新作業を除去する．

例えば，あるソフトウェアについての更新作業が 8 回行なわれており，各々の更新作業で同時更新されたクラスの数が以下のようになっていたとする．

$$8,\ 10,\ 11,\ 12,\ 12,\ 12,\ 14,\ 18$$

この時，四分位点 Q_1, Q_2, Q_3 は，

$$Q_1 = 10.75, Q_2 = 12, Q_3 = 12.5$$

となり，

$$Q_3 + (Q_3 - Q_1) \times 1.5 = 12.5 + (12.5 - 10.75) \times 1.5 = 15.125$$

であるので，外れ値の閾値は 15.125 となる．したがって，同時更新されたクラスの数が 18 である更新作業は外れ値となるため，これを除去する．

4.3 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 つのメソッドについて，その 2 つのメソッド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，$confidence$ というメトリクス [34] を用いる．

$confidence$ メトリクスは，ある 2 つのメソッド順序対 m_1, m_2 について，あるメソッドが更新されたとき，別のあるメソッド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．任意のメソッド m_1, m_2 について，

$$confidence_{m_1,m_2} = \frac{m_1 \text{と } m_2 \text{が同時に更新された回数}}{m_1 \text{が更新された回数}}$$

$confidence_{m_1,m_2} = 1$ となる時は，m_1 が更新された時は常に同時に m_2 も更新されることを意味する．

例えば，あるソフトウェア中のメソッド A.a1()，A.a2()，B.b()，C.c() が以下のように更新された場合を考える．

1. メソッド A.a1()，A.a2()，B.b()，C.c() が同時に 1 回更新される
2. メソッド A.a1()，A.a2()，B.b() が同時に 1 回更新される
3. メソッド A.a1()，B.b() が同時に 2 回更新される
4. メソッド A.a2()，C.c() が同時に 1 回更新される

このとき，各メソッド順序対についての $confidence$ 値は表 1 のようになる．表 1 では，$confidence_{\text{行の要素，列の要素}}$ の値をまとめている．

4.4 更新傾向グラフの作成

次に，$confidence$ 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．グラフの頂点はメソッドを表現する．頂点間の辺は向きを持っており，辺の始点が頂点 m_1，終点が頂点 m_2 であるとき，その辺は $confidence_{m_1,m_2}$ の値を持っている．

更新傾向グラフの例を図 4 に示す．図 4 は，表 1 について更新傾向グラフを作成したものである．

表 1: confidence の値

	A.a1()	A.a2()	B.b()	C.c()
A.a1()		0.5	1.0	0.25
A.a2()	0.67		0.67	0.67
B.b()	1.0	0.5		0.25
C.c()	0.5	1.0	0.5	

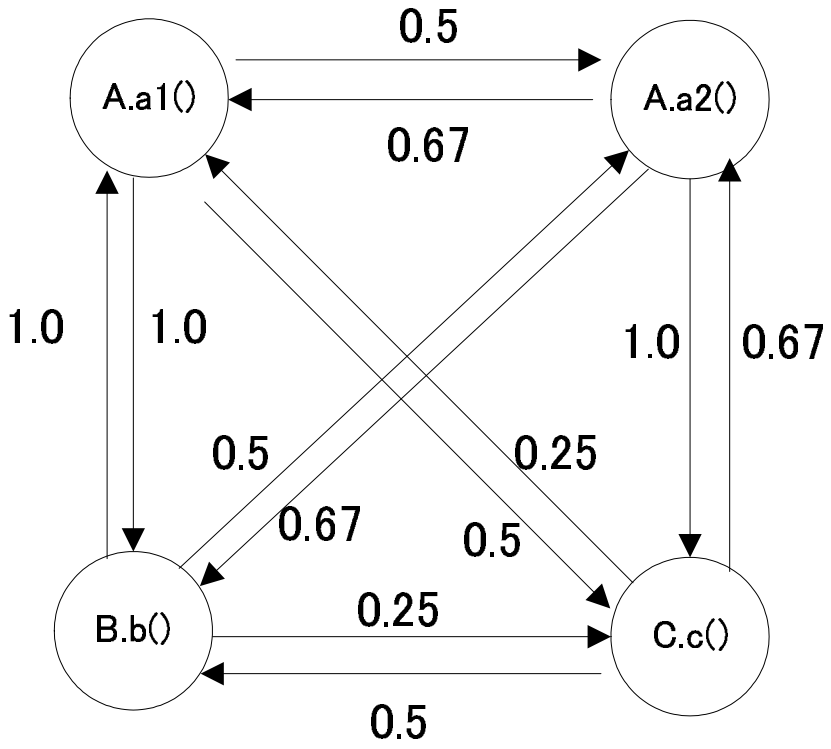

図 4: 更新傾向グラフの例

4.5 更新傾向グラフのクラスタリング

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，$confidence$ の高いメソッド同士をグループ化する．その結果を，クラスタ化更新傾向グラフと呼ぶ．クラスタ化更新傾向グラフの各頂点，辺は更新傾向グラフとほぼ同じものであるが，加えて各頂点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，階層的クラスタリング手法を用いて行なう．階層的クラスタリングは，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，という方法でクラスタリングを行なう手法である．階層的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を一つのクラスタと考え，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．

階層的クラスタリングは，クラスタ間の近さをどのように計算するかによっていくつか手法が存在する．

本研究では群平均法を用いる．群平均法では，2 つのクラスタ C_1と C_2 間の近さを以下のように定義する．

$$W_{C_1,\, C_2} = \frac{1}{n_1 \times n_2} \sum_{x_1 \in C_1} \sum_{x_2 \in C_2} (W_{x_1,\, x_2})$$

ここで，$W_{x_1,\, x_2}$ は頂点 x_1 と頂点 x_2 間の近さである．

本研究では，頂点 x_1 と頂点 x_2 の近さ $W_{x_1,\, x_2}$ を以下のように定義する．

$$W_{x_1,\, x_2} = \frac{confidence_{x_1,x_2} + confidence_{x_2,x_1}}{2}$$

すなわち，クラスタ間の近さはクラスタをまたぐメソッド間の $confidence$ 値の平均と考える．

図 4 の更新傾向グラフを群平均法を用いてクラスタリングを行なった例を図 5 で示す．なお，これはクラスタリングの閾値 W_{th} を 0.5 とした場合の結果である．

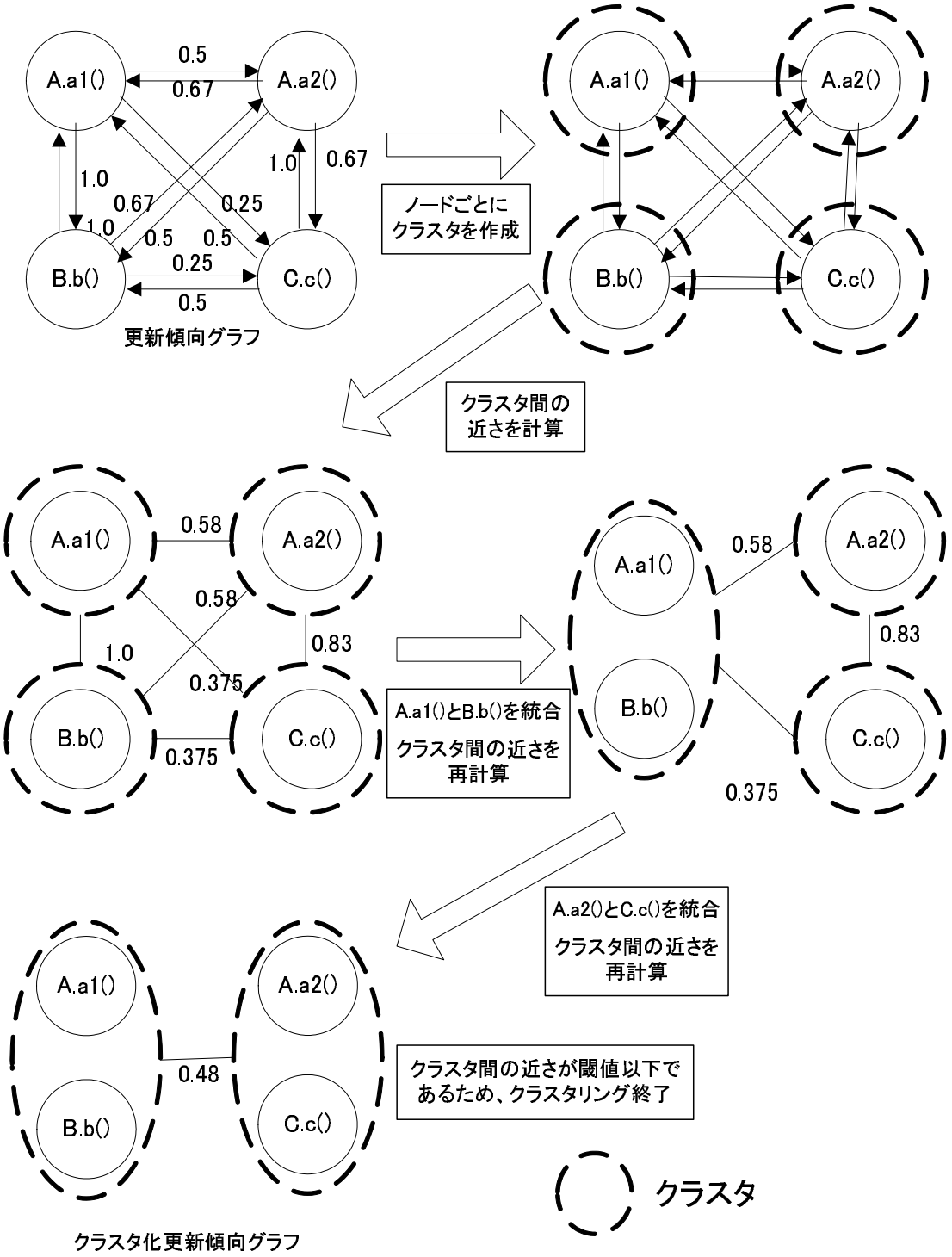

図 5: クラスタリングの経過

4.6 同時更新傾向の強いクラス群の抽出

クラスタリングの結果を開発者に提示する．開発者が適切にクラスタリングできていないと判断した場合は，クラスタリングの閾値 W_{th} を設定し直し再びクラスタリングを行なう．W_{th} については，ドキュメントなどから機能数がわかる場合は，それを参考に設定する．すなわち，機能数に対してクラスタの数が多すぎる場合は W_{th} を低くする．逆に機能数に対してクラスタの数が少なすぎる場合は W_{th} を高くする．

適切にクラスタリングができ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メソッド群が属しているクラス群を同時更新傾向の強いクラス群と考え，一つの機能を実装しているクラス群と考える．図 5 のようにクラスタリングが行なわれた場合，クラス群 {A , B} とクラス群 {A , C} がそれぞれ一つの機能を実装しているクラス群であると考える．

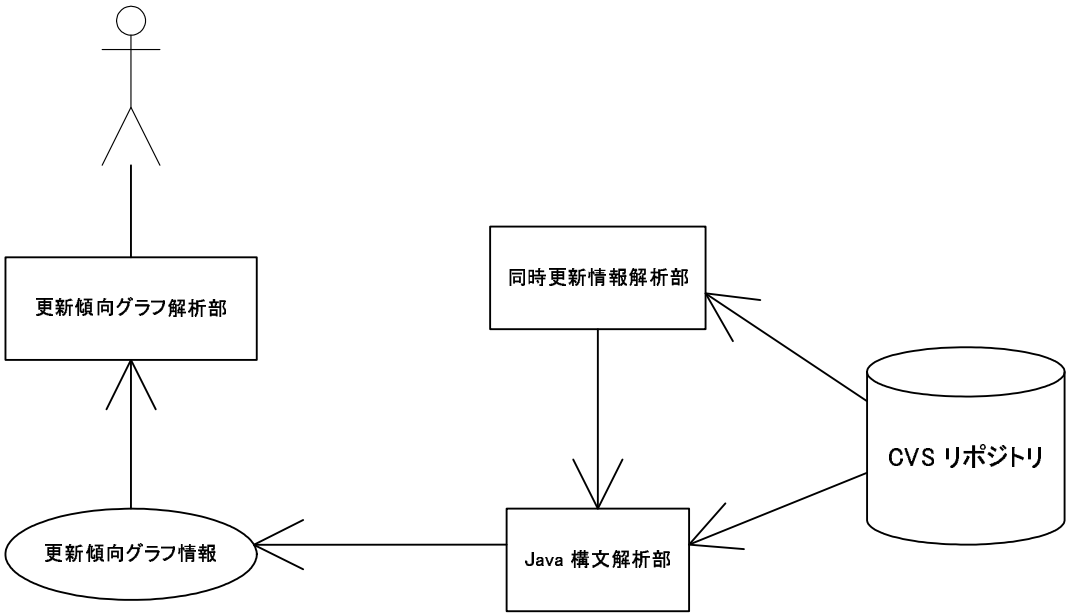

図 6: システム構成図

5 システムの実装

本節では，本システムの構成について述べる．

本システムでは，オブジェクト指向言語 Java を用いて記述されていて，版管理システム CVS によって管理されているソフトウェアを対象とする．

システムの構成図を図 6 で示す．本システムは，まず同時更新情報解析部が CVS システムのリポジトリにアクセスし，ソースコード中のどのファイルのどの行が同時に更新されたかを解析する．次に Java 構文解析部が，CVS のリポジトリから取得したソースコードの情報を基に，更新された行がどのメソッドに対応するものかを調べ，同時更新されたメソッドのリストを作成する．さらに更新傾向グラフ作成部が，同時更新されたメソッドのリストを基にメソッド間の $confidence$ メトリクス値を求め，更新傾向グラフの情報を出力する．更新傾向グラフ解析部は，更新傾向グラフの情報を基にグラフを描画し，ユーザの入力を基にグラフのクラスタリングを行ない，結果を表示する．

以下では，各部の動作を具体的に述べる．

5.1 同時更新情報解析部

CVS では，複数のファイルを一度にチェックインを行なう事ができ，本手法ではこの一度のチェックインで更新された箇所を同時に更新された箇所であると考える．しかし，CVS は一度のチェックインでどの箇所が更新されたかという情報は保持しておらず，更新の情報は

ファイル単位で保持している．同時更新情報解析部は，このファイル単位の更新の情報を用いて，同時に更新された箇所の特定を行なう．

CVS のリポジトリには，ソフトウェアの各ファイルに対して，そのファイルに対する更新の履歴を保持したファイルが存在する．例えば，リポジトリの中に「A,v」という名前のファイルがあれば，そのファイルに「A」という名前のファイルに対する更新の履歴が記されている．このような CVS のリポジトリ内の「ファイル名,v」というファイルは，RCS ファイルと呼ばれる．RCS ファイルには，対応するファイルに対して行なわれた各更新について，以下の情報が記されている．

- 更新者
- 更新日時
- 更新時のコメント
  なぜそのような更新を行なったか．
- 更新箇所
  ファイル中の何行目を更新したか．
- 更新内容
  新たに追加，更新された行の内容．

CVS では，複数のファイルを一度にチェックインする際，更新時のコメントとして同じコメントが用いられる．また，一度にチェックインを行なった場合，更新日時はほぼ同じになる．よって，同時更新情報解析部は対象とするソフトウェアの各ソースコードファイルに対応する全ての更新の情報を RCS ファイルから抽出し，以下の 3 つの条件を全て満たす更新を同時に行なわれた更新とし，同時に更新された箇所の特定を行なう．

- 更新者が同じである
- 更新時のコメントが同じである
- 更新日時の差が 3 分以内となっている

3 つ目の条件について補足する．CVS においては，一度のチェックインにより複数のファイルが更新された場合，更新が行われた時間として CVS リポジトリに保存される時刻は，実際にそのファイルへの変更がリポジトリに反映された時刻となっている．そのため，たとえ同じチェックインで更新が起きた場合でも更新が起きた時刻として保存される時刻がファイルごとに違う場合がある．よって，CVS リポジトリが保持している更新の時刻が違う箇

所でも，同じチェックインで更新された箇所となる可能性がある．そこで，更新が起きた時刻に若干の差がある場合でも，同時更新されたものであると考える．更新が行われた時刻の差の許容範囲を短くしすぎた場合，一度に行われたチェックインを複数のチェックインであるものと考えてしまうことがあり，逆に長くしすぎた場合 2 回以上のチェックインをまとめてしまうことがある．文献 [34] によれば，一度のチェックインの所要時間が 3 分を越えることはほぼ無いと考えられているため，更新が行われた時刻の差の許容範囲として 3 分を設定した．

5.2 JAVA 構文解析部

Java 構文解析部は，更新が行なわれた行とメソッドの対応を調べるために，Java のソースコードの構文解析を行ない，同時に更新されたメソッド群のリストを作成する．また，最新のリビジョンに含まれるメソッド，クラスの抽出を行なう．

Java 構文解析部の実装はコンパイラコンパイラ SableCC[27] を用いて行なった．SableCC は与えられた文法規則を基に，構文解析を行なうのに必要な Java のクラス群を自動的に生成するシステムである．SableCC の生成する構文解析システムは，入力ファイルが与えられた文法に従っているか否かを判定するだけのものであるが，システムが Java で実装されているため，そのクラスを拡張することにより目的に応じた構文解析器を作成することができる．

本システムでは，SableCC と既に用意されている Java1.4 の文法規則を用いて Java1.4 用の構文解析器を作成し，それを拡張することによりソフトウェアの各クラスが持っている全てのメソッドと，そのメソッドがソースコード中で記述されている位置 (行数) を特定する独自の構文解析器を作成した．この構文解析器を用いる事で，更新が行なわれた行とメソッドの対応を調べる．

ただし，構文解析を行なうのは最新のソースコードではなく，更新が行なわれた当時のリビジョンのソースコードである．そこで，Java 構文解析部は，CVS リポジトリから更新が行なわれた当時のリビジョンを取得し，そのソースコードを構文解析することで，更新が行なわれたメソッドを特定している．

5.3 更新傾向グラフ作成部

更新傾向グラフ作成部は，同時更新されたメソッド群のリストに基づき $confidence$ メトリクス値を計算し，更新傾向グラフの情報を求める．更新傾向グラフの情報には，グラフの頂点である最新リビジョンに含まれる全てのメソッドのリストと，グラフの辺である任意の最新リビジョンのメソッドの順序対の $confidence$ の値が含まれている．

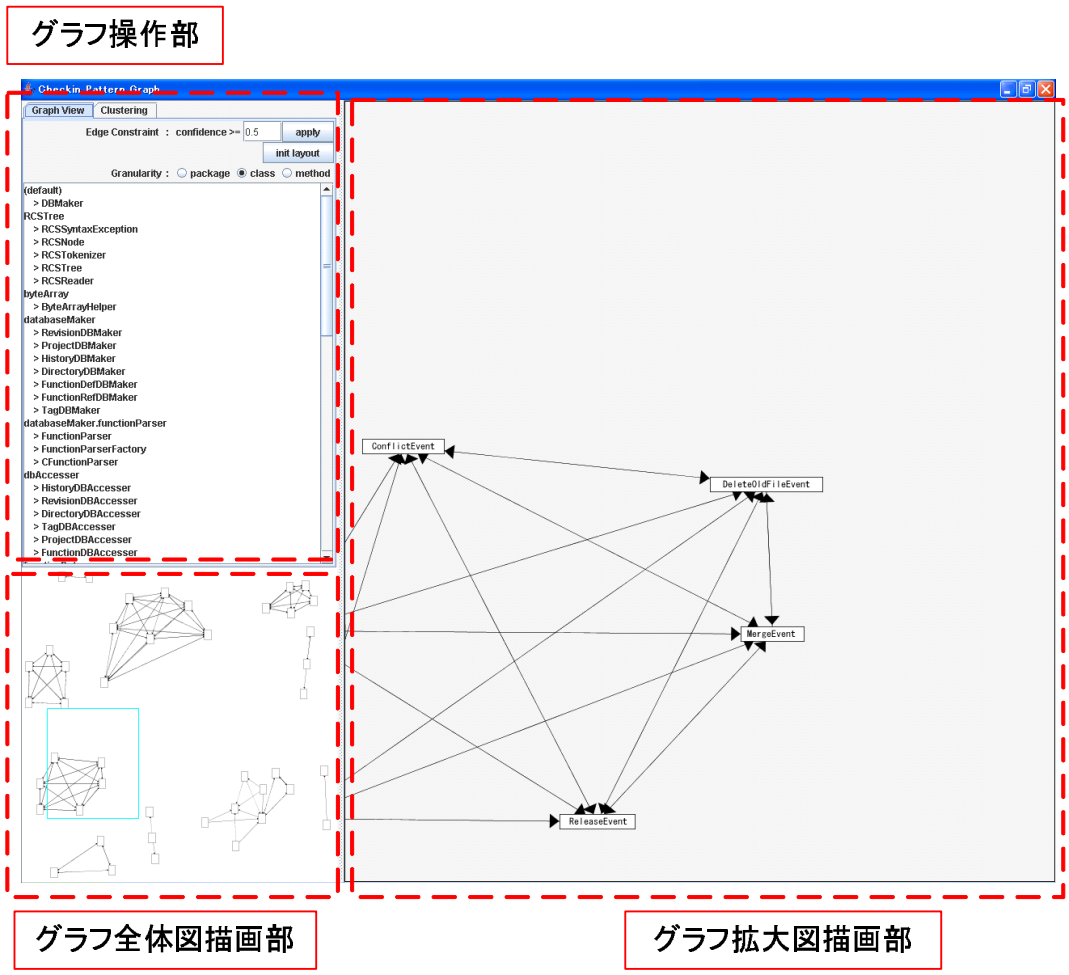

図 7: 更新傾向グラフ解析部の概観

5.4 更新傾向グラフ解析部

更新傾向グラフ解析部は，更新傾向グラフ作成部で作成した更新傾向グラフの情報をもとにグラフを描画し，クラスタリングを行なう．

グラフの描画にはグラフ描画ライブラリ JUNG(Java Universal Network/Graph Framework)[14]を用いた．JUNG は Java で記述されたライブラリであり，グラフの情報の解析や視覚化を行なうことができる．

更新傾向グラフ解析部の概観を図 7 に示す．更新傾向グラフ解析部は，グラフ操作部，グラフ全体図描画部，グラフ拡大図描画部と，グラフ操作部とタブで切替えられるクラスタリング部の計 4 つの部分から成り立っている．以下では，各部の動作について説明を行なう．

グラフ操作部

グラフ操作部では，グラフの頂点をリスト形式で表示する．また，更新傾向グラフの頂点や辺に対して以下の操作を行なう事ができる．

- 描画する辺の制限
  閾値を定め，その閾値以上の $confidence$ 値を持っている辺のみを描画を行なう．

- 頂点の配置の初期化
  グラフの頂点の配置をやり直し，グラフを再描画する．本ツールでは，グラフのレイアウトアルゴリズムとして，力学的モデルによるレイアウト手法 (Force-Directed Method) の一つである Fructerman，Reingold[16] の手法を用いている．このレイアウト手法は頂点の配置結果が初期配置に大きく依存するため，配置結果が適切でないと判断した場合は頂点の配置を初期化し，再度配置結果を求める．

- 更新傾向グラフの粒度の変更
  通常，更新傾向グラフの頂点はメソッドであるが，大規模なソフトウェアについてそのメソッドを全て描画すると，描画に非常に時間がかかったり，グラフの全体像を把握するのが難しくなる．このため，メソッド，クラス，パッケージをそれぞれ頂点とした場合の更新傾向グラフの描画を行なえるようにした．クラスを頂点した場合，クラスに属するメソッド間の $confidence$ 値の平均が定めた閾値以上である 2 クラス間に辺を描画する．パッケージを頂点とした場合も同様である．

なお，グラフ操作部における操作は，描画すべき頂点や辺を設定するためのものであり，クラスタリングの結果に影響を及ぼす事はない．

グラフ全体図描画部

グラフ全体図描画部は，更新傾向グラフの全体図を描画する．また，グラフ拡大図描画部に描画すべきグラフの範囲，拡大率の設定を行なう．ただし，グラフを描画する際に，頂点に接続されている全ての辺をグラフ操作部によって描画しないように設定していた場合，その頂点については描画は行わない．すなわち，孤立頂点は描画しない．

グラフ拡大図描画部

グラフ全体図描画部で設定した範囲，拡大率に従い，更新傾向グラフの拡大表示を行なう．

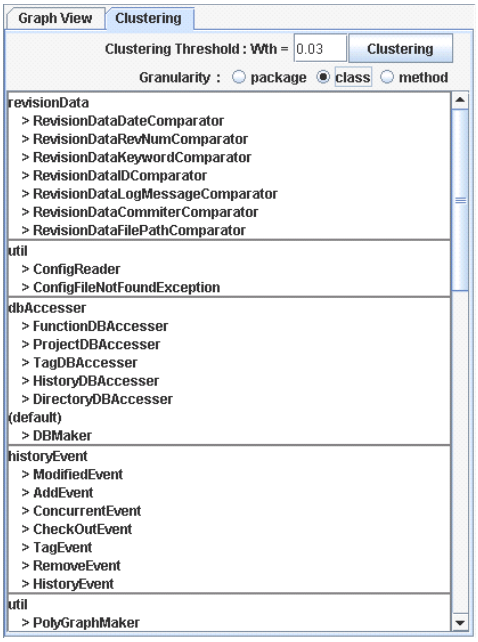

図 8: クラスタリング部の概観

クラスタリング部

クラスタリング部では，クラスタリングの閾値を定め，更新傾向グラフのクラスタリングを行ない，その結果の表示を行なう．クラスタリング部の概観を図 8 に示す．

クラスタリングの結果の表示は，パッケージ，クラス，メソッドの粒度から選択を行なう．クラスの粒度でクラスタリングの結果を表示する場合，同じクラスタに含まれるメソッド群が属しているクラスのリストが表示され，この結果が本手法の同時更新傾向の強いクラス群となる．

6 評価

本手法の妥当性を確かめるために，2 つのソフトウェアに対して検証実験を行った．実験の方法は，2 つの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機能実装クラス群となっているかどうかを検証するというものである．

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)[25] とオープンソースソフトウェアの図形エディタ jPicEdt[19] を用いた．以下，それぞれのソフトウェアについて実験を行った結果と考察を述べる．

6.1 実験結果

6.1.1 CREBASS

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 の規模は以下のとおりである．

- クラス数 : 60 クラス
- 総行数 : 7513 行
- 総更新回数 : 140 回

このソフトウェアに対して，本手法を適用した．クラスタリングの際に用いる閾値 W_{th} は 0.007 とし，10 個のクラス群が本手法で抽出された．

また，CREBASS のデータ解析部が実装している機能と，その機能を実装しているクラス群を開発者に尋ね，列挙してもらった．そして，各機能について，その機能を実装しているクラスを最も多く含むクラス群を，抽出した 10 個のクラス群の中から割り当てた．各機能について，その機能を実装しているクラスの数，割り当てたクラス群に含まれるクラスの数 (抽出クラス)，抽出クラスのうち実際に当該の機能を実装しているクラス群について表 2 にまとめている．

この結果より，適合率と再現率の計算を行なった．

$$\text{適合率} = \frac{\text{該当クラス}}{\text{抽出クラス}} = \frac{49}{65} = 0.75$$
$$\text{再現率} = \frac{\text{該当クラス}}{\text{実装クラス}} = \frac{49}{60} = 0.81$$

6.1.2 jPicEdt

jPicEdt は Java で記述された図形エディタである．長方形や円，テキストなどを組み合わせて図を編集し，その結果を文書整形システム Latex[21] で直接用いる事のできる形式で出力することができる．また，出力した図の情報を再び読み込み，編集する事もできる．jPicEdt の規模は以下のとおりである．

- クラス数 : 271 クラス
- 総行数 : 65696 行
- 総更新回数 : 247 回

本実験では，jPicEdt のファイルの入出力に関する機能について，その機能を実装しているクラス群をドキュメントやソースコードなどから判断し，CREBASS での実験と同様に本手法により抽出したクラス群と結果を比較した．結果を表 3 にまとめた．なお，クラスタリングの閾値 W_{th} は 0.05 とした．

また，CREBASS を対象とした実験と同様に，適合率と再現率の計算を行なった．

$$\text{適合率} = \frac{\text{該当クラス}}{\text{抽出クラス}} = \frac{18}{24} = 0.75$$
$$\text{再現率} = \frac{\text{該当クラス}}{\text{実装クラス}} = \frac{18}{82} = 0.22$$

表 2: CREBASS に本手法を用いた結果

機能名	実装クラス	抽出クラス	該当クラス
データベースを作成する	8	10	7
データベースからデータを取得する	6	8	5
C のファイルを構文解析する	3	10	3
RCS ファイルを解析する	5	5	5
設定ファイルを読み込む	2	2	2
折れ線グラフを作成する	5	5	5
リビジョン条件を比較する	7	7	7
関数データを管理する	5	8	5
ヒストリデータを管理する	14	7	7
プロジェクトデータを管理する	5	3	3
合計	60	65	49

6.2 実験結果の考察

今回の実験ではCREBASSを対象とした場合は概ね良い結果を得られたと考えるが，jPicEdtについては該当するクラス群を適切に抽出する事ができなかった．ここでは，両プロジェクトの違いと実験の結果との関連について述べる．

図9と図10はそれぞれCREBASSとjPicEdtの更新傾向グラフを本研究で作成したシステムを用いてノードの粒度をクラスとして描画したものである．$confidence$の閾値としては両者とも0.1としている．

図9と図10を比較すると，CREBASSの更新傾向グラフはいくつかの小さなグラフに分裂しているのに対し，jPicEdtの更新傾向グラフはノード間に非常に多くの辺が存在し，複雑に接続されているのがわかる．このことから，jPicEdtはCREBASSに比べ特定のクラス群を同時に更新する傾向が薄いということが言える．このため，jPicEdtではクラスタリングを適切に行なう事ができなかったものと考える．

また，表4はCREBASSとjPicEdtについて，全クラス数と最初の同時更新で更新されたクラスの数をまとめたものである．

表4の値を比較すると，jPicEdtの最初に更新されたクラスの数は非常に多く，jPicEdtの全クラスの約8割に達していることがわかる．本手法では，他の更新作業に比べ非常に多くのクラスが同時更新された更新作業があった場合，その更新作業を無視すると言う手順を用いているため，jPicEdtの最初の更新は無視をしていた．しかし，jPicEdtの全てのクラス

表3: JPicEdtに本手法を用いた結果

機能名	実装クラス	抽出クラス	該当クラス
tex形式の図を読み込む	22	4	4
pstricks形式の図を読み込む	29	0	0
eepic形式の図を読み込む	4	0	0
tex形式で図を出力する	9	5	4
pstricks形式で図を出力する	10	5	5
eepic形式で図を出力する	8	10	5
合計	82	24	18

表4: CREBASSとjPicEdtの全クラス数と最初の同時更新で更新されたクラスの数

	最初に更新されたクラスの数	全クラス数
CREBASS	4	60
jPicEdt	219	271

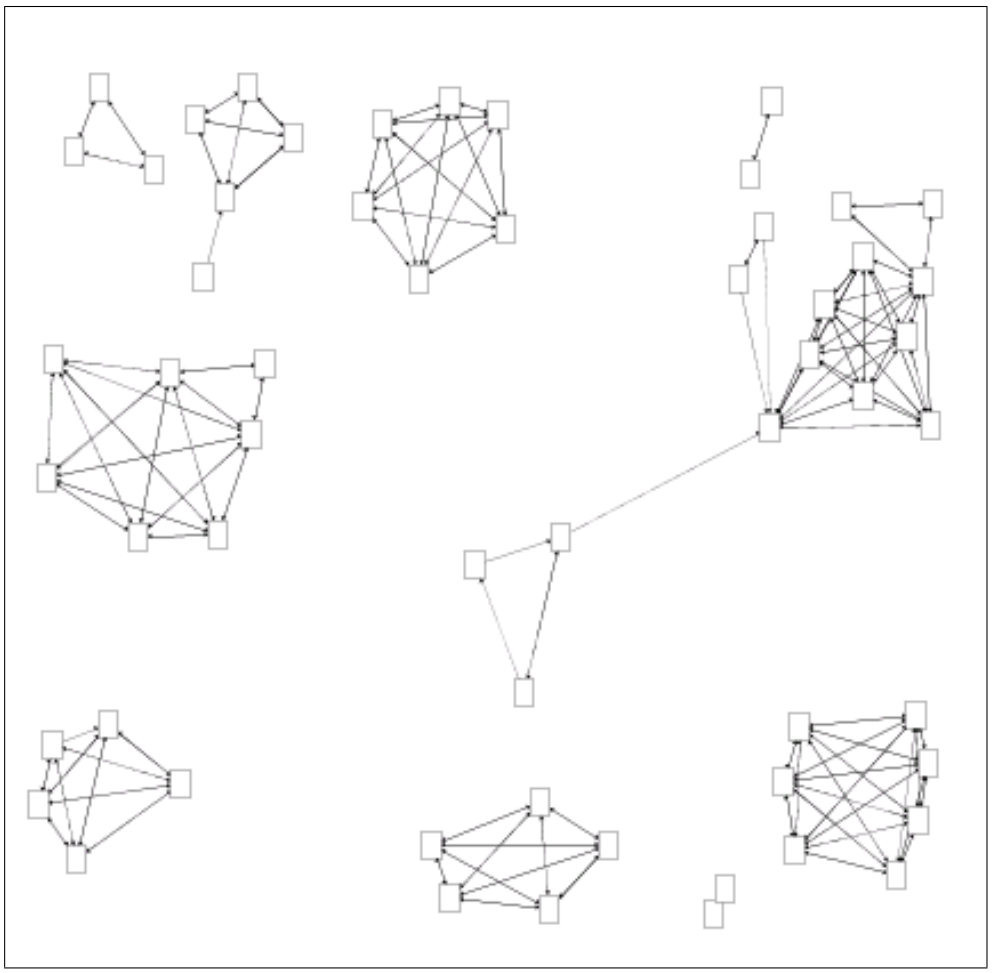

図 9: CREBASS の更新傾向グラフ (粒度はクラス)

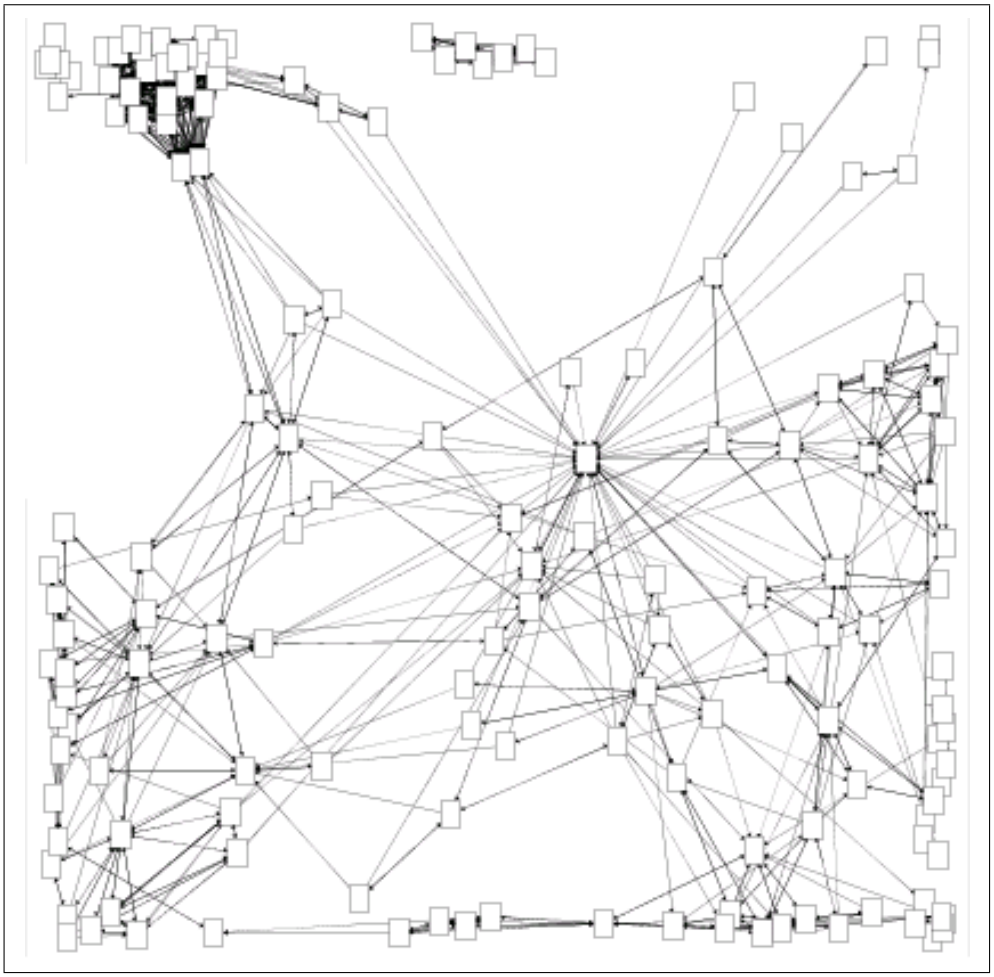

図 10: jPicEdt の更新傾向グラフ (粒度はクラス)

のうち 37 個のクラスは，この最初の一回の更新作業以外に一度も更新が行なわれておらず，「pstricks 形式の図を読み込む」「eepic 形式の図を読み込む」などを実装しているクラスもこれらのクラスに含まれていた．このこともクラスタリングを適切に行なう事ができなかった要因の一つであると考える．

jPicEdt の最初に更新されたクラスの数が CREBASS に比べて非常に多くなったのは，版管理システムの利用を開始する時期が両者で違っていたためであると考えられる．CREBASS はシステムの開発当初から版管理システムを用いて開発を行なっていたのに対し，jPicEdt はある程度システムに機能を実装させた後に版管理システムの利用を開始している．ある程度システムの開発を行なった後に版管理システムの利用を開始した場合，版管理システムの利用を開始する以前の更新の記録がリポジトリ内に残らず，その間に開発を行なった機能について分析が難しくなる．

これらのことから，オブジェクト指向ソフトウェアのクラスを機能別に分類することにおいて，本手法は有用であると考える．ただし，開発当初から版管理システムを利用していないプロジェクトに対して本手法を用いて分析を行なう場合，版管理システムの利用を開始する以前の履歴が取得できず，適切にクラスを分類することができなくなる可能性がある．

7 関連研究

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 [34]．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．また文献 [17] では，ソフトウェアの更新の履歴を用いて，ソースコード中のある部分に変更を加えた時，その変更が他のどの部分に影響を与えるかを予測する研究を行なっている．この研究では，同時に変更すべき箇所の集合を抽出し，その中の一つの要素が変更された時，他の要素も変更する事を開発者に促すということを提案している．識別子の名前や，メソッド，関数の呼び出し関係などと，実際の更新の履歴を用いる事で，同時に変更すべき箇所の集合を特定している．

ソフトウェアの履歴情報を現在のソフトウェア理解に役立てることを目的とした研究もある．文献 [7] では，ソフトウェア中のモジュールの依存関係を分析し，その依存関係が何故存在するのかを表現した依存グラフを作成する事で，ソフトウェアの構造理解を支援している．依存関係が何故存在するかを調べるために，ソフトウェアの更新の履歴を利用しており，依存関係が変化した際の更新のコメントを参照するという方法を採っている．モジュールという大きな単位を用いる事でソフトウェアの全体像を把握する際には有用であると考えるが，クラスなどのより詳細な単位でソフトウェアを理解をするには向かない．

文献 [4] ではソフトウェア中で頻繁に更新されるクラス群を特定し，可視化する手法の提案を行なっている．ソフトウェア中で，頻繁に更新されるクラスのみを描画したクラス図や，複数回同時に更新が行なわれたクラス群のみを描画したクラス図などを提案している．さらに，提案したクラス図と既存のクラス図とを比較し，更新が頻繁に行なわれるクラスの特徴や，複数回同時に更新されたクラス間にはどのような関係があるのかを調査した．更新が頻繁に行なわれるクラスはメソッドや属性の数が多いクラスであること，またデザインパターンを実装しているクラス群は同時に更新されることが多いことなどを明らかにした．

8 むすび

本研究では，オブジェクト指向ソフトウェアの機能と実装箇所の対応を特定することを目的に，ソフトウェアの開発履歴を利用してクラスを機能別に分類する手法の提案を行なった．具体的には，メソッドの同時更新の履歴を用いて，同時更新の傾向が強いと考えるメソッド群を特定し，その結果を基にクラスを分類する手法の提案を行なった．また，提案した手法を実装したツールの作成を行なった．

さらに，作成したツールを用いて適用実験を行なった．CREBASS を対象とした実験では，高い適合率，再現率でクラス群を分類する事ができ，ソフトウェア保守作業において，その作業を行なうために理解すべき箇所を特定でき，ソフトウェア理解のコストが軽減できる．また，jPicEdt を対象とした実験の結果から，本手法がソフトウェアの更新の形態や，版管理システムの導入の時期に依存する手法であるということもわかった．

最後に今後の課題としては，さらなる手法の改良があげられる．具体的には以下のものがあげられる．

- 静的手法と組み合わせる
  現在の手法ではソフトウェアの開発履歴のみからクラスを分類しているが，この手法は開発の履歴が正確に取得できる場合は有効であると考えるが，版管理システムに存在する更新の履歴が少ないクラスについては分類が難しい．そこで，静的なソースコードの解析結果と共に本手法を用いる事でさらに良い結果が望める．

- 更新の内容を考慮する
  現在の手法ではメソッドの更新の内容については考慮していない．しかし，例えばコメントのみの変更は無視するなど，更新の内容を考慮し，機能追加・修正の際の更新の特徴を発見することで，さらに適切にクラスを分類できると考える．

- ソースコード以外のプロダクトとの同時更新の履歴を考慮する
  ソースコードだけでなく，ドキュメントなどソースコード以外のプロダクトとの同時更新の履歴を解析する事で，クラスを分類するだけでなく，分類したクラスが実装している機能を特定するということ可能になるのではないかと考える．

謝辞

本研究の全過程を通して，常に適切な御指導および御助言を賜りました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授に心より深く感謝いたします.

本論文を作成するにあたり，適切な御指導および御助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 助教授に心から感謝いたします.

本研究において，様々な御協力を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 川口 真司 氏に深く感謝します.

本研究において，様々な御協力を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻 中山 崇 氏に深く感謝します.

最後に，その他様々な御指導，御助言等を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に深く感謝いたします.

参考文献

[1] Ulf AskLund, Lars Bendix, Henrik B Christensen, and Boris Magnusson. The Unified Extensional Versioning Model. *System Configuration Management:9th International Symposium,SCM-9*, pp. 100–122, 1999.

[2] David E. Bellagio and Tom J. Milligan. *Software Configuration Management Strategies And IBM Rational ClearCase:A Practical Introduction*. Addison-Wesley, 2005.

[3] Brian Berliner. CVS II:Parallelizing Software Development. In *USENIX Assosiation,editor,Proceedings of the Winter 1990 USENIX Conference*, pp. 341–352, 1990.

[4] James M. Bieman, Anneliese A. Andrews, and Helen J. Yang. Understanding change-proneness in oo software through visualization. In *Proceedings of the 11th International Workshop on Program Comprehension(IWPC'03)*, pp. 44–53, 2003.

[5] K. Chen and V. Rajlich. Case Study of Feature Location Using Dependence Graph. In *Proceedings of the 8th International Workshop on Program Comprehension(IWPC'00)*, pp. 241–247, 2000.

[6] Reidar Conradia and Berbard Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, Vol. 30, No. 2, pp. 232–280, June 1998.

[7] Ahmed E.Hassan and Richard C. Holt. Using development history sticky notes to understand software. In *Proceedings of the 12th IEEE International Workshop on Program Comprehension(IWPC'04)*, pp. 183–192, 2004.

[8] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Feature-driven program understanding using concept analysis of execution traces. In *Proceedings of the 9th Internatinal Workshop on Program Comprehension(IWPC'01)*, pp. 300–309, 2001.

[9] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating Features in Source Code. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, Vol. 29, No. 3, pp. 210–224, March 2003.

[10] Andrew David Eisenberg and Kris De Volder. Dynamic Feature Traces:Finding Features in Unfamiliar Code. In *Proceedings of the 21st International Conference on Software Maintenance(ICSM'05)*, pp. 337–346, 2005.

[11] K. Erdos and H. M. Sneed. Partial Comprehension of Complex Programs. In *Proceedings of the 6th International Workshop on Program Comprehension(IWPC'98)*, pp. 98–105, 1998.

[12] Karl Fogel. *Open Source Develpment with CVS*. The Coriolis Group, 2000.

[13] N. Ford and M. Woodroffe. *Introducing software engineering*. Prentice-Hall, 1994.

[14] Jung - Java Universal Network/Graph Framework.
http://jung.sourceforge.net.

[15] Peter Fröhlich and Woflgang Nejdl. WebRC Configuration Management for a Cooperation Tool. *System Configuration Management:7th International Symposium,SCM-7*, pp. 175–185, 1997.

[16] Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-directed Placement. *Software - Practice and Experience*, Vol. 21, No. 11, pp. 1129–1164, 1991.

[17] Ahmed E. Hassan and Richard C. Holt. Predicting change propagation in software system. In *Proceeding of the 20th International Conference on Software Maintenance(ICSM'04)*, pp. 284–293, 2004.

[18] IEEE std 1219:standard for software maintenance, 1997.

[19] jPicEdt for Latex.
http://jpicedt.sourceforge.net.

[20] 鯉江英隆, 西本卓也, 馬場肇. バージョン管理システム (CVS) の導入と活用. SOFT BANK, December 2000.

[21] LaTeX - A document preparation system.
http://www.latex-project.org.

[22] Pigoski T. M. *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994.

[23] PVCS Merant, Inc.
http://www.merant.com/pvcs/.

[24] Micrsoft Visual SourceSafe Microsoft Corporation.
http://msdn.microsoft.com/ssafe/.

[25] 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.

[26] 大月美佳. 入門 CVS Concurrent Versions System. SHUWA SYSTEM CO.,Ltd, 2001.

[27] SableCC Java parser generator.
http://sablecc.org.

[28] The FreeBSD Project.
http://www.freebsd.org.

[29] The OpenBSD Project.
http://www.openbsd.org.

[30] Walter F. Tichy. RCS - A System for Version Control. *SOFTWARE - PRACTICE AND EXPERIENCE*, Vol. 15, pp. 637–654, 1985.

[31] Version Control with Subversion.
http://svnbook.red-bean.com/.

[32] W. Eric Wong, Joseph R. Horgan, Swapna S. Gokhale, and Kishor S. Trivedi. Locating program features using execution slices. *1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, pp. 194–203, 1999.

[33] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. SNIAFL:Towards a Static Non-Interactive Approach to Feature Location. In *Proceedings of the 26th International Conference on Software Engineering(ICSE'04)*, pp. 293–303, 2004.

[34] 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'03)*, pp. 95–105, 2003.