特別研究報告

題目

ソースコード修正漏れ防止を目的とした
同時変更箇所検索ツールの出力結果の並べ替え

指導教員

楠本 真二 教授

報告者

佐飛　祐介

平成 27 年 2 月 13 日

大阪大学 基礎工学部 情報科学科

平成 26 年度 特別研究報告

ソースコード修正漏れ防止を目的とした
同時変更箇所検索ツールの出力結果の並べ替え

佐飛　祐介

内容梗概

ソフトウェア開発において，あるコード片を修正する際に他のコード片にも同様の修正を行うことがある．そのためには開発者が同時に修正を加えるべきコード片を把握している必要がある．もし修正を加えるべきコード片を把握していなければソースコードの修正漏れが発生する恐れがある．そのような修正漏れを少なくするため，同時変更箇所検索ツールがよく用いられる．現在，我々の研究グループと共同研究を行っているある企業では独自に同時変更箇所検索ツールを開発している．このツールは文字列を入力すると入力された文字列と一致する箇所をソースコード内から見つけ，その箇所を出力する．このツールから出力された文字列のある箇所全てに対して，開発者は目視で修正の有無の判断を行っている．しかし，ツールの出力の順番がファイルのパスの辞書順となっている．そのため，ツールの出力の箇所が膨大になると開発者の集中力の低下により修正漏れが発生してしまうという問題が起こっている．そこで本研究では，修正が必要となる可能性が高い順にツールの出力の順番を並べ替える手法を提案する．提案手法ではロジカルカップリングを用いて並べ替えを行う．ロジカルカップリングとはプログラム要素 (ファイル，メソッド等) 間に存在する依存関係を指す．ロジカルカップリングの関係にあるプログラム要素は同時に修正が必要となる可能性が高い．そのためロジカルカップリングを用いると修正が必要となる可能性が高い順にツールの出力の順番を並べ替えることができる．本研究では，ある企業が開発している 3 つのシステムに対して実験を行い，並べ替えを行わない場合との比較を行った．実験の結果，提案手法の並べ替えによって修正されたファイルを上位に表示できた件数がそうでない件数より多いことを確認した．特にツールの出力が 50 箇所以上の場合には，提案手法を用いた並べ替えがより有効に働くことを確認した．

主な用語

ソフトウェア開発
ロジカルカップリング
コード片検索
産学連携

目 次

図 目 次

表 目 次

1　はじめに

ソフトウェア保守は，ソフトウェアの開発に関する全ての工程において最もコストがかかる工程である [1][2]．ソフトウェアの開発，保守においては，あるコード片を変更する際に他の類似するコード片にも同様の変更を加えるか否かを検討する必要がある．例えば，バグ修正および機能の追加・削除等の修正をコード片に対して行う際には，修正すべき全てのコード片を把握しなければならない．もし把握していなければ，修正すべきコード片の見落としによる修正漏れが発生する恐れがある．修正漏れが発生すると，後のバグの要因となり，システムの可用性が低下する要因となる．

このような修正漏れを防ぐために，修正前のコード片に対して grep 等の文字列検索ツールがしばしば用いられる [3]．文字列検索ツールは入力として文字列を与えると，入力された文字列と一致する箇所をソースコード内から見つけ，その箇所を表示するツールである．あるコード片を修正する際にコード片に関連する文字列を文字列検索ツールに入力すると，同じ文字列を含む箇所が出力される．そのため，同様の修正を加える必要のある箇所をもれなく確認することが出来る．本研究では，このような目的で使用される文字列検索ツールを**同時変更箇所検索ツール**と呼ぶ．開発者はツールにより出力された箇所に対して 1 つずつ修正を加えるか否かを判断する．このように同時変更箇所検索ツールを用いることで，修正すべきコード片を見落とす可能性は低くなる．

ある企業では，同時に修正すべき箇所の見落としをなくすために，独自に同時変更箇所検索ツールを開発している．このツールは，開発者から入力として文字列を受け取り，入力された文字列を含む箇所のメソッド名，ファイル名，行番号などの情報を要素とするリストを開発者に提示する．開発者はリストの要素を選択することで，選択した要素に該当する箇所を閲覧することができる．リストの要素は，キーワードが含まれるファイル名の辞書順に表示される．開発者は，そのリストの上から順に修正が必要か否かを確かめる．そのためリストの要素数が多い場合，リストの下部に修正すべき箇所があると集中力の低下により見落としが発生してしまうことが，課題となっている．

そこで本研究ではロジカルカップリングを利用して，ツールの出力の順番を並べ替える手法を提案する．提案手法は検索を行う前にソフトウェアの開発履歴を解析し，ファイル単位のロジカルカップリングを抽出する．検索を実行した時に，抽出したファイル単位のロジカルカップリングを用いてツールの出力の順番を並べ替える．

ロジカルカップリングとは，プログラム要素 (ファイル，メソッド等) 間に存在する論理的な依存関係を指す．ファイル単位のロジカルカップリングはソフトウェアの開発履歴から過去に同時に更新が行われたファイルを抽出することで容易に得ることができる [4]．ロジカルカップリングを元にして，ある修正が行われたときに同時に修正が加えられるであろう

プログラム要素を予測する手法も存在する [5].

ロジカルカップリングを用いた出力結果の並べ替えによって本当に修正が必要な要素をリストの上位に表示することができているかを確認するために，提案手法をある企業で現在開発されている 3 つのシステムに対して適用した．その結果，検索結果が 50 箇所以上表示される場合，並べ替えによって修正されたファイルをリストの上位に表示できていることを確認した．

以降 2 章では，本論文にて用いる技術および用語について述べる．3 章で提案手法について説明し，4 章で評価実験について説明する．5 章で実験結果の考察を行う．6 章で妥当性について述べ，7 章では関連研究について述べる．最後に 8 章で本研究のまとめと今後の課題について述べる．

2　準備

本論文で用いる技術および用語について説明する．

2.1　同時変更箇所検索ツール

ソフトウェアの保守において，あるコード片を修正する際に他のコード片にも同様の修正を加えるか否かを検討する必要がある. そのため，同時に修正を加えるか否かを検討するべき全てのコード片を把握する必要がある．もし，そのようなコード片を把握していなければ修正漏れが発生する可能性がある．修正漏れが発生すると，後のバグの原因となり，システムの可用性が低下する要因となる．それだけでなく修正漏れが見つかった場合に，それを修正するための追加コストが発生してしまう．このような修正漏れを防ぐために，文字列検索ツールがしばしば用いられる [3]．あるコード片を修正する際にコード片に関連する文字列を文字列検索ツールに入力すると，同じ文字列を含む箇所が出力される．その文字列このような目的で使用される文字列検索ツールを**同時変更箇所検索ツール**と呼ぶ.

例えば，”keywordA” という変数名を”keywordB” に変更する修正を考える．ここで開発者が同時変更箇所検索ツールに”keywordA” を入力として与えると，ソースコード中における全ての”keywordA” の出現箇所を開発者に提示する．開発者は”keywordA” の出現箇所を 1 つずつ目視で判断し，実際に”keywordB” に変更する必要があるか否かを判断する．このように同時変更箇所検索ツールを用いることで，修正漏れを削減できる.

文字列検索ツールは，多数のテキストエディタや開発環境に標準機能として実装されている．そのため，それらのツールを同時変更箇所検索ツールとして使用することができる.

2.1.1　grep

grep は UNIX 系のオペレーティングシステムにおける検索コマンドである．grep に対してキーワードを与えると，テキストファイル中におけるキーワードの出現箇所を開発者に提示する．基本的な使い方は以下の通りである.

```
grep <オプション>  <キーワード>  <テキストファイル>
```

単に入力されたキーワードと一致する箇所をテキストファイルから検索するだけでなく，オプションを利用することで入力されたキーワードが含まれる行数なども取得できる.

2.1.2　PowerUnit

ある企業では，修正漏れを防ぐために PowerUnit というツールを独自に開発している．PowerUnit とは grep に除外ファイルの設定などの機能拡張を施した Eclipse プラグインで

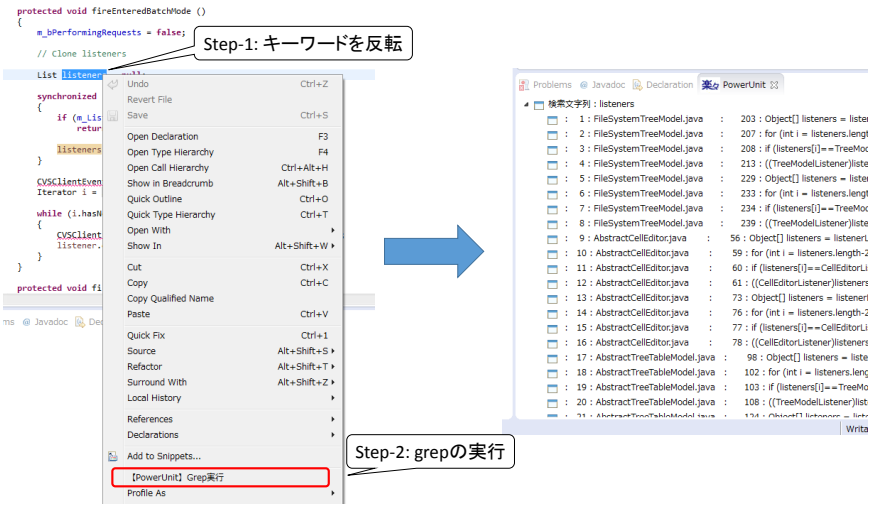

図 1: PowerUnit の使用例

ある．PowerUnit の入力はソースコードより選択されたキーワードである．出力はプロジェクトのソースコード内において，入力されたキーワードと一致した箇所に関する情報を要素とするリストである．リストの要素に含まれる情報は以下に示す 5 つである．

- キーワードが出現するファイル名
- キーワードが出現するクラス名
- キーワードが出現するメソッド名
- キーワードが出現する行のソースコード
- キーワードが出現する行番号

PowerUnit の使い方は以下の通りである．

Step-1: ソースコードのビューを開き，入力したいキーワードを反転させる．

Step-2: キーワード上で右クリックを行い，開かれたメニューから grep を実行する．これにより PowerUnit はプロジェクトの中からキーワードを検索する．

これにより，キーワードに一致した全ての箇所の情報がリスト形式で表示される．また，リストの要素をダブルクリックすることで，ビューにキーワードの出現箇所が表示される．さらに PowerUnit は以下の機能をもつ．

- 指定したファイル（*Test.java 等）を検索結果から除外する機能

- 見つかったキーワードに関する以下の情報を Excel ファイルに出力する機能
  - キーワードが含まれるファイル名，メソッド名およびキーワードが出現する行番号
  - そのキーワードに関する修正を行う（もしくは行った）か否か

PowerUnit では，リストの要素はキーワードを含むファイルのパスの辞書順に表示されている．また，同一ファイルにおいて複数の箇所が検出された場合はそのキーワードが見つかった行数の昇順に表示されている．見つかったキーワードについて修正を行うか否かは，リストの上位から順に目視で判断することとなる．そのため，多数の検索結果がリストに表示された場合，集中力の低下により修正漏れが発生する恐れが大きくなるという問題点が存在する．特にリストの下部に表れる要素については実際に修正漏れが起こっていることが，課題となっている．

2.2 バージョン管理システム

バージョン管理システムは，ソフトウェア開発において使用されるソースコードやその他のリソースについて，開発者間での共有や開発履歴の管理を目的として利用されているシステムである．バージョン管理システムでは管理対象となるリソースやそれらの開発履歴を保存するデータベース (以下リポジトリと表記する) をもつ．リポジトリで管理されているリソースに対して変更を反映させるコミットなどの操作はこのリポジトリに対して行われる．また，バージョン管理システムにおいて，開発状況の過去の一点におけるリソースのことをスナップショットと呼ぶ．リポジトリに保存されているリソースや開発履歴を解析することで，開発者にとって有用な情報を提供することが可能となる [6][7]．このようなリポジトリマイニング (mining software repositories) と呼ばれる技術が近年注目されている [8]．バージョン管理システムには CVS (Concurrent Versions System)，SVN (Subversion)，Git など様々な種類が存在する．

図 2 はバージョン管理システムにおける開発履歴の例を示している．図 2 での矢印は開発の流れを示しており，右に行くほど開発が進んでいることを指す．また，各白丸は開発途中でのリビジョンを指し，その上部に表示されている文字列はリビジョン番号である．また，その下部に表示されているファイルはそのリビジョンで更新されたファイル群である．

2.3 ロジカルカップリング

ソフトウェア開発においては, プログラムの一部が変更された際に同時に変更しなければいけない箇所がある．そのようなプログラム要素 (ファイル，メソッド等) 間には論理的結合関係が存在する [4].

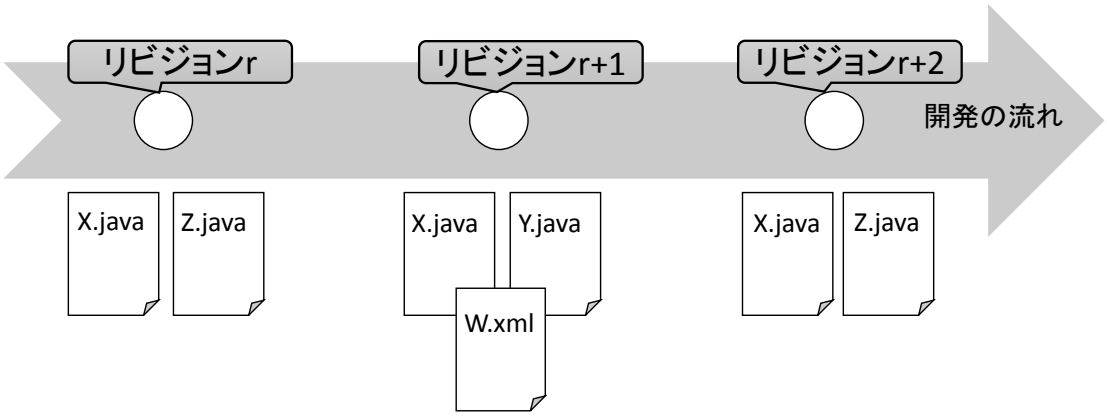

図 2: バージョン管理システムにおける開発履歴の例

ロジカルカップリングはプログラム要素間に存在する論理的結合関係を指している [9]. ファイル単位のロジカルカップリングは同一開発者によって極めて短時間，もしくは同時刻に更新されることが多い [9]，そのため，開発履歴から開発者，更新時刻を解析することによって，ファイル単位のロジカルカップリングを抽出することが可能になる．また，ロジカルカップリングはソフトウェアの欠陥と関係があることが研究によって明らかになっている [10]．開発履歴からロジカルカップリングを抽出し，ソースコードの変更時にその関係を提示することによって変更作業の効率化を目指す研究が行われている [5][11][12].

3　提案手法

本研究で提案する手法について述べる．

3.1　概要

本研究の目的は，同時変更箇所検索ツールを用いたソースコードに対する修正において，漏れを減らすことである．そこで本研究では，同時変更箇所検索ツールにおいて修正される可能性が高い箇所を検出し，その箇所を上位に並べ替える手法を提案する．

提案手法を用いることで, 修正される可能性が高い箇所が上位に現れるため，集中力の低下によって生じる修正漏れを減らすことができる．本手法では，同時変更箇所検索ツールとして PowerUnit，ツールの出力の並べ替えには抽出されたファイル単位のロジカルカップリングを用いる．

提案手法の入力を以下に挙げる．

- リポジトリや出力パスなどの情報を記載したテキストファイル
- PowerUnit から出力された検索結果のリスト
- 検索を行う際に入力されたキーワードを含むファイルのパス

この実装したツールにおいて，入力として与えるテキストファイルに記載されている情報は以下の 4 つである．

- リポジトリのパス
- ファイル単位のロジカルカップリングのリストを出力するパス
- ファイルのペアを同時更新とみなす閾値 (秒)
- 現時点から何日前までの開発履歴を利用してファイルを抽出するかという開発履歴の利用範囲（日）
  - 値を-1 とすると開発履歴全体を利用する

出力は，ロジカルカップリングを用いて並べ替えが行われた PowerUnit のリストである．提案手法全体の流れを図 3 に示す．本手法は以下に示す 2 つの Step で行われる．

Step-1:　ファイル単位のロジカルカップリングの抽出

Step-2:　リストの要素の並べ替え

以下，提案手法の各 Step について詳細に述べる．

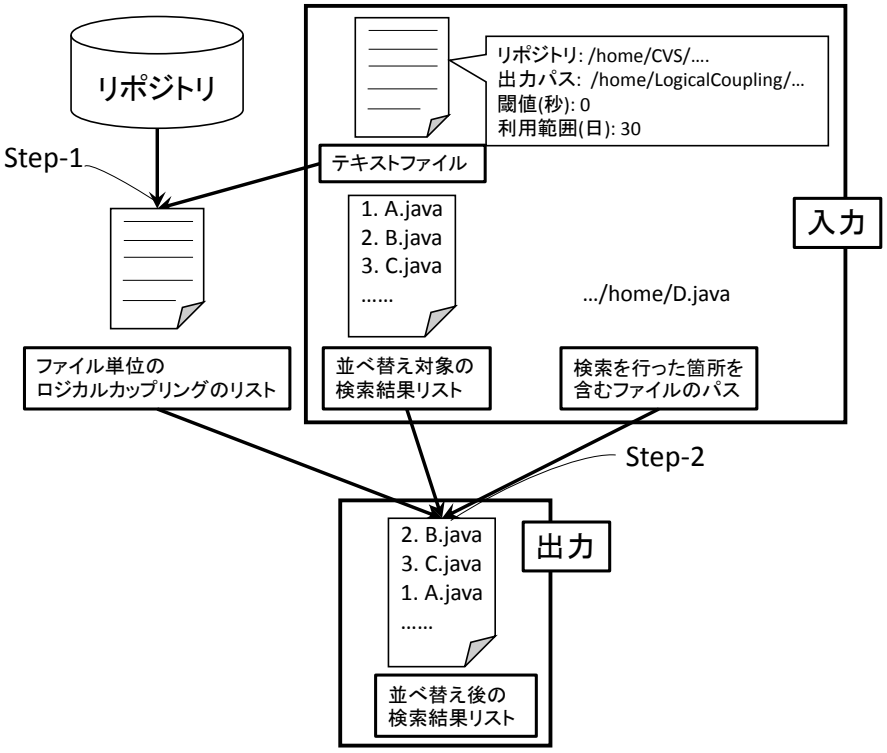

図 3: 提案手法全体の流れ

3.2 Step-1: ファイル単位のロジカルカップリングの抽出

この Step では検索対象となるソースコードを持つソフトウェアのリポジトリを解析し，ファイル単位のロジカルカップリングのリストを得る．本手法におけるファイル単位のロジカルカップリングの定義は以下の 2 つの条件に合致するファイルペアである．

- 2 つのファイルの更新時刻の差が，開発者が指定する閾値以下である
- 同じ開発者によって更新されている

入力として与えるテキストファイルにはリポジトリのパスなどの情報が記載されている．その情報を元にしてリポジトリにアクセスし，開発履歴を取得する．開発履歴から同時に更新されたファイル群を取り出し，全通りのファイルのペアを作り出す．

以下，図 2 を用いてロジカルカップリングの関係にあるファイルペアの抽出の例について説明する．括弧内に書かれた 2 つのファイルを同時更新されたファイルとして表すと，図 2 の例においては，同時更新されたファイルのペアとして

リビジョン r: (X.java,Z.java)

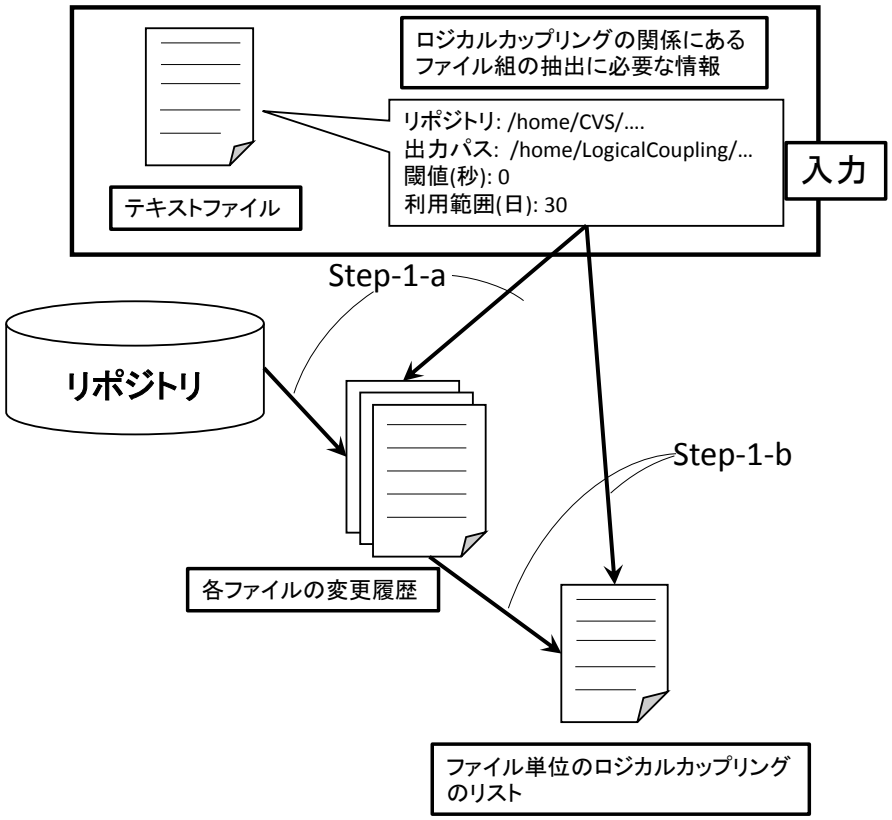

図 4: ファイル単位のロジカルカップリングを抽出する流れ

リビジョン r+1: (X.java,Y.java), (X.java,W.xml), (Y.java,W.xml)

リビジョン r+2: (X.java,Z.java)

が抽出され，それぞれがファイル単位のロジカルカップリングとなる．それぞれのファイルのペアをリストにした形でファイルに出力する．

本手法において，ファイル単位のロジカルカップリングを抽出する流れを図 4 に示す，ファイル単位のロジカルカップリングの抽出は以下の 2 つの Step で行われる．

Step-1A: 各ファイルのコミット情報などの開発履歴の取得

Step-1B: 同時更新の条件に合致するファイルのペアの抽出

以下，ロジカルカップリングのデータ抽出における 2 つの Step について，その処理内容を詳細に述べる．

3.2.1 Step-1A: 各ファイルにおける開発履歴の取得

この Step では入力されたリポジトリのパスを利用してリポジトリにアクセスする．アクセスした後，リポジトリから各ファイルの開発履歴を取得する．また開発履歴から，ファイルごとの以下の情報を取得する．

- ファイルのパス
- コミットされた日時
- コミットした開発者

3.2.2 Step-1B: 同時更新の条件に合致するファイルのペアの抽出

この Step では Step-1A で取得した情報，また同時更新とみなす閾値と開発履歴の利用範囲を利用して，同時更新の条件に合致するファイルのペアを抽出する．具体的には以下の手順で行われる．

1. 開発履歴よりコミットされた日時が利用範囲内にある全てのコミットを抽出する
2. その中から同じ開発者によってコミットされ，コミットされた日時の差が同時更新とみなす閾値以内であるファイル同士を見つけてペアを作る

この 2 つの手順を行い，全ての同時更新の条件に合致するファイルのペアを全て抽出する．

3.3 Step-2: リストの要素の並べ替え

この Step-2 では Step-1 で得られたファイル単位のロジカルカップリングと PowerUnit から得られた出力, また検索を行った箇所を含むファイルのパスを用いて，リストの要素の並べ替えを行う．まず，入力として与えられたファイルのパスから，そのファイルと同時に更新されたファイル，そして Step-1B よりそのファイルとの同時更新回数を得る．次に，その抽出したデータを用いて並べ替えを行う．並べ替えは同時更新回数の降順になるように行う. 検索結果のうち検索を行った箇所を含むファイル内に見つかったものは，他のどの検索結果よりも上位になるように並べ替えを行う．それ以外のファイルで同時更新回数が等しいものは，ファイルのパスの辞書順に並べ替え，同ファイル内に見つかった検索結果は見つかった行数の昇順に並べ替える．図 5 を用いて並べ替えの例を示す，あるプロジェクトにおいて X.java の a の箇所に存在する “add(” をキーワードとしてプロジェクト内で検索を行うと Y.java と Z.java 内にキーワードが見つかる．ハイライトされた箇所がキーワードの見つかった箇所であり，それぞれを識別するために a から f のアルファベットで表す．

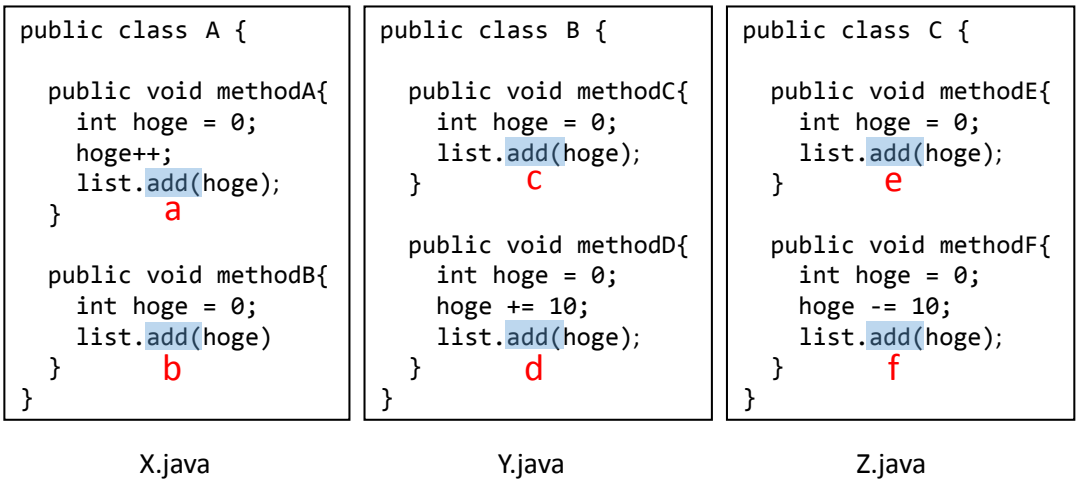

図 5: 同時変更箇所検索を行うソースコードの例

PowerUnit の出力ではキーワードを含むファイルのパスの辞書順に, 同ファイル内の検索結果は見つかった行数の昇順に表示されるため,

```
a -> b -> c -> d -> e -> f
```

という順番に検索結果が表示される.

ここでバージョン管理システムにおける開発履歴として図 2 に示したものを用いる. このとき X.java と Y.java の同時更新回数が 1 回, X.java と Z.java の同時更新回数が 2 回となる. さらに検索結果のうち検索を行ったファイルと同じ X.java 内に見つかったものは他のどのファイルよりも上位に並べ替えを行うため

```
a -> b -> e -> f -> c -> d
```

という順番に検索結果を並べ替える.

4　評価実験

本研究において行った実験について説明する．

4.1　目的

この実験の目的は, 提案手法を用いることで実際に修正すべきファイルが上位になるように並べ替えられているかを確かめることである．

4.2　方法

この実験を行うにあたり，以下のデータを使用した．

- ある企業で開発を行っている 3 つのシステム，3 つのシステムの詳細を表 1 に示す．
- システム A〜C に対して行われた過去の検索の日時，検索結果，結果に対する修正の有無が記載された Excel ファイル．

Excel ファイルの例を図 6 に示す．図 6 の例では検索キーワード，そして検索結果として見つかったキーワードそれぞれについて順位，キーワードが見つかったファイルのパス，修正の有無が記載されている．また，修正有となっている箇所をハイライトして表示している．また実験には順位データを利用する．順位データとは，Excel ファイルに含まれる順位を上位から順に並べたものである．図 6 の例では,（1,2,3,4,5,6,7,8）を順位データとみなす．

これらのデータを用いて以下の Step で実験を行った．

Step-1:　企業から提供して頂いた Excel ファイルから並べ替え前の順位データを取得する

Step-2:　検索が行われた当時のスナップショットをリポジトリから取得する

Step-3:　Step-2 において得られた当時のスナップショットを検索対象として，提案手法を用いて並べ替えた検索結果と順位データを取得する

Step-4:　Excel ファイルにおいて修正が行われたファイルを全て上位に並べ替えた理想的な検索結果および順位データを作成する. 理想的な検索結果については後述する．

表 1: 実験対象システム

名前	コミット数	開発者数	開発期間	過去の検索回数
システム A	約 26,000 回	9 人	約 6 年	18 回
システム B	約 260 回	45 人	約 7ヶ月	11 回
システム C	約 700 回	13 人	約 1 年	28 回

検索キーワード:add(		検索日時:14/04/01	
順位	ファイルパス	行数	修正の有無
1.	/home/A.java	15行目	修正無
2.	/home/B.java	86行目	修正無
3.	/home/C.java	46行目	修正有
4.	/home/D.java	13行目	修正有
5.	/home/E.java	36行目	修正有
6.	/home/F.java	19行目	修正無
7.	/home/G.xml	98行目	修正無
8.	/home/H.java	31行目	修正有

図 6: Excel ファイルの例

Step-5: Step-1,3 で得られた順位データと Step-4 で得られた理想的な順位データをそれぞれ相関係数を用いて比較する

Step-6: Step-1〜5 を全ての Excel ファイルに対して適用する

この実験において対象にするバージョン管理システムは CVS [13] である．また，同時更新とみなす閾値を 0 秒，開発履歴の利用範囲を開発履歴全体とする．

4.2.1 理想的な検索結果および順位データの作成

図 6 はこの実験において使用した Excel ファイルの例である．この例を用いて Step-4 の理想的な検索結果と理想的な順位データの作成について説明する．図 6 の検索結果から理想的な順位データを生成する例を図 7 に示す．図 7 では左の PowerUnit の検索結果のうち，修正有の箇所が上位になるように並べ替える．このうち，修正有の箇所は過去に実際に修正が行われたファイルであり，提案手法において上位に並べ替えられるべき箇所である．修正有の箇所が複数ある場合は並べ替え前の順位の昇順に並べ替える．そのため，修正有となっている 3, 4, 5, 8 位の項目を全て上位になるように並べ替える．並べ替えた後のデータより，理想的な順位データを抽出する．図 7 の例では理想的な順位データは（3,4,5,8,1,2,6,7）となる．

4.2.2 相関係数

この実験において用いる相関係数について説明する．

今回の比較における確率変数は順位であり，正規分布ではない．そのため，スピアマンの順位相関係数 [14] を用いて順位データ間の比較を行う．スピアマンの順位相関係数は-1 から 1 の値をとり，-1 に近いほど 2 つの順位データ間の負の相関が強く，1 に近いほど正の相関が強くなる．

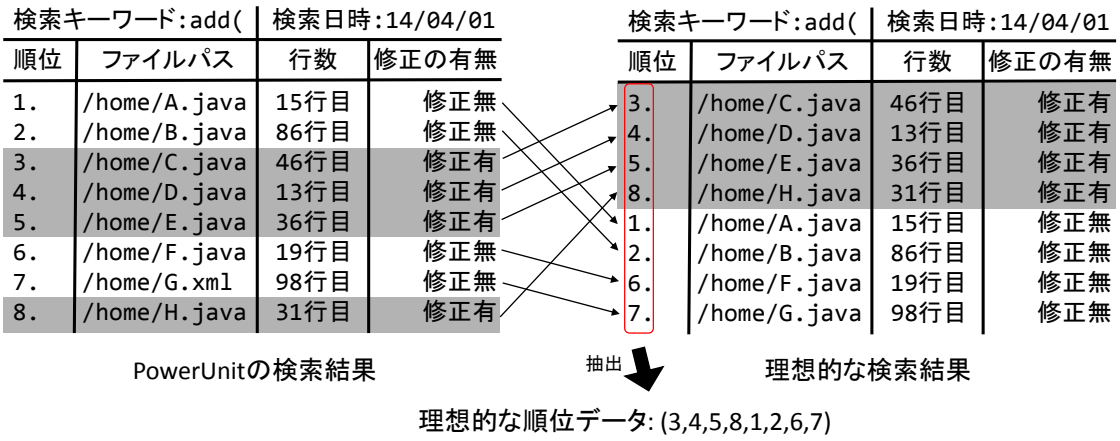

図 7: 理想的な順位データの生成例

ある 2 組の順位データ X,Y 間のスピアマンの順位相関係数 ρ は以下の計算式で求められる．

$$\rho = 1 - \frac{6\sum D^2}{N^3 - N} \tag{1}$$

ここで D は対応する X と Y の対応する順位の差，N は値のペアの数である．図 8 は実験において算出された相関係数の計算例である．図 8 の上部は PowerUnit による順位データと理想的な順位データの相関係数の計算を，下部は提案手法の順位データと理想的な順位データの相関係数の計算を示す．この例の場合，図 8 上部の相関係数 ρ_1 は

$$\rho_1 = 1 - \frac{6 * (2^2 + 2^2 + 2^2 + 3^2 + 3^2)}{5^3 - 5} = 1 - 1.5 = -0.5$$

図 8 下部の相関係数 ρ_2 は

$$\rho_2 = 1 - \frac{6 * (1^2 + 1^2 + 1^2 + 0^2 + 3^2)}{5^3 - 5} = 1 - 0.6 = 0.4$$

となる．ρ_1 よりも ρ_2 のほうが相関係数が高く，提案手法の順位データと理想的な順位データとの方が正の相関が強い．このことより，図 8 の例では提案手法の並べ替えによって修正すべきファイルを上位に並べ替えられていることがわかる．以下，PowerUnit の順位データと理想的な順位データとの相関係数を **PowerUnit の相関係数**, 提案手法の順位データと理想的な順位データとの相関係数を**提案手法の相関係数**と呼ぶ.

4.3 結果

以上の実験方法を 3 つのシステムに適用した結果を図 9 に載せる．横軸に並べ替え前の相関係数，縦軸に並べ替え後の相関係数をとっており，システム A を丸，システム B を三角，

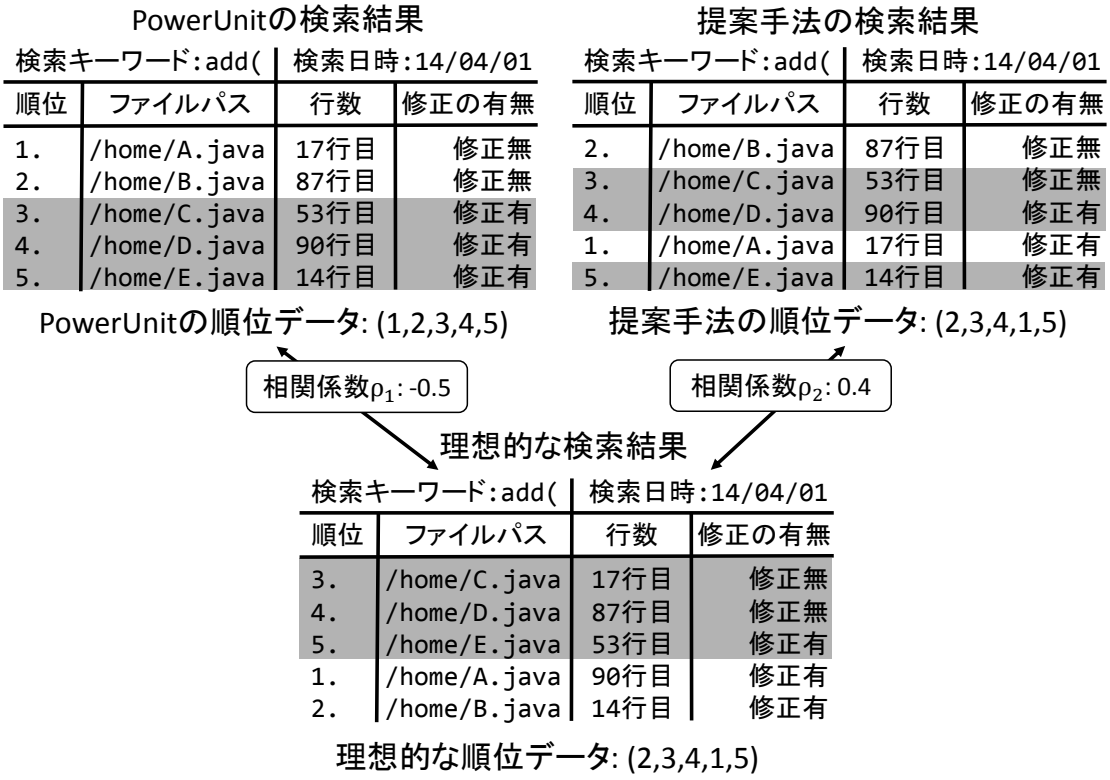

図 8: 順位データの例

システム C を四角で表している．また，図形内が塗りつぶされているマーカーは検索結果が 50 箇所以上であったことを示す．これは座位での単純作業の場合，作業開始より 50 分経過するとミスの発生確率が高くなるとの報告がある [15]. 1 つの検索結果に対して修正の有無を判断するのにかかる時間を 1 分と仮定すると，50 箇所以上の検索結果において該当するためである．また PowerUnit の相関係数を x，提案手法の相関係数を y としたときの $y = x$ の直線を各グラフに表示している．この直線上にプロットされているマーカーは相関係数の変化が無かった例となる．

最後に，横軸に検索結果の箇所数，縦軸に PowerUnit と提案手法の相関係数の変化をとったグラフを図 10 に載せる．縦軸の相関係数の変化は提案手法の相関係数から PowerUnit の相関係数を引いた数値である．直線の上部にプロットされている点ほど並べ替えによって相関係数が増加し，理想的な検索結果に近づいていることを示す.

図 9,10 より，並べ替えによって相関係数の変化が生じていることがわかる．また，システム A,B ではマーカーが $y = x$ の直線から離れているものが多い, このことによって並べ替えによって相関係数が多く変化した例が多いことがわかる．対照的にシステム C では直線 $y = x$ 上にあるマーカーが多く，並べ替えによって相関係数が変化していない例が多いこと

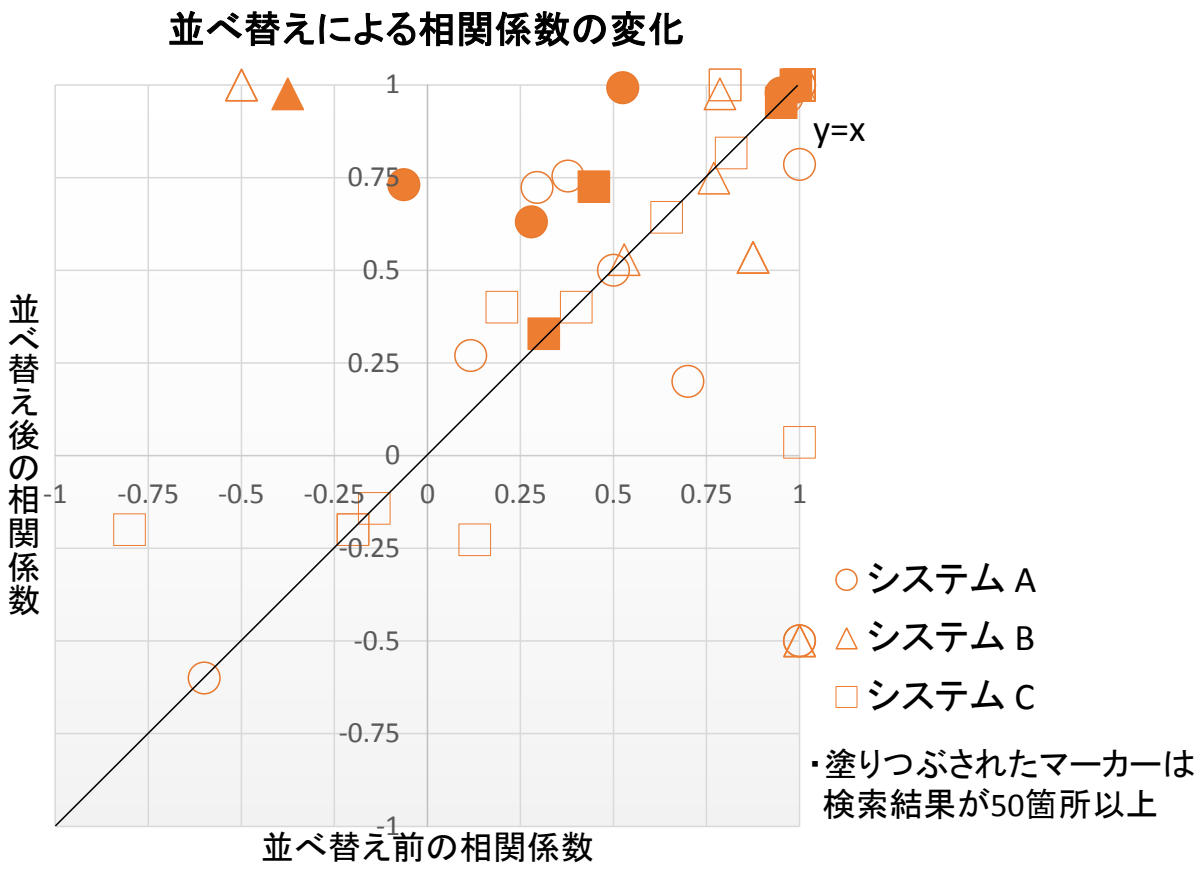

図 9: 並べ替えによる相関係数の変化

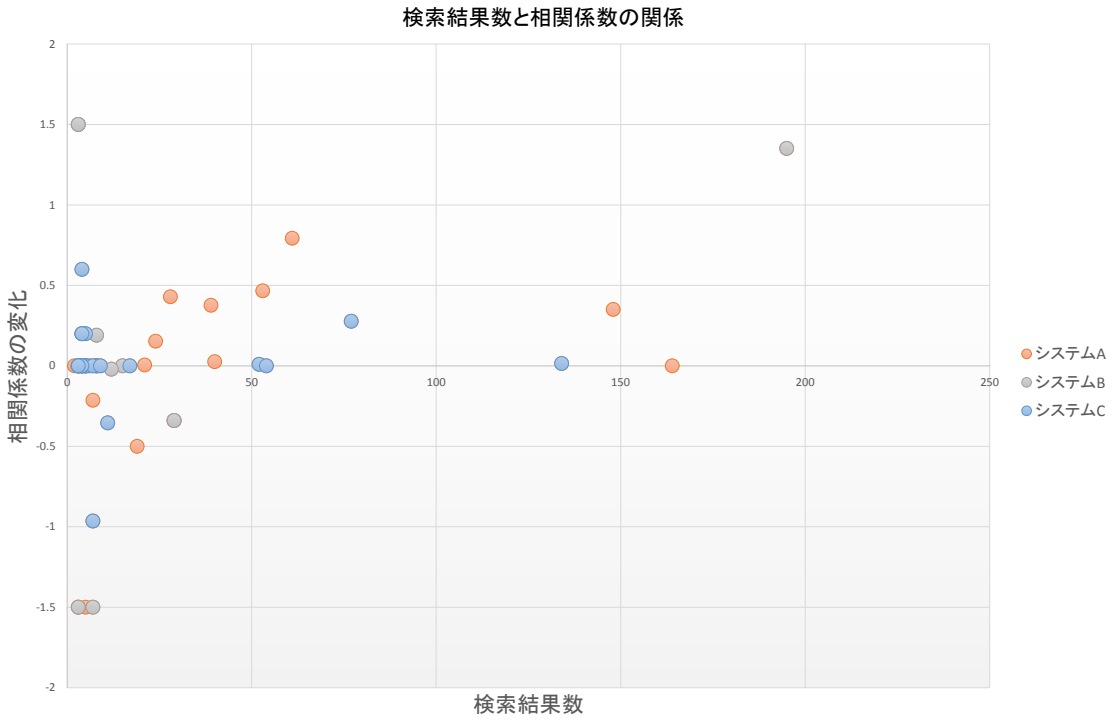

図 10: 検索結果の箇所数と相関係数の変化の関係

がわかる．図 10 では左のほうにマーカーが集中しており，また大きく相関係数が変化しているものが見られる．このことより，検索結果の箇所数が少ないほど並べ替えによる相関係数の変化が大きいことがわかる．

5　考察

4 章で得られた実験結果に対する考察を行う．

以下，各実験対象に対して考察を行う．各システムの検索結果において相関係数が増加した件数，減少した件数，変化しなかった件数を表 2 に示す．システム A, B, C 全体の検索結果である 57 件中 33%にあたる 19 件が相関係数が増加している．また，相関係数の増加の件数は減少の件数よりも多い．このため，提案手法の並べ替えによって相関係数が減少した件数より, 相関係数が増加した件数の方が多いことがわかる．

5.1　システム A について

図 9 と表 2 より，システム A については 18 件中 8 件が並べ替えによって相関係数が増加している．相関係数が変化していない 6 件に対しても，そのうち 5 件において並べ替え後の相関係数が正の値となっている．よってシステム A における検索結果全体の約 72%にあたる 13 件が並べ替えにより，修正が行われたファイルを上位に表示できている．

しかしながら，並べ替えによって相関係数が減少し理想的な検索結果から遠ざかった事例も 18 件中 4 件存在する．その内 1 件は相関係数が並べ替え前の 1 から並べ替え後の-0.5 に変化し，並べ替えによって修正が行われたファイルを下位に表示している．他の 3 件も，0.2 から 0.5 ほど並べ替えによって相関係数が減少し，修正が行われたファイルを下位に表示している．しかし，検索結果が 50 箇所以上である四角のマーカーに着目すると並べ替えによって相関係数が減少しているマーカーは存在しない，これよりシステム A において検索結果が 50 箇所以上の検索結果については提案手法が有効である．

5.2　システム B について

図 9 と表 2 より，システム B においてはシステム A, C よりも並べ替えによって大きく相関係数が変化したものが数件見られる．その内 1 件は検索結果が 50 箇所以上と大きく，また相関係数が並べ替え前の-0.375 から並べ替え後の 0.976 と大きく増加していた．このこと

表 2: 各システムにおける相関係数の変化

	増加	変化なし	減少
システム A	8 件	6 件	4 件
システム B	4 件	2 件	5 件
システム C	7 件	19 件	2 件
合計	19 件	27 件	11 件

より，システム B においても検索結果が 50 箇所以上の検索結果については提案手法によって修正が行われたファイルを上位に表示できていることがわかる．

5.3 システム C について

図 9 と表 2 より，システム C については直線 $y = x$ の近くにプロットされている点が多く，システム A, B と比較すると並べ替えによる相関係数の変化は小さい．変化なしのものについては検索結果全体の約 68%にあたる 28 件中 19 件となっている．システム C にて 4 件存在する検索結果が 50 箇所以上の検索結果に関しても 4 件中 3 件が $y = x$ の直線上に存在し，相関係数の変化がない．しかし，残りの 1 件は提案手法の並べ替えによって修正が行われたファイルを上位に表示できている．以上より，システム C については 4 件中 1 件ではあるが検索結果が 50 箇所以上の検索結果については提案手法によって修正が行われたファイルを上位に表示できていることが分かる．

5.4 検索結果の箇所数と相関係数の変化について

図 10 より，検索結果が 50 箇所以上の事例に関しては提案手法の並べ替えにより相関係数が減少しているものはない．しかし，検索結果が 50 箇所未満の事例に関しては相関係数が大幅に増加したもの，減少したものが混在している．これは，スピアマンの順位相関係数の性質上，確率変数が少ない場合には並べ替えで相関係数の値が大幅に変わるためである．また，相関係数が大幅に減少した事例は検索結果が 50 箇所未満の場合に集中している．これらの事例については修正すべきファイルが下部に並べ替えられている．しかし，検索結果が少ないため，集中力の低下による修正漏れが発生する恐れは小さいと考えられる．また，検索結果が 50 箇所以上の場合には提案手法が有効であるといえる．

6　妥当性への脅威

本研究の結果の妥当性に関して，以下で挙げる点に留意する必要がある．

6.1　実験対象

本研究において，企業で開発が行われた 3 つのシステムを対象にして実験を行った．しかしながら，実験に使用した 3 つのシステムの中には開発期間の短いものやコミット数の比較的少ないものも存在している．そのため，実験対象を別の企業のソフトウェアにして実験を行った場合，本実験で得られた結果とは異なる結果が得られる可能性がある．

6.2　Excel ファイル

本研究で行った実験において，上位に並べ替えられるべきファイルの判定は，過去の検索結果に基づいた修正が行われた否かを基準にしている．しかし Excel ファイルが作成された当時，行われた修正は PowerUnit を用いた検索結果に基づいて行われたものであり，集中力の低下により修正漏れとなったファイルが存在する可能性がある．そのようなファイルも修正有とした上で実験を行った場合，本実験で得られた結果とは異なる結果が得られる可能性がある．

6.3　ロジカルカップリング

本研究では，ファイルが同時変更され，ファイル間に依存関係があるとみなす時間を 0 秒として，ファイル単位のロジカルカップリングを抽出している．しかし，ロジカルカップリングに対して依存関係があるとみなす閾値を大きくすると，ロジカルカップリングとして抽出されるファイルのペアが変わる．そのため，本実験で得られた結果とは異なる結果が得られる可能性がある．

7 関連研究

7.1 キーワード検索

泉田らは修正が必要なコード片の見逃し防止のために，grep とコードクローン検出ツールが有効であることを示している [16].

Shonle らはキーワード検索の結果を効果的に可視化する方法として AspectBrowser を提案している [17]. AspectBrowser はソースファイルをその長さに比例したサイズの縦長の矩形で表現し，プログラム中の全ファイルを 1 画面に並べる．開発者が指定したキーワードを含む行に対応する位置に着色することでキーワードが複数のファイルに渡ってどのように存在しているかを可視化している．

石尾らはキーワード検索によって抽出された手続き集合に対し，共通する性質を抽出し表形式で可視化を行う方法を提案している [18].

7.2 ロジカルカップリング

Zimmermann らはバージョン管理システムの変更履歴から同時に変更されたプログラム要素を抽出することにより，変更されるファイルを提示する手法を提案している [5]. Zimmermann らは提案手法を用いた実験により，変更時に適切なファイルを提示できることを示している．

また，Ying らは同様にバージョン管理システムの変更履歴から同時に変更されたファイル群を抽出し，ファイルの変更パターンを提示する手法を提案している [11]. Ying らは提案手法をオープンソースソフトウェアに適用することによって，バージョン管理システム内の変更履歴からファイルの変更パターンを調査することの有用性を示している．

Ying らの手法はバージョン管理システムの変更履歴よりファイル単位やより細かい粒度でのロジカルカップリングを抽出し，それらを用いて変更される可能性の高いファイルを提示するという点で，提案手法と類似している．しかし，これらの手法は変更の可能性が高いファイルを提示するのみである．そのため，各ファイルに同時更新回数の重みを付け，変更される可能性の高い順にファイルを並べ替えるという点で，提案手法は既存手法に比べて優れているといえる．

Robbes らはファイル単位よりも細かい粒度での変更情報を用いて, ロジカルカップリングを抽出する手法を提案している [19]．Robbes らの手法ではロジカルカップリングを抽出する際，同時に変更されたプログラム要素のみに着目するのではなく，他の様々なメトリクスを用いていることが特徴的である．これにより，開発期間が少ない場合でも，多くのロジカルカップリングを抽出できることを示している．

松村らはファイルの同時更新回数のみでなく，ファイルの変更内容を分析することにより，同時更新されるファイル間のロジカルカップリングの有無をより詳細に判定する手法を提案

している [9]. 松村らはこの提案手法をオープンソースソフトウェアに適用し，同時更新だけでは抽出できないプログラム要素間のロジカルカップリングを抽出できることを示している．

Geiger らはコードクローンとロジカルカップリング間の依存関係を調べるためのフレームワークを提案し，それらをオープンソースソフトウェアに適用することにより，コードクローンとロジカルカップリング間の依存関係を調査している [20]. また，そのフレームワークを用いてどのファイルにコードクローンやロジカルカップリングがどの程度存在するかを可視化する手法も提案している．

また，ロジカルカップリングの可視化手法として Hanakawa らや Lanza らはファイル単位とモジュール単位のロジカルカップリングに対応した可視化手法を提案している [21][22].

8　おわりに

本研究では，同時変更箇所検索ツールにおいて要修正箇所を上位に並べ替える手法を提案した．提案手法は，まず開発履歴を分析し，ファイル単位のロジカルカップリングを取得する．次に，取得したファイルのペアを用いて，同時変更箇所検索ツールにおける検索結果の並べ替えを行う．

また，提案手法の有効性を調べるために企業内で開発を行っている 3 つのシステムに対して実験を行った．実験の結果，提案手法の並べ替えによって修正されたファイルを上位に並べ替えられた件数が下位に並べ替えられた件数より多いことを確認した．また，検索結果が 50 箇所以上出力される場合，提案手法がより有効に働くことを確認した．

今後は，コードクローン等の他の技術と組み合わせることで提案手法の精度向上に取り組む予定である．

謝辞

本研究を行うにあたり，ご多忙にも関わらず暖かく励まして頂き，常にお心遣い頂きました 楠本 真二 教授に心から感謝申し上げます．

本研究に関して，常時有益なご助言を多数頂き，的確なご指導を賜りました 岡野 浩三 准教授に深く感謝申し上げます．

本研究において，常に鋭く的確なご助言を頂き，様々な面より親切なご指導を賜りました 井垣 宏 特任准教授に深く感謝申し上げます．

本研究の全過程を通し，常に熱心かつ丁寧なご指導を賜り，日頃の議論を通じて的確なご助言と多大なるご助力を頂きました 肥後 芳樹 助教に心より深く感謝申し上げます．

本研究において，常に多大なるご協力を頂き，快く実験データを提供して頂きました，住友電工情報システム株式会社の 松岡 宏和 さん，豊川 宏美 さんに深く感謝申し上げます．

本研究を行うにあたり，快く相談に乗っていただき，的確なご助言を頂きました，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博士後期過程 2 年の 村上 寛明 氏に深く感謝申し上げます．

また，本研究に関して多くのご助言を頂くとともに様々な面において親切なご助力，理解あるご助言を頂きました楠本研究室の皆様に心より感謝いたします．

最後に，本研究に至るまでに，講義，演習，実験等全ての科目でお世話になりました大阪大学大学院情報科学研究科，並びに大阪大学基礎工学部情報科学科の諸先生方に，この場を借りて心から御礼申し上げます．

参考文献

[1] S. W.L. Yip and T. Lam. A software maintenance survey. pp. 70–79, 1994.

[2] A April and A Abran. *Software Maintenance Management: Evaluation and Continuous Improvement*. Wiley-IEEE Computer Society Press, 2008.

[3] Thomas D. LaToza and Brad A. Myers. Developers ask reachability questions. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Vol.1*, pp. 185–194, 2010.

[4] H Gall, K Hajek, and M Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the 1st International Conference on Software Maintenance*, pp. 190–200, 1998.

[5] T Zimmermann, P Weisgerber, S Diehl, and A Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, pp. 563–572, 2004.

[6] D Schuler and T Zimmermann. Mining usage expertise from version archives. In *Proceedings of the 5th International Working Conference on Mining Software Repositories*, pp. 121–124, 2008.

[7] C.C. Williams and J.K. Hollingsworth. Automatic mining of source code repositories to improve bug finding techniques. *Software Engineering, IEEE Transactions on*, Vol. 31, No. 6, pp. 466–480, 2005.

[8] 松下誠. ソフトウェア工学の新潮流（１）リポジトリマイニング. ソフトウェアエンジアリング最前線 2009, pp. 21–24, 2009.

[9] 松村知子, 横森励士, 大杉直樹, 川口真司, 松下誠. ファイルの同時変更パターンと変更差分の分析による論理的結合関係の自動抽出. ソフトウェアシンポジウム 2005, pp. 104–112, 2005.

[10] M D'Ambros, M Lanza, and R Robbes. On the relationship between change coupling and software defects. In *Proceedings of the 16th Working Conference on Reverse Engineering*, pp. 135–144, 2009.

[11] Annie T. T. Ying, G C. Murphy, Raymond Ng, and Mark C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, Vol. 30, No. 9, pp. 574–586, 2004.

[12] J S Shirabad, T C. Lethbridge, and S Matwin. Mining the maintenance history of a legacy software system. In *Proceedings of the 19th International Conference on Software Maintenance*, pp. 95–104, 2003.

[13] Concurrent versions system. http://www.nongnu.org/cvs/.

[14] 森敏昭, 吉田寿夫. 心理学のためのデータ解析テクニカルブック. 北大路書房, 1990.

[15] 井上公基, 後藤英次郎, 長谷川徹也. 単純繰り返し作業による作業姿勢が作業負担に及ぼす影響. 森林利用学会誌, Vol. 13, No. 2, pp. 75–80, 1998.

[16] 泉田聡介, 植田泰士, 神谷年洋, 楠本真二, 井上克郎. ソフトウェア保守のための類似コード検索ツール. 電子情報通信学会論文誌 D, Vol. 86-D-I, No. 12, pp. 906–908, 2003.

[17] M Shonle, J Neddenriep, and W Griswold. Aspectbrowser for eclipse: A case study in plug-in retargeting. In *Proceedings of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 78–82, 2004.

[18] 石尾隆, 井上克郎, 佐々木裕介. コード片に共通した特性を自動抽出するソースコード閲覧ツールの試作. 情報処理学会研究報告. ソフトウェア工学研究会報告, Vol. 2010-SE-170, No. 9, pp. 1–8, 2010.

[19] R Robbes, D Pollet, and M Lanza. Logical coupling based on fine-grained change information. In *Proceedings of the 15th Working Conference on Reverse Engineering*, pp. 42–46, 2008.

[20] R Geiger, B Fluri, Harald C. Gall, and M Pinzger. Relation of code clones and change couplings. In *Proceedings of the 9th International Conference of Funtamental Approaches to Software Engineering*, pp. 411–425, 2006.

[21] N. Hanakawa. Visualization for software evolution based on logical coupling and module coupling. In *Proceedings of the 14th The Asia-Pacific Software Engineering Conference*, pp. 214–221, 2007.

[22] M Lanza and M Lungu. The evolution radar: Visualizing integrated logical coupling information. In *Proceedings of the 3rd International Workshop on Mining Software Repositories*, pp. 22–23, 2006.