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

松村 知子
大杉 直樹
奈良先端科学技術大学院大学
情報科学研究科
{tomoko-m, naoki-o}@is.naist.jp

横森 励士
南山大学数理情報学部
情報通信学科
yokomori@it.nanzan-u.ac.jp

川口 真司
松下 誠
大阪大学大学院情報科学研究科
{s-kawagt, matusita}@ist.osaka-u.ac.jp

要旨

長年にわたって機能の追加・変更，フォールト修正が行われてきた大規模なレガシーソフトウェアでは，あるコード片を変更する時に同時に別のコード片が変更されなければいけないといった論理的結合関係が多数存在する．開発者がこのような結合関係に気づかずにコードを変更すると，変更漏れやフォールト混入の危険性が高くなる．この問題を解決するために，ファイルの変更履歴から頻繁に同時変更されるファイルの組合せを分析し，抽出されるファイル間の結合関係をプログラムコード変更時の変更すべきファイルの推薦などに活用する研究が行われている．本研究では，抽出された結合関係の精度を向上させるため，ファイルの同時更新数のみでなく，同時更新時のファイルの変更内容(差分)を分析し，同時変更されるファイル間の結合関係の有無を判定する手法を提案する．提案手法により，ファイルの推薦などをより正確に行うことができ，保守作業の効率化が期待できる．

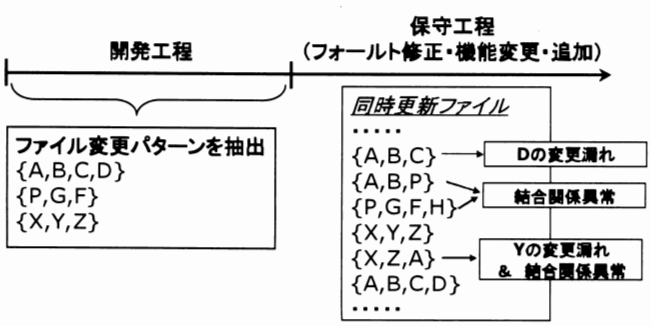

図 1 Logical Coupling 適用例

1. はじめに

長年にわたって機能の追加・変更，フォールト修正などが行われてきた大規模なレガシーソフトウェアでは，ファイル(モジュール)間の結合関係が増加し，システム構造が複雑化している[1]．このようなシステムでは，あるコード片の変更に対して，別のコード片も同時に変更しなければならないといった論理的結合関係(以下，Logical Coupling[2])の増加がみられる．このような結合関係の中には，設計段階で把握されていたものだけではなく開発や保守の製造段階で無計画に発生するものがある．これらは，ドキュメントに明記されていない上，一部の開発者・保守作業者のみが把握していて，時間の経過や作業者の入れ替わりによって潜在化してしまう．潜在化したこのような結合関係は，プログラムコード変更時の変更漏れやそれに伴うフォールト混入の原因となる危険性がある．

そのため，Logical Coupling をファイル変更履歴から抽出し，プログラムコードの変更時にその関係を作業者に提示することによって変更作業の完成度の向上・効率化を目指す研究が行われている[3][5][6]．論理的結合関係にあるファイルの変更は，同一作業者によって極めて近い時間内，もしくは同時に更新されることが多い．そのため，ファイル変更履歴上の変更者・変更時間によって， Logical Coupling の強いファイルの組合せを抽出することが可能になる．Logical Coupling の具体的な適用例を図 1に示す．これは,開発期間中のファイル変更履歴から同時変更される頻度の高いファイルの組合せ(以降，「ファイル変更パターン」と呼ぶ)を抽出し，あらかじめルールとして定めた上で，それ以降の工程でのファイル変更の漏れや異常を検出するという例である．しかし，実際の開発では，プログラムコード変更時に本来

あまり関係のない変更(例えば，複数の細かいバグの修正など)が同時に行われることも多いため，ファイルの同時変更回数から抽出される結合関係だけでは精度が低く，検出される変更漏れや異常が有効でない情報であることも多い．

本研究では，ファイル変更履歴上の同時更新回数だけでなく，各ファイルの変更内容(差分)の共通単語を調べることによって，Logical Coupling の精度を向上させる方法を提案する．本当に関連のある変更を行った場合，同時更新された各ファイルの差分，もしくは各ファイル変更パターンの変更毎の差分には共通した単語が含まれている可能性が高いと考えられる．この仮説に基づけば，差分中の共通した単語数を計測し，変更内容(差分)の似ている度合い(以降，類似度と呼ぶ)を取得し，この類似度を用いて変更内容の関連の強さを推定できる．類似度の高い，つまり論理的結合関係が存在するファイル変更パターンのみを精選することによって，Logical Coupling の精度向上が可能になると考えている．

提案手法では，文書内に出現する単語群に基いて文書間の類似度を測定する手法を，変更内容(差分)の単語分析に応用する．類似していない変更内容のデータを排除することで，結合関係の精度を向上させることができる．また，必要な情報はすべてCVSなどの構成管理データから取得可能で，プログラミング言語や開発手法に依存しないことから，現場への適用も容易であると考えられる．

以降，2 章では Logical Coupling を用いた従来研究と本研究との相違について，3 章で提案する手法について述べる．4 章では，計測事例について述べ，5 章でまとめと今後の課題について述べる．

2. Logical Coupling

2.1. Logical Coupling の定義と特長

ソフトウェアシステムには，プログラムコードのある部分が変更される場合に同時に変更されなければいけない部分がある,といった結合関係が存在する．例えば，ある機能変更時，既存関数を変更する必要が生じたときに，既存機能への影響を配慮して，その関数をコピーし，一部改変するといった変更を加えることがある．このようなケースでは，一方の関数を変更する場合，コピーした関数に同様の変更を加える必要がある場合が多い．

Logical Coupling[2]は，そのようなプログラムコード間に存在する関係を示している．このような結合関係は，上記のような関数単位に限らず，ファイル単位・モジュール単位などでも計測することが可能である．

ファイル単位での Logical Coupling は，システムの変更履歴から同時に変更される頻度の高いファイルパターンの分析によって容易に抽出することができる．これは，開発・保守が長期にわたる場合，このような結合関係を持つファイルパターンは頻繁に同時更新される，という前提に基づく．ファイル単位でのLogical Couplingは，構文解析が不要であるため，大規模なシステムでも容易に抽出することができ，プログラミング言語に依存しないため，プログラミング言語の混在するシステムや.ini，テキストファイル，プロパティファイルなども対象に含めることができる．また，コードの更新に応じて結合関係の動的な更新が容易である．

2.2. 従来研究

Ying らは，ファイル変更パターンを抽出する際の最適な最小支持度(同時更新回数)を，2 つの大規模のオープンソースプロジェクトを用いて，適合率・再現率を計測し評価している[3]. [3]では,ファイル変更パターンの抽出においてFP-Tree Algorithm[4]が利用されている．また，抽出されたファイル変更パターンから実際にルールとなる関係を手作業で取り出し，その内容からパターンの有効性を示している．

Shirabad らは，ファイル更新履歴とファイル変更時の問題報告書やプログラムコード，ソースコード内のコメントに含まれる単語の類似度などの情報を用いて，Logical Coupling の精度向上に寄与する情報を調査した[5]．その結果，ファイル変更時の問題報告書の情報が最も有効である，と結論付けている．しかし，これは評価したシステムにおいて，ファイルの変更と問題報告書が一元管理され，関連付けた参照が可能であることを前提としている．

Zimmermann らは，Logical Coupling の粒度をプログラムの要素単位に細分化することによって，より有効な変更推薦システムを実現した[6]．プログラムコード更新内容を，変数・関数などのプログラム要素に関連付け，その要素間の Logical Coupling をアソシエーションルール[1]の分析方法を用いてルールを抽出している．

従来研究は，いずれもファイル変更パターンをベース

[1] Association Rule:連関規則とも呼ばれる．ある事象が発生した時に，別な事象が発生する事後確率が高いような事象の組み合わせで，「属性 A を持つ対象は属性 B の生じる傾向がある」と一般的に記述される．

にしながら，結合関係の精度を向上させるために他の情報(ドキュメントやプログラムの構文情報)を用いることによって，ファイル単位での Logical Coupling の特長を相殺している．本研究では，この点を考慮し，CVSなどの構成管理システムのみから得られる情報を対象に，結合関係の精度の向上を図る．

3. 提案する手法

3.1. 提案手法の流れ

本研究では，ファイル変更履歴上の同時更新回数だけでなく，その変更内容(差分)の類似度を使って，Logical Coupling の精度を向上させる方法を提案する．具体的な手順は，次のとおりである(図 2 参照)．

(1) ファイル更新履歴から同時に更新されていると考えられる処理(以降，トランザクションと呼ぶ)をまとめる．各トランザクション中で更新されているファイルを分析し，同時更新回数が N 回以上のファイル変更パターン(図 3 参照)を取り出す．

(2) 各ファイル変更パターンで，同時変更トランザクション毎に変更内容を取得し，単語リストを作成する．

(3) (2)で作成した単語リストから類似度を計測し，各ファイル変更パターンに対してトランザクション毎，もしくはトランザクション間の結合関係の有無を判定する(3.4節参照)．

(4) 結合関係のあるトランザクション数を基に，各ファイル変更パターンの結合度を求め，一定の閾値をクリアするものを，精選されたファイル変更パターンとして抽出する．

以降，各手順の詳細を説明する．

3.2. ファイル変更パターンの抽出

ファイル変更パターンは，CVS 等の構成管理システムで取得できる変更履歴から抽出する．ファイル変更履歴には，変更したファイル名，日時，変更者，コメントなどが付加されている．これらの情報を用いて，変更者が意図的に 1 処理で変更しようとしたファイル群を推定することができる[7]．トランザクションはこれらの情報から作成するが，CVSの運用ルール(コメントの形式等)によって，正確に取り出す機構をあらかじめ設定することも可能である．

トランザクション毎のファイル群をアソシエーションルールの作成に用いるApriori Algorithm[8]を用いて，閾値Nを最小支持度(最小同時変更回数)とするファイル変更パターンを抽出する．図 3は最小同時変更回数を 14 としたときに抽出されたファイル変更パターンの例である．最適な N は，変更履歴数などシステムに依存する．

3.3. 変更差分からの単語リスト作成

3.2節で抽出された各ファイル変更パターンに対して，同時更新された場合の変更内容(差分)を取り出し，そこから単語リストを作成する．図 4 は単語リストの作成方法を示している．この図では，File α，β，γ，δ が 1 つのファイル変更パターンである．この 4 ファイルが同時変更されるトランザクション毎にそのトランザクションにおけるバージョンとその直前のバージョンの差分を取り出す．同時

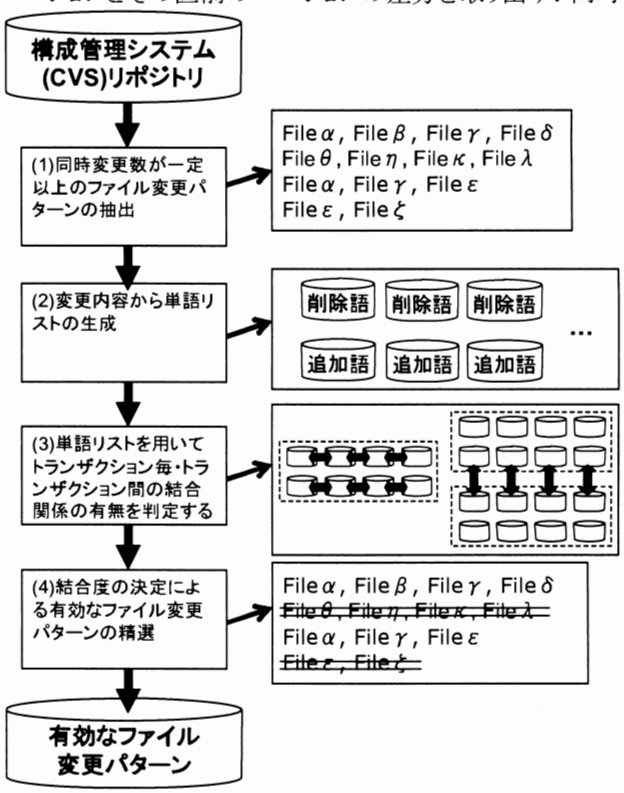

図 2 本研究の提案手法の流れ

```
ファイル数 = 5
同時更新回数 = 14
ファイル名 =
{bzflag/src/mediafile/ImageFile.h,
bzflag/src/mediafile/SGIImageFile.cxx,
bzflag/src/mediafile/SGIImageFile.h,
bzflag/src/mediafile/WaveAudioFile.cxx,
bzflag/src/mediafile/WaveAudioFile.h}
```

図 3 ファイル変更パターン例

更新されるトランザクション間に別の更新が存在する File α，γ，δは，矢印で対応付けられたファイル間での差分を使用する．図中の差分情報は CVS の diff コマンドで取得できる内容である．この差分を分析し，「削除語」「追加語」リストをファイル毎に作成する．変更された単語の場合，変更前の単語が「削除語」，変更後の単語が「追加語」にリストアップされる．

3.4. 単語リストの比較による結合関係の精選

3.3節で作成した単語リストを用いて，各トランザクション内のファイルの結合関係（以降，類似的結合関係と呼ぶ）と異なるトランザクション間の結合関係（以降，連携的結合関係と呼ぶ）の 2 種類の結合関係についてその有無を判定する．実際の関連する変更内容が，すべてこの 2 つの関係に集約できるかは未調査であるが，ここでは関連の強い変更内容には「同時に同じ機能について変更を加える場合」と「過去の変更に対して同じパターンの変更を加える場合」が最も頻度が高いと考え，この2つの関係に着目する．

3.4.1. 類似的結合関係

類似的結合関係とは，1 トランザクション内の更新ファイル群に同じ追加・変更を行わなければならない関係である．例としては，「特定の機能を持つ関数群には，同時に同じ処理を追加しなければならない」「ディレクトリ構造の変更などにより，参照するファイルのパス文字列をすべて同時に変更しなければならない」などが挙げられる．

これは，各ファイルのトランザクション毎の追加語と削除語の類似度を計算し，類似度が閾値 X を超える場合，そのトランザクションを有効な結合関係にある同時更新とみなす．類似度は，同時更新ファイル群の最も少ない削除もしくは追加単語数（以降，最小単語数と呼ぶ）に対する全ファイルに共通する削除もしくは追加単語数(以降，共通単語数と呼ぶ)の割合とする．

図5はFile α，β，γ，δの2つのトランザクションにおける単語リストを示している．アルファベット A～R はそれぞれ「追加語」「削除語」を示している．トランザクション1においては，「追加語」「削除語」が完全に一致しているので,類似度は 100%となり,有効な結合関係にある．一方，トランザクション2では「追加語」「削除語」に一部相違が見られる．このケースでは，類似度は 75%となる(最小単語数＝4，共通単語数＝3)．類似度の閾値 X=80%とすると，このトランザクションは有効な結合関係にはない，と判定される．

このような判定結果，各ファイル変更パターンに対して，有効な結合関係にあるトランザクション数を取得でき，これが一定数 Y を超える場合，ファイル変更パターンを有効であると判定する．ここで閾値となる X,Y は，システムの更新数，ファイル数に基いて決定する必要がある．

3.4.2. 連携的結合関係

連携的結合関係とは，異なるトランザクション間で変更される単語リストに共通する単語がある場合の関係である．例としては，「複数のファイル上にあるデータテーブルに追加・削除を行う場合，あるファイル上にあるテーブルの最大値を示すマクロ定義を必ず同時に更新しなければならない」「ある機能を複数ファイルにまたがって追加したが，仕様の変更が多く，抜本的な変更（全削除→新規に追加）が繰り返される」などが上げられる．ここでは，「ある機能が他のファイルに移動した」などの場合を考慮して，ファイルの枠を取り払い，各トランザクションの「追加語」「削除語」リストを作成し，そのリスト間で全単語数に占める共通単語数の割合がある閾値 R 以上の場合，そこに結合関係が存在するとみなす．

図 6 では，File α，β，γ，δの同時更新トランザクション3，4各々において「削除語」「追加語」の共通語は存在しないが，トランザクション3での追加語がトランザクション4で削除されて，別のファイルに追加されている．トランザクション3のすべて

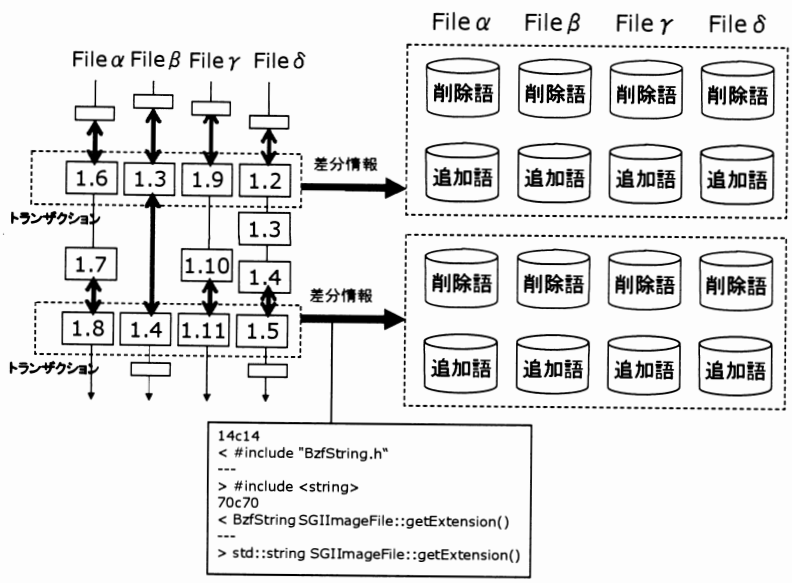

図 4 変更内容(差分)からの単語リストの作成

のファイルの「追加語」の 5/6 がトランザクション4の「削除語」および「追加語」に含まれる. このケースではトランザクション 3 の追加単語数が「最小単語数」となるので, いずれのケースも類似度約 83.3%となり, 閾値 R=80%とすると,このトランザクション間には連携的結合関係が存在する. 一方, 図 5 のトランザクション1, 2間には,「削除語」「追加語」ともに全く共通語が存在しないため,連携的結合関係はない.

このような判定結果, ファイル変更パターンの各トランザクションの組合せで有効な結合関係にあるものが一定数 S を超える場合, ファイル変更パターンは有効であると判定する. ここでの閾値 R,S は, 類似的結合関係の場合と同様, システムの更新数, ファイル数に基いて決定する必要がある.

4. 計測事例

4.1. 計測方法

本研究では, 3つのオープンソースプロジェクトのCVSレポジトリを用いて, 類似的結合関係・連携的結合関係についての計測を行った. これらのオープンソースプロジェクトの詳細を表 1 に示す. これらはいずれも

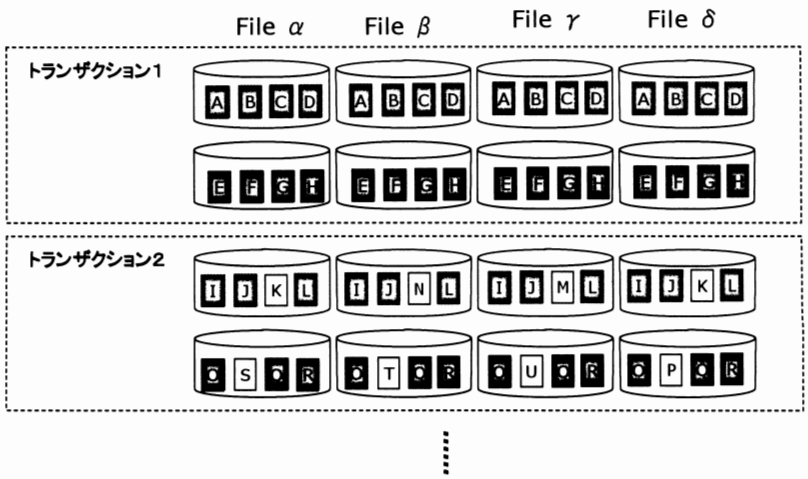

図 5 類似的結合関係例

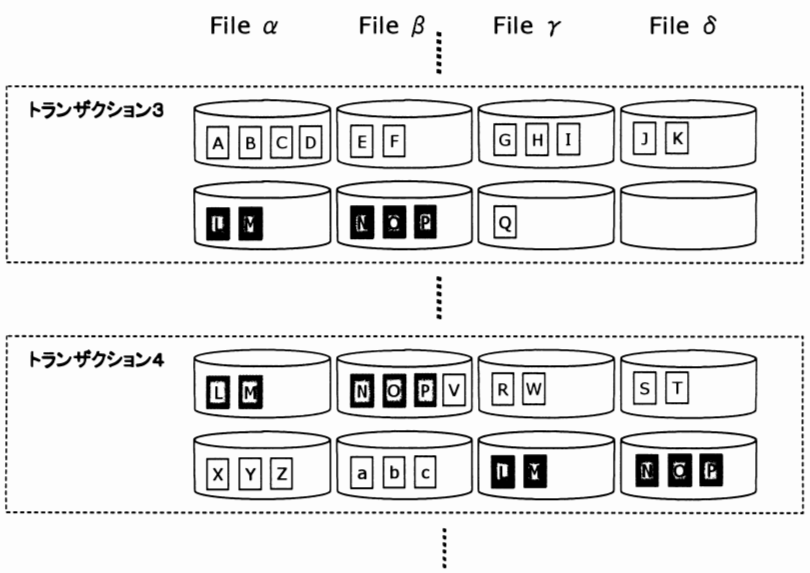

図 6 連携的結合関係例

表 1 測定事例に用いたオープンソースプロジェクト

プロジェクト名	ファイル数	トランザクション数	測定対象期間	
			測定開始日	測定終了日
Azureus	2709	6249	2003/7/10	2004/9/17
DooMLegacy	1562	1083	2000/2/22	2003/11/30
BZFlag	1796	7877	2000/3/5	2004/11/25

表 2 図 3 のファイル変更パターンに対する類似的結合関係

		全ファイルの共通単語数	bzflag/src/mediafile/ImageFile.h	bzflag/src/mediafile/SGIImageFile.	bzflag/src/mediafile/SGIImageFile.	bzflag/src/mediafile/WaveAudioFile	bzflag/src/mediafile/WaveAudioFile.	共通単語数/最小単語数
トランザクション1	追加語	1	1	1	1	1	1	1
	削除語	1	1	1	1	1	1	1
トランザクション2	追加語	2	2	2	2	2	2	1
	削除語	1	1	2	1	3	1	1
トランザクション3	追加語	1	1	1	1	1	1	1
	削除語	0	0	0	0	0	0	0
トランザクション4	追加語	1	1	1	1	1	1	1
	削除語	1	1	1	1	1	1	1
トランザクション5	追加語	2	2	2	2	2	2	1
	削除語	1	1	1	1	1	1	1
トランザクション6	追加語	13	13	13	13	13	13	1
	削除語	0	0	0	0	0	0	0
トランザクション7	追加語	5	5	5	5	5	5	1
	削除語	5	5	5	5	5	5	1
トランザクション8	追加語	1	1	1	1	1	1	1
	削除語	1	1	1	1	1	1	1
トランザクション9	追加語	1	1	1	1	1	1	1
	削除語	1	1	1	1	1	1	1
トランザクション10	追加語	1	1	1	1	1	1	1
	削除語	1	1	1	1	1	1	1

```
Index: bzflag/src/mediafile/ImageFile.h
diff -r1.2 -r1.3
25c25
<         // static BzfString getExtension();
---
>         // static std::string getExtension();
```

```
Index: bzflag/src/mediafile/SGIImageFile.h
diff -r1.2 -r1.3
23c23
<         static BzfString      getExtension();
---
>         static std::string    getExtension();
```

```
Index: bzflag/src/mediafile/WaveAudioFile.h
diff -r1.2 -r1.3
23c23
<         static BzfString      getExtension();
---
>         static std::string    getExtension();
```

```
Index: bzflag/src/mediafile/SGIImageFile.cxx
diff -r1.2 -r1.3
14c14
< #include "BzfString.h"
---
> #include <string>
70c70
< BzfString  SGIImageFile::getExtension()
---
> std::string  SGIImageFile::getExtension()
```

```
Index: bzflag/src/mediafile/WaveAudioFile.cxx
diff -r1.2 -r1.3
14d13
< #include "BzfString.h"
108c107
< BzfString   WaveAudioFile::getExtension()
---
> std::string   WaveAudioFile::getExtension()
```

図 7 表 1 のトランザクション 2 の削除語の相違点

SourceForge.net[2]からアクセス可能で，CVS の変更履歴の取り出しは， EASE プロジェクト[9]で開発しているEPM ツールの機能を使用した．表 1 のファイル数は CVS 履歴に出現するファイル数である．トランザクションデータ(数)は CVS 履歴の変更時間・変更者・変更時コメントから専用プログラムを作成して取り出した．測定対象期間は，対象とした CVS 履歴データの期間であって，当該プロジェクトの開発開始・終了日とは一致しない．また,ファイル変更パターンの取り出し，単語リストの作成も，専用のプログラムを作成して行った．変更差分の抽出は，CVS の diff コマンドを用いた．

[2] オープンソース開発用 Web サイト
http://sourceforge.net/

表 3 連携的結合関係計測結果例

トランザクション番号①	追加語/削除語(ADD/DELETE)	トランザクション番号②	追加語/削除語(ADD/DELETE)	共通単語数	トランザクション①語数	トランザクション②語数	共通単語数/最小単語数
6	ADD	15	ADD	5	6	37	0.833333
7	ADD	8	DELETE	2	6	2	1
8	ADD	9	DELETE	2	19	2	1
9	ADD	10	DELETE	2	7	2	1
10	ADD	11	DELETE	3	9	3	1
12	ADD	14	DELETE	12	13	34	0.923077
12	ADD	15	ADD	12	13	37	0.923077
13	ADD	14	DELETE	19	19	34	1
13	ADD	15	ADD	19	19	37	1
14	ADD	15	DELETE	2	2	2	1
14	DELETE	15	ADD	34	34	37	1

表 4 各プロジェクトでの計測結果

プロジェクト名	最小同時更新回数	総パターン数	有効パターン数	
Azureus	14	239	類似的	220
			連携的	1
			合計(重複なし)	221
DooMLegacy	9	268	類似的	31
			連携的	6
			合計(重複なし)	37
BZFlag	14	292	類似的	164
			連携的	85
			合計(重複なし)	207

4.2. 結果例

4.2.1. 類似的結合関係

表2は図3のファイル変更パターンに関する単語リストの抽出結果である. 1列目が各トランザクションにおける全ファイルに共通な単語数, 次が各ファイルの追加語数・削除語数, 最後が各トランザクションでの最小単語数分の共通単語数である. このファイル変更パターンでは, 10回のトランザクションデータのうち2回目のトランザクションの削除語以外はすべて一致しており, 単語数が 0 の場合を除いて類似度はすべて 1(すなわち 100%)である. また, 2 回目のトランザクションの変更内容の差分を図 7 に示す. 相違は, C++とヘッダーファイルによって, include 文が異なることから来る. 従って, この相違は結合関係を弱めるものではない.

この結果から, このファイル変更パターンは, 有効な論理的結合関係(すなわち, Logical Coupling)を持つ, と判定することができる.

4.2.2. 連携的結合関係

表 3 に 1 つのファイル変更パターンの連携的結合関係の計測結果を示す.

このファイル変更パターンは, 2 つのファイルからなり 17 回同時更新されているが, 各トランザクションのファイル間の類似的結合関係はほとんど見られない(付録 A 参照). 一方, トランザクション間の類似度を見ると, 11 個の組合せで, 共通単語数の割合が 80%を超えている(表 3 は 80%を超える組合せのみを示している). 図 8 は, 表 3 の 3 行目の組合せの差分内容である. これらの差分は, Protocol.h の const で定義されている変数の追加やその設定値の変更時に, version.h の BZ_PROTO_VERSION 及び BZ_REV の値を更新しなければいけない, という論理的結合関係(すなわち, Logical Coupling)が存在することを示している. このような関係は, 類似的結合関係では抽出できない.

4.3. 各プロジェクトの計測結果

表 4 に各オープンソースプロジェクトにおける計測結果を示す. 総パターン数は, CVS 履歴情報のみの分析から最小同時更新回数以上の同時変更回数を持つファイル変更パターンを取り出した結果である(最小同時更新回数は, 24 時間内にパターン抽出が完了した最小の回数を用いている). 有効パターンは, 各結合関係の類似度の閾値を80%とし, かつ8個以上の有効な組合せが存在したファイル変更パターン数である. 類似的・連携

的結合関係でそれぞれ取り出した後，両方の関係で有効なパターンの重複を除いた有効パターン数が合計になっている．

4.4. 考察

今回の計測では，本手法の有効性までは示すことはできなかったが，差分の単語群の類似度から有効な Logical Coupling を抽出可能であることを確認することができた．

また，論理的結合関係抽出の過程で，いくつかの課題も確認できた．

- 類似的結合関係で，予約語やコメントに多く含まれる単語(英語の場合，the や of など)によって，類似度が上がってしまう

```
CommitDate = 2004/06/22 18:27:41      Owner = trepan
Index: bzflag/include/Protocol.h
===================================================================
RCS file: /cvsroot/bzflag/bzflag/include/Protocol.h,v
retrieving revision 1.58
retrieving revision 1.59
diff -r1.58 -r1.59
110a111,112
> const uint16_t              WorldCodeWeapon = 0x7765;              // 'we'
> const uint16_t              WorldCodeZone = 0x7A6e;                // 'zn'
121a124,125
> const uint16_t              WorldCodeWeaponSize = 24;  // basic size,
> const uint16_t              WorldCodeZoneSize = 34;    // basic size,
Index: bzflag/include/version.h
===================================================================
RCS file: /cvsroot/bzflag/bzflag/include/version.h,v
retrieving revision 1.45
retrieving revision 1.46
diff -r1.45 -r1.46
29c29
< #define BZ_PROTO_VERSION        "1114"
---
> #define BZ_PROTO_VERSION        "1115"
41c41
< #define BZ_REV              4
---
> #define BZ_REV              5
```

```
CommitDate = 2004/06/27 08:55:31      Owner = trepan
Index: bzflag/include/Protocol.h
===================================================================
RCS file: /cvsroot/bzflag/bzflag/include/Protocol.h,v
retrieving revision 1.59
retrieving revision 1.60
diff -r1.59 -r1.60
82a83
> const uint16_t              MsgPlayerUpdateSmall = 0x7073;         // 'ps'
Index: bzflag/include/version.h
===================================================================
RCS file: /cvsroot/bzflag/bzflag/include/version.h,v
retrieving revision 1.46
retrieving revision 1.47
diff -r1.46 -r1.47
29c29
< #define BZ_PROTO_VERSION        "1115"
---
> #define BZ_PROTO_VERSION        "1116"
41c41
< #define BZ_REV              5
---
> #define BZ_REV              6
```

図 8 表 3 の 3 行目の差分内容

- 連携的結合関係で，1 つのファイルの変更内容が桁違いに多いと，他のファイルの変更が全く関連なくても，類似度が上がってしまう

また，表 4 のとおり，同じ閾値を用いてもプロジェクト毎に精選率は大きく異なり，実際に抽出される論理的結合関係の種類も一定の割合ではないことが判明した．原因として，ファイルの更新ルールやプロジェクトの性質の違いなどが考えられる．例えば，Azureus というプロジェクトは，多言語対応をしているため各言語に対応する属性ファイルを持ち，これらのファイルへの頻繁な同時変更が類似的結合関係を持つパターン数に反映している．

これらの結果から，類似度計測方法の改善や，結合関係の分類とそれに基づく計測対象の拡張などが必要と考えられる．

5. まとめと今後の課題

本研究では，ファイル間の Logical Coupling の抽出を，構成管理システムの情報のみを用いてより精度を向上させる方法を提案した．ファイル間の Logical Coupling は，大規模なレガシーソフトウェアの保守の際に変更漏れやそれに伴うフォールトの混入に有効な情報として注目はされているが，精度の低さによって実用化が困難であった．精度を向上させるために，従来研究では構文解析情報や付随する他のドキュメント情報の利用が提案されてきたが，本研究では構成管理システムから取得できるファイルの変更履歴，および各変更時の差分のみを用いた結合精度の向上方法を提案した．システム開発における構成管理システムの活用が普及する中，高精度の Logical Coupling 抽出が構成管理システムからの情報のみで可能になると，ソフトウェア開発現場への導入も容易になると考えられる．

提案方法を評価するために，3 つのオープンソースプロジェクトに適用した結果，論理的結合関係を抽出することが可能であることが確認できた．

今後は，実用的なシステムの構築のために，以下の作業が必要になると考えている．

- 変更内容の分類による結合関係の拡張(類似的・連携的結合関係以外の結合関係の抽出)
- 差分中の単語に基づく類似度計測方法の確立(類似度測定方法としては，LSA(Latent Semantic Analysis)[10]の適用なども考えられる．)
- 具体的な最小変更回数など閾値の決定，もしくは決定方法の確立
- 抽出された高精度の Logical Coupling を使ったファイル変更パターンによる変更推薦・フォールト検出

システムの開発とシステムの有効性評価

謝辞

本研究の一部は文部科学省「e-Society 基盤ソフトウェアの総合開発」の委託に基づいて行われた．本研究にあたり多くのご協力を頂いた EASE プロジェクト関係諸氏に感謝します．

参考文献

[1] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, “Does code decay? Assessing the evidence from change management data,” IEEE Trans. on Software Engineering, Vol. 27, No. 1, pp. 1 - 12, Jan. 2001.

[2] H. Gall, M. Jazayeri, J. Krajewski: ”CVS Release History Data for Detecting Logical Couplings”, Proc. 6th International Workshop on Principals of Software Evolution (IWPSE'03) pp.13-23, Sep. 2003. Helsinki, Finland

[3] A. T. T. Ying, G. C. Murphy, R. Ng, M. C. Chu-Carroll: ” Predicting Source Code Changes by Mining Change History ”, IEEE transactions on Software Engineering, pp.574-586, Vol.30, No.9, Sep. 2004.

[4] J. Han, J. Pei, Y. Yin: ” Mining Frequent Patterns without Candidate Generation ”, Proc. Int'l Conf. Management of Data, W. Chen et al., eds., pp.1-12, 2000.

[5] J. S. Shirabad, T. C. Lethbridge, S. Matwin : ” Mining the Maintenance History of a Legacy Software System”, Proc. International Conference on Software Maintenance (ICSM'03) pp.95-104, Sep. 2003. Amsterdam, The Netherlands

[6] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, Andreas Zeller: ” Mining Version Histories to Guide Software Changes ”, Proc. 26th International Conference on Software Engineering (ICSE'04) May, 2004 Edinburgh, Scotland, United Kingdom

[7] Thomas Zimmermann, Peter Weißgerber: ”Preprocessing CVS Data for Fine-Grained Analysis”, Proc. International Workshop on Mining Software Repositories (MSR), Edinburgh, Scotland, UK, May 2004

[8] R. Agrawal, R. Srikant: ”Fast algorithm for mining association rules”, Proc. 20th Very Large Data Bases Conference(VLDB), pp.487-499. Morgan Kaufmann, 1994

[9] EASE(Empirical Approach to Software Engineering)Project, http://www.empirical.jp/project/index.html

[10] Shinji Kawaguchi, Pankaj K. Garg, Makoto Matsushita, Katsuro Inoue: "MUDABlue: An Automatic Categorization System for Open Source Repositories", The 11th Asia-Pacific Software Engineering Conference(APSEC2004), 2004.

付録

A： 4.2.2節での計測データ(トランザクション毎の類似度)

		全ファイルの共通単語数	bzflag/include/Protocol.h	bzflag/include/version.h	共通単語数/最小単語数
トランザクション1	追加語	13	13	18	1
	削除語	0	0	0	0
トランザクション2	追加語	1	1	1	1
	削除語	1	1	1	1
トランザクション3	追加語	0	1	2	0
	削除語	0	1	2	0
トランザクション4	追加語	1	1	1	1
	削除語	1	1	1	1
トランザクション5	追加語	1	1	1	1
	削除語	1	1	1	1
トランザクション6	追加語	0	5	1	0
	削除語	0	0	1	0
トランザクション7	追加語	0	4	2	0
	削除語	0	0	2	0
トランザクション8	追加語	0	17	2	0
	削除語	0	0	2	0
トランザクション9	追加語	0	5	2	0
	削除語	0	0	2	0
トランザクション10	追加語	0	7	2	0
	削除語	0	0	2	0
トランザクション11	追加語	0	1	2	0
	削除語	0	1	2	0
トランザクション12	追加語	0	12	1	0
	削除語	0	0	1	0
トランザクション13	追加語	0	17	2	0
	削除語	0	0	2	0
トランザクション14	追加語	0	0	2	0
	削除語	0	32	2	0
トランザクション15	追加語	0	35	2	0
	削除語	0	0	2	0
トランザクション16	追加語	0	3	2	0
	削除語	0	3	2	0
トランザクション17	追加語	0	8	2	0
	削除語	0	0	2	0