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

信学技報
IEICE Technical Report
SS2006-80 (2007-02)

ソフトウェア変更間の関連抽出のための差分集合構築手法

今枝 誉明† 市井 誠† 早瀬 康裕† 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番3号
E-mail: †{t-imaeda,m-itii,y-hayase,matusita,inoue}@ist.osaka-u.ac.jp

あらまし 現在広く用いられている版管理システムのリポジトリ閲覧システムでは個々の変更を表示するのみであるため，開発者が過去に行われた変更を参照する際に，その変更に関連する変更を人手で探さなければならなかった．そこで本研究では，変更間の関連を抽出することを目的として，変更の内容に基いて，変更の集合を構築する手法を提案する．また，提案手法を実現するシステムを試作し，実際のソフトウェアに対して適用した結果，本手法の有効性と問題点を確認することが出来た．

キーワード ソフトウェア理解，クラスタリング，版管理システム

An Approach to Making Change Sets for Extracting Relations among Software Changes

Takaaki IMAEDA†, Makoto ICHII†, Yasuhiro HAYASE†, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama-cho, Toyonaka-shi, Osaka, 560-8351 Japan
E-mail: †{t-imaeda,m-itii,y-hayase,matusita,inoue}@ist.osaka-u.ac.jp

Abstract At present, repository viewers for widely used version control systems only show the single changes, therefore, when developers consult the changes performed in the past, they must manually search for relationships between them. In this paper we propose an approach for extracting relationships between changes that groups changes into sets of related changes based on their content. The approach has been implemented as a prototype system, and used to analyze version control repository of an actual software system. The results consented to prove the soundness of our approach and also to find potential problems.

Key words Software Comprehension, Clustering, Version Control System

1. はじめに

近年のソフトウェアの開発規模の増大に伴い，その開発形態は多人数化，分散化している．また，ソフトウェアの大規模化や複雑化，老朽化に伴い，ソフトウェア保守はますます困難になり，長期間に渡って使用されるソフトウェアでは，保守コストがソフトウェア全体のコストの大部分を占めると言われている[1].

複雑化しているソフトウェアを効率よく管理するため，近年のソフトウェア開発では一般的に，ソフトウェアを構成するファイル(ソースコード，ドキュメント等)に対する変更を記録するシステムである，版管理システムを用いる[2].

また，版管理システムに蓄積された過去の変更を理解することで，開発および保守作業を効率的に行うことが出来る[3].

しかし，蓄積された膨大な情報の中から，開発者が必要とする情報を的確に取得することは容易ではない．既存のシステムにより，過去の変更を個別に閲覧することは可能である．しかし，変更は互いに関連を持つことがあるため，開発者は一つの変更だけでなく，関連する変更を同時に参照しなければならない．例えば，ある関数定義の変更と対応する関数呼び出しの変更や，欠陥を埋め込んだ変更とその欠陥を修正した変更が挙げられる．しかし，既存の版管理システムでは，関連する変更を人手で探さなければならず，開発者の負担となっている．

そこで本研究では，版管理システムに蓄積されたファイルに対する変更を関連付ける手法を提案する．提案する手法では，変更箇所に含まれる単語に着目し，類似した単語を持つ変更をグループ化することで，関連した変更を一度に閲覧することが出来るようにする．

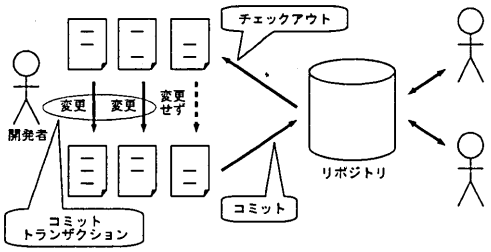

図 1　CVS を用いた開発プロセスの例

また，提案手法を実現するシステムを試作して実際のソフトウェアに対して適用し，本手法により関連のある変更の集合が抽出出来るかどうかを確認する．

以降，2. 章では，ソフトウェア開発で用いられる版管理システムについて説明し，その問題点について説明する．3. 章では，提案する手法について説明し，4. 章で本手法の適用事例について述べる．最後に 5. 章でまとめと今後の課題について述べる．

2. ソフトウェア開発環境

本章では，版管理システムについて説明した後，変更履歴を利用する際の問題点を指摘する．

2.1 版管理システム

版管理システムとは，ファイルに対して施された追加・削除・修正等の変更を記録するシステムである．更に，開発者間の変更の競合を検知して，並行開発をサポートするシステムも存在する．

現在広く用いられている版管理システムとして，CVS (Concurrent Versions System) [4] が挙げられる．図 1 に，CVS を用いた開発プロセスの例を示す．CVS を用いた開発では，リポジトリと呼ばれるデータベースを中心に作業を行う．リポジトリには開発対象のソフトウェアを構成する全てのファイルの，過去の任意の時点での状態が保存されている．ファイルはパス名で識別される．また，各ファイルのある時点での状態をリビジョンと呼び，リビジョンは時系列に従って連続した番号で識別される．開発者は以下の作業を繰り返し，開発・保守を進めていく．

（1）　リポジトリから必要なファイルを手元にコピーする
（2）　コピーされたファイルの一部または全てを変更する
（3）　変更したファイルをリポジトリに登録する (コミット)

コミット作業により，リポジトリ中にファイルの新たなリビジョンが生成される．

CVS の特徴として，以下のものが挙げられる．

- ファイル単位で変更が記録されること
- ファイルに対する変更は直前のリビジョンとの行単位での差分の形式で記録されること

  具体的には，削除または追加された行の位置と内容の組の集合が記録されている．修正された行は，削除と追加を組み合わせて表現される．
- 変更した複数のファイルを一度のコミット作業でリポジトリに登録出来ること

  この時の一群のファイルに対する変更をまとめてコミットトランザクション (以下単にトランザクション) と呼ぶ．ただし，CVS ではトランザクションを構成するファイルやリビジョンの情報は記録されていない．
- コミット時にコミットログと呼ばれる，変更理由等を記述した文章を残すことが出来ること

2.2 版管理システムに蓄積された変更履歴の利用

開発者は，開発・保守工程において版管理システムに記録された変更履歴を参考にすることで効率的に作業を行うことが出来る．例えば，機能を追加する時や欠陥を修正する時に，過去の類似した変更を真似ることで簡単に作業を終えることが出来る．

しかし，変更を個別に調べるだけでは，関連する変更を見落としてしまうことで誤解が生じる可能性があるため，全ての関連する変更を知らなければならない．例えば，ある変更に伴い他の変更が行われていた場合や，ある変更により欠陥を混入させ，後に行われた変更でその欠陥の修正を行っていた場合に，前者の変更のみを真似ると欠陥を埋め込んでしまう．しかし，蓄積された膨大な変更の中から関連する変更を人手で探すのは困難であり，開発者の負担となっている．

2.3 関連研究

変更履歴の閲覧を支援するシステムとして ViewVC [5] や，CREBASS [6] がある．ViewVC は版管理システムのリポジトリ内に保存されている変更履歴情報を Web ブラウザ経由で視覚的に閲覧できるシステムであり，ファイルの各リビジョンの内容の表示する機能，ファイルの任意のリビジョン間の差分を色付きで表示する機能，そして正規表現を用いたリビジョン検索機能等を備える．また CREBASS は，上記の ViewVC の機能に加え，リビジョン関係を考慮した関数クロスリファレンス機能を備える．

また，変更履歴から，ファイルや変数・関数等のプログラム要素間の関連を抽出するための研究として Zimmermann らの研究 [7] や，Gerardo らの研究 [8] が挙げられる．[7] では，変更履歴から，プログラム要素間の相関ルール (association rule) の抽出を行い，ある要素が変更された際に，抽出したルールを基に変更すべき要素を提示する．また [8] では，版管理システム中のコミットログと，欠陥追跡システム中の変更要求記述との文書上の類似度 (textual similarity) を計算し，変更要求とファイルの関連をデータベースに格納する．そして新たな変更要求の記述から，過去の類似した変更要求を抽出し，変更される可能性のあるファイルを提示する．

3. 提案手法

2.2 節で述べた問題点を解決するために，開発者に対して，ある一つの変更だけでなく，関連する変更を自動的にまとめて提示することを考える．

我々はここで，コミットログ，および変更時に記録される差分の中に出現する単語に着目した．なぜなら，コミットログには変更理由が記述されており，差分は何が削除され，何が追加

されたかという変更の内容であるため，これらに出現する単語は変更を特徴付けていると考えられるためである．

そこで，出現する単語を基に変更をクラスタリングすることで，関連する変更の集合が得られることが期待できる．本手法では，単語に基づいた関連抽出のために，ベクトル空間モデルに基いた手法である LSA (Latent Semantic Analysis) [9] を用いる．

以降の節では，本手法で用いる LSA の説明，手法の説明に当たり必要な用語の定義，および手法の手順の説明を行う．

3.1 潜在的意味解析手法 LSA

LSA は文書群の中から，単語間の潜在的な関連を考慮して，単語間もしくは文書間の類似度を算出する手法である．

LSA ではまず，単語文書行列と呼ばれる，文書内に出現する単語の出現回数を要素とする行列を作成する．続いて単語文書行列に対して特異値分解と呼ばれる統計的な計算と，次元の削減を行うことで，単語間の潜在的な関連を明らかにする．そして，次元を削減した行列から文書ベクトルと呼ばれるベクトルを取り出し，文書間の類似度を計算する．ベクトル空間モデルにおいて，文書間の類似度算出には一般的に，ベクトル間の cosine 尺度が用いられる．

3.2 用語の定義

2.1 節で述べた，ファイルに対する変更を表す差分の内，削除行の内容と追加行の内容の集合を**変更差分**と呼ぶ．

また，変更差分に以下の情報を加えたものを，**変更情報**と呼ぶ．

- 変更したファイルのパス名
- 変更後のリビジョン番号
- コミットした開発者
- コミットされた日時
- コミットログ

そして，トランザクションを表す変更情報の集合を**トランザクション情報**と呼ぶ．

3.3 提案手法の手順

本手法では，CVS リポジトリに保存された変更履歴をグループ化する．手法は図 2 で示すように，以下の 3 つのステップからなる．

（1） リポジトリから変更情報を取り出す

（2） 変更情報の集合からトランザクション情報の集合を構築する

（3） LSA を用いてトランザクション情報間の類似度を求め，トランザクション情報をクラスタリングする

以降で，手法の各ステップを説明する．

3.3.1 変更情報の取り出し

手法の最初のステップでは，リポジトリに記録された全てのファイルに対する全ての変更から，変更情報を作成する．

3.3.2 トランザクション情報の集合の構築

3.3.1 節のステップで取り出してきた変更情報の集合を分類し，トランザクション情報の集合を構築する．本手法では，変更情報からトランザクションを特定するために，Zimmermann らの提案した定義 [10] を用いた．この定義では，変更の集合が

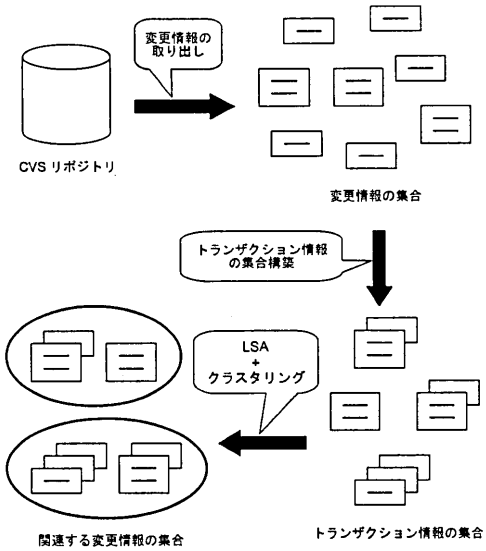

図 2 手法の流れ

以下の 3 つの条件を全て満たすとき，その変更の集合をトランザクションとしている．

- コミットした開発者が同じである
- コミットログが同じである
- コミットされた日時の差が 200 秒以内である

3.3.3 LSA を用いた類似度計算とクラスタリング

このステップでは，3.1 節で説明した LSA の計算結果を用いて，トランザクションのクラスタリングを行う．以下にその手順を述べる．

（1） 単語文書行列作成

本手法では，単語文書行列を作成する際に，1 つのトランザクション情報を 1 つの文書とみなして行列を作成する．この行列は，トランザクション数を m，単語数を n とした時，$m \times n$ 行列となる．

トランザクション情報からの単語抽出は，トランザクション情報に含まれる全ての変更情報に共通するコミットログと，個々の変更差分を対象に行う．まず全てのトランザクション情報から単語を抽出し，出現数を数え上げる．トランザクション情報 T 中の単語 w の出現数は，T に含まれる全ての変更差分中の単語 w の出現数を足し合わせたものに，コミットログ中に w が出現した回数を加えたものである．

変更差分から単語とその出現数を抽出するために，本手法では以下で説明する WCA 法，WCD 法の 2 種類の方法を用いる．ただし説明中の，$C(D,w)$ は変更差分 D 中に単語 w が出現した回数，D_{add} は D の内の追加行の集合，D_{del} は D の内の削除行の集合を表すものとする．つまり，$D = D_{\mathrm{add}} \cup D_{\mathrm{del}}$ である．

（a） WCA 法: 変更差分中に出現する単語を全て数える方法

この方法は，変更差分中に出現する単語が全て，変更の特徴付けに寄与するとする算出方法である．変更差分中の単語の出現数は次式により求める．

$$D\text{ 中の }w\text{ の出現数} = C(D, w)$$

例えば，ある変更により削除された行が

```
foo bar baz
```

追加された行が

```
foo bar qux
```

であった場合，単語 foo，bar はそれぞれ 2 回，単語 baz，qux はそれぞれ 1 回出現したと数える．

（b） WCD 法: 変更差分中の削除行・追加行における各単語の出現数の差の絶対値を単語の出現数とする方法

この方法は，変更差分中の削除行または追加行のどちらか一方にだけ多く表れる単語が，変更の特徴付けに寄与するという考えに基く算出方法である．変更差分中の単語の出現数は次式により求める．

$$D\text{ 中の }w\text{ の出現数} = |C(D_{\mathrm{add}}, w) - C(D_{\mathrm{del}}, w)|$$

(a) と同じ例の場合，単語 foo，bar は削除・追加両方に 1 回ずつ出現しているので 0 回，単語 baz，qux は 1 回出現したと数える．

（2） 不要語の除去

抽出した単語のうち，少数のトランザクション情報にのみ出現する単語，および多数のトランザクション情報に出現する単語を取り除く．少数の文書にのみ出現する単語は文書間を関連付けるのに役に立たず，多数の文書に出現する単語は文書を特徴付けるのに不適当とされるためである [11]．本手法では，単一トランザクション情報にのみ出現する単語，および過半数のトランザクション情報に出現する単語を除去するものとした．

（3） LSA の適用

トランザクション情報間の類似度を算出するために，不要語の除去を行った単語文書行列に対して 3.1 節で述べた LSA を適用する．

（4） トランザクション情報のクラスタリング

LSA の結果得られた行列から，各トランザクション情報 T_i を表現する文書ベクトル v_i を抽出し，文書ベクトル間の類似度に基いてトランザクション情報をクラスタリングする．ここでは階層的クラスタリング手法 [12] を採用した．階層的クラスタリング手法とは，各トランザクション情報が独立したクラスタである状態を開始状態とし，最も類似度の高いクラスタから順次結合していく手法である．

文書ベクトル間の類似度算出には，3.1 節でも述べた通り，ベクトル間の cosine 尺度を用いる．また，クラスタ間の類似度算出には，最も類似度の低い文書間の類似度をクラスタ間の類似度とする完全リンク法 [12] を採用する．クラスタ C, D 間の類似度 $\mathrm{sim}(C, D)$ は，以下の式により求められる．

$$\mathrm{sim}(C, D) = \min_{v_C \in C, v_D \in D} \cos(v_C, v_D)$$

クラスタリングのアルゴリズムは以下の通りである．ただし，s_{th} は閾値を表す．

N : 総トランザクション数
$T_1, T_2, \ldots, T_N$: トランザクション情報
$S \leftarrow \{\{T_i\} \mid 1 \leqq i \leqq N\}$
while $|S| \neq 1$ **do**
  以下の条件を満たすクラスタの組 $(a, b)(a, b \in S)$ を求める
    $a \neq b \ \wedge\ \forall c, d \in S,\ c \neq d \ \wedge\ \mathrm{sim}(a, b) \geqq \mathrm{sim}(c, d)$
  if $\mathrm{sim}(a, b) \leqq s_{\mathrm{th}}$ **then**
    ループ終了
  end if
  $S \leftarrow S \cup \{a \cup b\} - \{a, b\}$
end while
S がトランザクション情報のクラスタリング結果となる

4. 適用事例

本章では，前述の提案手法に基いて試作した差分集合構築システムを，実際のソフトウェアに適用した事例について述べる．

4.1 適用対象

適用対象としては，我々の研究室で開発されたソフトウェア部品検索システム SPARS-J [13] を用いた．SPARS-J の概要は表 1 の通りである．

4.2 適用方針

WCA 法と WCD 法の両方で単語抽出を行い，それぞれでクラスタリングを行う．クラスタリングの際の閾値 sim_{th} は共に 0.8 とした．

また，本稿では複数トランザクション情報を含むクラスタに対してのみ評価を行う．3.3.3 節の手法によりトランザクション情報のクラスタリングを行うが，抽出する単語や閾値等様々な要因により，一つのクラスタが単一のトランザクション情報のみから構成されることがある．そのような単一のトランザクション情報から構成されるクラスタは，トランザクション情報をクラスタリングする本手法の評価対象として適切でないと考えられるためである．

クラスタの評価方針としては，「有用」，「有用でない」のどちらかに分類することとする．本稿では，以下のいずれかの条件を満たすようなようなトランザクション情報を含むクラスタを，含まれる変更間に関連があり，クラスタとして有用であると判断した．

- 欠陥を混入した変更を含むトランザクション情報と，その欠陥を修正した変更を含むトランザクション情報
- 関数仕様の変更を含むトランザクション情報と，それに伴う呼び出しの変更を含むトランザクション情報
- 一つの機能の実装を目的とした変更を含むトランザク

表 1 SPARS-J の概要

開発期間	2003/03/03 - 2006/01/08
総リビジョン数	2795
総トランザクション数	834
ソースファイル数	345
最新リビジョンでの総行数	95479
開発言語	C/C++

ション情報の集合

- 利用するライブラリの置換のための変更を含むトランザクション情報の集合

また，変更行数が多いトランザクション情報からなるクラスタは，関連があったとしてもその理解が困難であるため，本稿では有用でないと判断した．

次節では，以上の適用方針に基づいた適用結果を示す．

4.3 適用結果

結果は表 2 の通りである．以下で，それぞれの内容について詳しく述べていく．

4.3.1 WCA 法でのクラスタリング結果

WCA 法でのクラスタリング結果では，変更差分中に出てくる全単語を数え上げているので，変更内容の似ている，すなわち同じ単語を多く持つトランザクション情報がクラスタを形成しているパターンが多く見つかった．その中でも特に，機能追加を行った変更とその直後の欠陥修正を行った変更からなるクラスタや，ある変更とその箇所に対するリファクタリングを行った変更からなるクラスタが，生成されたものの大半を占めていた．

具体例としては，図 3 のようなクラスタが挙げられる．追加された行は行頭に '>' が付いており，削除された行は '<' が付いている．図 3 の例ではクラスタは 3 つのトランザクション情報からなっている．このクラスタでは，データベース操作時に必要な比較関数に関連のある変更集合が抽出出来た．1 つ目のトランザクションでは，ファイル SPARS/src/DB/db_common.c で比較関数の定義を行っており，その中で変数 ai，bi が u_int8_t * 型と宣言されている．2 つ目のトランザクションでその型に合わせるために，変数に代入している箇所でキャストを行う修正がなされたが，そのキャストも間違っていたため，3 つ目のトランザクションで同一箇所に更なる修正が施されている．

4.3.2 WCD 法でのクラスタリング結果

WCD 法のクラスタリング結果では，変更差分中に出てきた単語が，削除と追加で相殺される形になるので，変更内容としてはさほど似ていなくても，変更の特徴をよく表していると思われる単語を共通して持つクラスタが見られた．

具体例を図 4 に示す．図 4 の例では，最初のトランザクションでデータベースの項目にファイルの mtime (更新時間) を追加する変更が行われ，続くトランザクションでデータベースへアクセスする関数の引数に対し，その項目が追加されている．

4.3.3 有用でないクラスタ

以上は明確な関連があり，有用と思われるクラスタの例であるが，そうでないクラスタも見られた．有用でなかった例として，コマンドラインオプションを追加する変更からなるクラスタや，CGI プログラムに対する変更からなるクラスタ等が挙げられる．前者では，複数の個別のコマンドラインツールに対して，別々にオプションを追加していたが，それぞれで usage，getopt (C 言語のコマンドラインオプション解析ライブラリ) 等の単語が共通して出現していた．また後者では，HTML タグ文字列が共通して多く出現していた．これらのクラスタは，機能的な関連がなく，同時に参照する必要がないと判断し，有用でないクラスタに分類した．

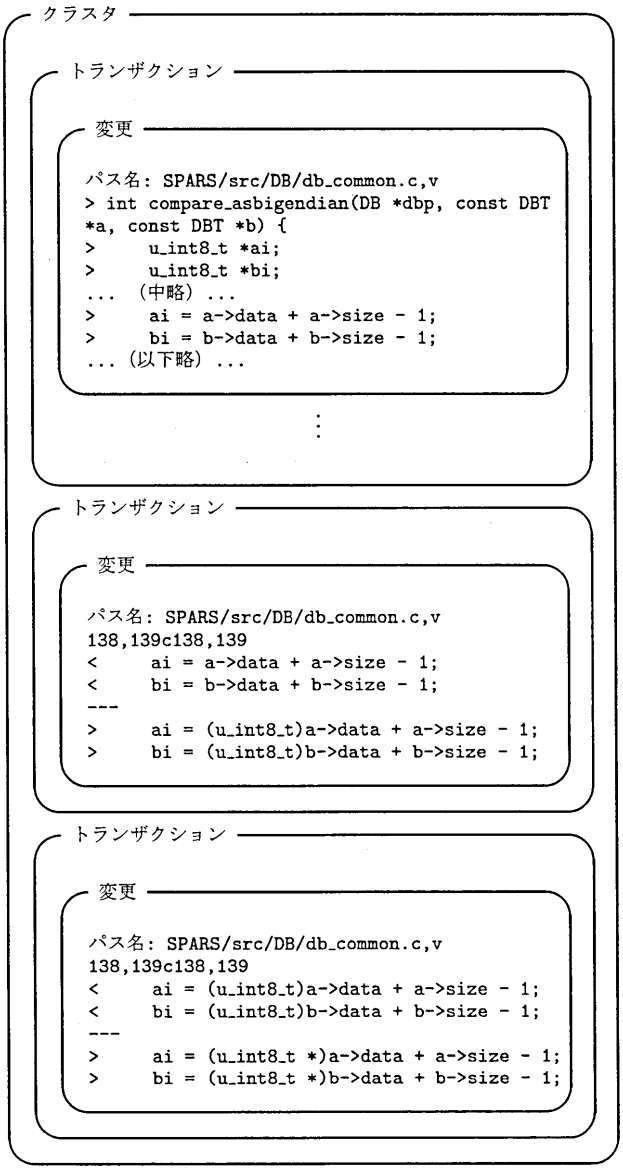

図 3 WCA 法での有用なクラスタの例 (抜粋)

表 2 クラスタリング結果

		WCA 法	WCD 法
クラスタ数		644	666
複数トランザクション情報を含むクラスタ数		116	113
	有用なクラスタ数	47 (40.5%)	52 (46.0%)
	有用でないクラスタ数	69 (59.5%)	61 (54.0%)

4.4 考　　察

本稿の適用事例では，有用と思われるクラスタが生成出来た一方，そうでないクラスタも多数見られた．これは，出現する単語を基にする手法の性質によるもので，類似した単語が出現すればクラスタとしてまとめられるのが原因であると考えられ

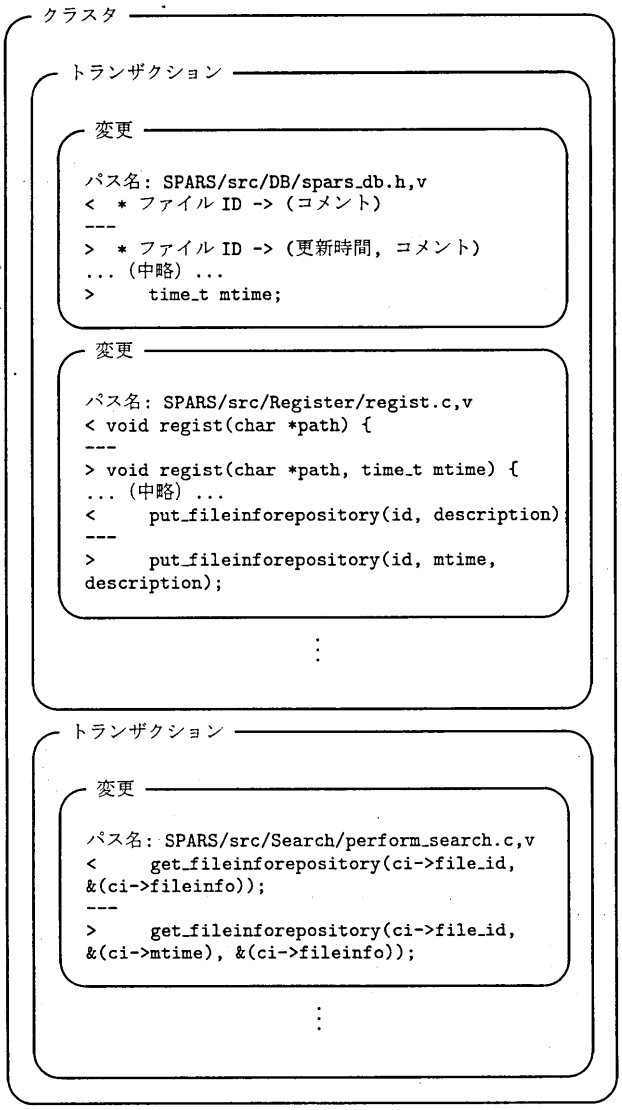

図 4 WCD 法での有用なクラスタの例 (抜粋)

る．有用であると思われる例でもそうでない例でも，クラスタを構成する各トランザクション情報が単語を共有していたからである．有用なクラスタには，ソースコード中の特定の変数・関数名等の識別子が多く見られ，有用でないクラスタには，上記 HTML タグや，コミットログ中の一般的な単語が多く見られた．

また，WCA 法と WCD 法の比較としては，有用なクラスタの割合で後者が前者を上回っていたが，複数トランザクション情報を含むクラスタは前者の方が多く生成していることが分かった．原因としては，WCD 法では変更差分内で共通する単語の重みを下げる結果になるので，変更の特徴を際立たせる一方で，変更行数が少ない場合に特徴が得られないためであると考えられる．

そのため，両者の方法を組み合わせる等の方法によって，各変更内容に関してより特徴的な単語を抽出し，そうでない単語を除去する必要がある．

5. ま と め

本研究では，版管理システムに蓄積された変更情報をグループ化し，関連ある変更の集合を構築する手法を提案した．本手法では，コミットログと変更差分中に出現する単語を基に類似度を計測し，トランザクション情報のクラスタリングを行う．

また，提案した手法に基いてシステムを試作し，実際のソフトウェアに対し適用を試みた．その結果，関連のある変更を抽出出来る場合があることが分かったが，手法が有効に働かない場合があることも分かった．

今後の課題としては，クラスタリングの際の閾値の決定方法や単語の適切な抽出手法の確立，手法の定量的な評価等が挙げられる．また，適用対象をソースコードに限定し，本手法と静的解析手法を組み合わせることも検討している．

文 献

[1] Neville J. Ford and Mark Woodroffe. “Introducing software engineering”. *Prentice-Hall*, 2004.

[2] Jacky Estublier. “Software Configuration Management: A Roadmap”. In *Proceedings of the Conference on The Future of Software Engineering in 22nd ICSE*, pp.279–289, 2000.

[3] Peter H. Feiler. “Configuration Management Models in Commercial Environments”. *CMU/SEI-91-TR-7 ESD-9-TR-7*, 1991.

[4] Brian Berliner. “CVS II: Parallelizing Software Development”. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pp. 341–352, 1990.

[5] ViewVC. http://www.viewvc.org/.

[6] 中山 崇, 松下 誠, 井上 克郎. “関数の変更履歴と呼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 SS2004-2, Vol.104, No.47, pp.7–12, 2004.

[7] Thomas Zimmermann, Peter Weissgerber, Stephan Diehl, and Andreas Zeller. “Mining Version Histories to Guide Software Changes”. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pp. 563–572, 2004.

[8] Gerardo Canfora and Luigi Cerulo. “Impact Analysis by Mining Software and Change Request Repositories”. In *Proceedings of the 11th International Software Metrics Symposium (METRICS'05)*, 2005.

[9] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. “Indexing by Latent Semantic Analysis”. *Journal of the Society for Information Science*, Vol.41, No.6, pp.391–407, 1990.

[10] Thomas Zimmermann and Peter Weissgerber. “Preprocessing CVS Data for Fine-Grained Analysis”. In *Proceedings of 1st International Workshop on Mining Software Repositories (MSR)*, 2004.

[11] 北 研二, 津田 和彦, 獅子堀 正幹. “情報検索アルゴリズム”. 共立出版, 2002.

[12] 徳永 健伸. “情報検索と言語処理”. 東京大学出版会, 1999.

[13] 横森 励士, 梅森 文彰, 西 秀雄, 山本 哲男, 松下 誠, 楠本 真二, 井上 克郎. “Java ソフトウェア部品検索システム SPARS-J”. 電子情報通信学会論文誌 D-I, Vol.J87-D-I, No.12, pp.1060-1068, 2004.