社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－SE－151（7）
2006／3／23

ソースコードの差分を用いた関数呼び出しパターン抽出手法の提案

中山 崇[†]　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

一般にソフトウェア部品の利用法の理解には部品に附属するドキュメント等が用いられるが，それらが無い部品では利用法の学習が困難である．こうした部品の利用法理解を支援する手法にコーディングパターンの抽出が挙げられる．コーディングパターンとはソースコードに頻出する構造のよく似たコード記述であり，開発者はこれを閲覧する事で実現したい機能に必要な処理を学習できる．しかし，既存のパターン抽出手法では関連の無い機能が 1 つのパターンにまとめられてしまうため，正確な利用法の学習ができないという問題点が存在した．本稿では，この問題点を解決するため，ソースコードの差分から個別の機能に限定した関数呼び出しパターンを抽出する手法を提案する．

Extracting Function Call Patterns from Source Code Deltas

Takashi Nakayama[†]　Makoto Matsushita[†]　Katsuro Inoue[†]

[†] Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

Generally, we learn software components usage from documents attached to them. But it is difficult to learn usage if the components have no documents. Extracting coding patterns is a method to support comprehending usage of components with no documents. A coding pattern is a code description which appears frequently in source codes, and developers can learn a series of a process by viewing it. Existing pattern extracting methods, however, combine several features into one pattern, thus developers cannot comprehend correct usage. We propose a method to extract function call patterns from source code deltas in order to improve the problem.

1 まえがき

近年のインターネットの普及により，WWW を通じて大量のソフトウェア資産が容易に入手可能となった．このようなソフトウェア資産を新たなソフトウェアの部品として開発を進めることをソフトウェア再利用と呼び，高品質なソフトウェアを一定期間内に効率良く開発するための代表的なソフトウェア工学技術の一つとして知られている [3, 6]．実際に再利用を行うには，まず再利用するソフトウェア部品の利用法を理解する必要がある．一般に，ソフトウェア部品の利用法の理解はそれに附属するドキュメント等をもとに行うが，それらが附属していないソフトウェア部品では利用法に関する情報が得られないため，部品の利用法を学習するのは困難であった．

こうした問題を解決する手法としてコーディングパターンの抽出手法がある．コーディングパターンとはソースコードに頻繁に現れる構造のよく似たコード記述である．開発者はこれを閲覧する事でソフトウェア部品の再利用に必要な処理を学習できる．しかし，既存のパターン抽出手法は，同じ箇所で呼び出される関数には関連があるとする仮定を持つことから，特に関連の無いコード記述同士が 1 つのパターンとして抽出されるという問題点が存在するため，部品の再利用が困難になっていた．

そこで本稿では，版管理システムに蓄積されているソースコードの差分から，個別の機能に限定したソフトウェア部品の利用法を理解するための関数呼び出しパターン抽出手法を提案する．この手法は，版管理システムへのソースコードの変更内容の保存が一般に個別の機能毎に行なわれているという仮定にもとづき，ソースコードの差分に含まれる関数利用実績に共通して現れるコード記述を関数呼び出しパターンとして抽出する．

以降，2 節では背景となるソフトウェア部品の再利用とコーディングパターンについて，3 節で版管理システムについて述べる．次に，4 節では本稿で提案する関数呼び出しパターン抽出手法について説明し，5 節でその実装について述べる．そして 6 節で評価を行ない，7 節でまとめと今後の課題について述べる．

2 ソフトウェア再利用

本節では，ソフトウェア部品の利用法と，その理解を支援するコーディングパターンについて説明する．

2.1 ソフトウェア部品の利用法

ソフトウェア部品を再利用するには，まず再利用する部品の利用法を学習する必要がある．例えば，再利用する部品が関数である場合は，実現したい処理に必要な他の関数群やそれらの呼び出し順，各関数の引数や返り値の扱い方などを学習しなければ望んだ機能を実現できない．

一般に，部品の利用法の学習はそれに附属するドキュメントやサンプルプログラムをもとに行われる．開発者はドキュメントを読む事でその部品への理解を深め，サンプルプログラムを見ること事で基本的なコーディングの仕方を学習する．

商用のソフトウェア部品等にはこのようなドキュメントが附属しているため，そこから利用法の学習ができる．しかし，一般に入手できる多くのソフトウェアではこれらが附属していないことも多い．そのような部品では利用法の学習が困難となり，再利用が難しくなる．

2.2 コーディングパターン

ドキュメント等が附属していない部品の利用法の学習を支援するために，再利用対象の部品を利用しているソースコードからその部品の利用例を取り出す事で利用法の理解に役立てるという研究が過去になされている [8, 11]．その中の一つに，再利用対象の部品を利用しているソースコードからコーディングパターンと呼ばれるものを抽出し，それを開発者に提示する事で利用法の学習を支援するという研究が行なわれている [2]．コーディングパターンとはソースコードに頻出する構造のよく似たコード記述である．コーディングパターンを閲覧する事で，開発者は実現したい処理に必要な関数群とその呼び出し順を，そしてパターンを利用しているソースコードを閲覧する事で関数への引数の与え方や返り値の扱い方などを学習できる．

既存のコーディングパターン抽出手法では，同じ箇所に現れる関数呼び出しは一連の機能を実現するためのものであると見なす，という特徴をもつ．構造化手法をはじめ，様々な開発技法において，関連するコードを 1 つの単位にまとめる事の

重要性が強調されている [4, 7].

しかし，この仮定のもとでコーディングパターンを抽出すると，実際には関連のない関数間に関連があると誤判定する場合も少なからず存在する．例えば図 1 のように，ソースコード内の幾つかの箇所で，計算処理を行うコード記述とデータ出力を行うコード記述が続けて書かれていたとする．前述の仮定のもとでコーディングパターンを抽出すると，この 2 つの処理は同じ箇所で呼び出されているため，関連があるコード記述として 1 つのコーディングパターンにまとめられる．その後，他の開発者がこのソフトウェアの計算処理部分を再利用する事になったとする．この開発者は計算処理部の利用法を学習するために関連するコーディングパターンを閲覧するが，そこには計算処理に関連の無いデータ出力処理が混在しているため，これを閲覧した開発者は計算処理を正しくコーディングできないことになる．

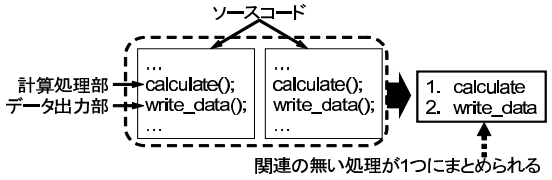

図 1: 関連の無い処理を含むパターン

本稿ではこの問題点を解決するために，版管理システム内のソースコードの差分から個別の機能に限定したコーディングパターンを抽出する手法を提案する．

3 版管理システム

本節では，まず版管理システムとその役割について説明した後，ソースコードを更新する際に見られる一般的な傾向について述べる．

3.1 版管理システム

版管理システムとは，プロダクトの開発履歴を保存，提供する機構である．ソースコードやリソースといった各プロダクトの履歴データは，リポジトリと呼ばれるデータ格納庫に蓄積される．その内部では，プロダクトのある時点における状態であるリビジョンを単位として管理する．1 つのリビジョンには，ソースコードやリソースなどの実データと，作成日時やログメッセージなどの属性データが格納されている．開発者はソースコードに対して変更を加える度に，その変更内容をチェックインという操作によって版管理システムに保存する事で開発作業を進めていく．

3.2 ソースコード更新時の傾向

版管理システムを用いた開発では，一度のチェックインで変更されるソースコードはある 1 つの機能についてのソースコードである事が一般的である．

Gall らは版管理システム内の開発履歴から，ソースコードからは分からない，Logical Couplings と呼ばれる依存関係を抽出し，ソフトウェアの理解や保守に役立てる研究を行なった [5]．Zimmermann らは開発履歴からソースコードの変更ルールの抽出を行ない，変更点の予測や不完全な変更による不具合を防ぐ為のシステムを開発している [12].

これらの研究では，一度のチェックインで扱うソースコードが共通の機能を実装しているという事を仮定している事が共通している．これらの研究が良い結果を残している事を考えると，実際のソフトウェア開発において一度のチェックインで扱うソースコードは個別の機能に関わるものであるという仮定は妥当であると考える事ができる．

4 ソースコードの差分を用いた関数呼び出しパターン抽出手法

本節では，版管理システム内のソースコードの差分から，個別の機能に限定したコーディングパターンを抽出する手法について述べる．

本手法では C 言語でかかれたソースコードを対象としてパターンの抽出を行う．また，本手法では部品の利用法を関数の利用関係という面から捉えているため，コーディングパターンという語句ではなく，関数呼び出しパターンという語句を用いる．

本手法は図 2 のように，3 つのステップに分けられる．まず，ソースコードの差分からソースコードの特徴を取得する．次に，取得されたソースコードの特徴から特徴シーケンスを生成する．最後に，生成された全特徴シーケンスに対して sequential pattern mining と呼ばれる手法を適用する事で関数呼び出しパターンを抽出する．以下，これら 3 つのステップについて説明する．

4.1 ソースコードの特徴の取得

始めに，版管理システム内のソースコードの差分からソースコードの特徴を取得する．ソースコードの特徴とは，関数の利用例を構成する要素であ

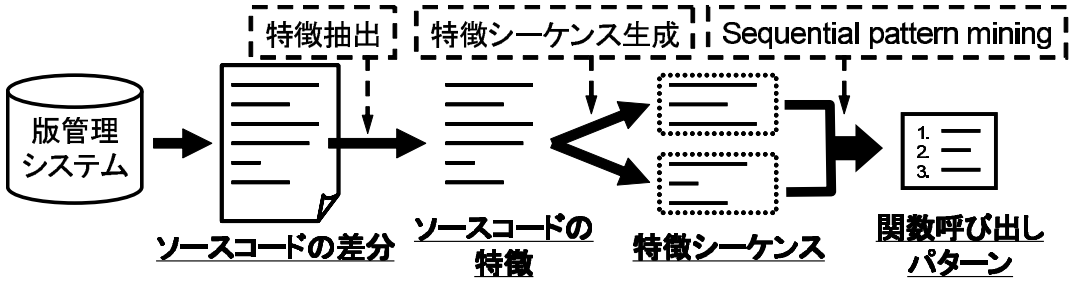

図 2: 関数呼び出しパターン抽出の流れ

り，本手法では関数呼び出し，及び条件文，繰り返し文の開始，終了位置をソースコードの特徴とする．以下では，条件文の開始，終了位置と繰返し文の開始，終了位置といったソースコードの特徴を制御文要素と呼ぶ．

ソースコードの特徴の取得は，全ての隣接するリビジョン間のソースコードの差分の内，新しいリビジョンにおいて追加，編集された行のみから行われる．ソースコードの差分のみから取得するのは，3.2 節で述べたように，ソースコードの差分に注目する事で個別の機能に限定したソースコードを取得できるからである．

次に，後の計算にかかるコストを抑えるために，不要なソースコードの特徴を除去する．本手法では不要な要素を標準関数呼び出しと，同じ構造の制御文要素のみの繰り返しとする．標準関数はドキュメント等が充実していることから，本手法を用いて利用法を理解する必要が無いため，除去の対象とした．また，制御文要素のみが同じ形で繰り返されるものも，関数呼び出しパターンに寄与しているとは考えにくいため，除去の対象とした．

4.2 特徴シーケンスの生成

次に，取得したソースコードの特徴から特徴シーケンスを生成する．特徴シーケンスとは関数の利用実績を抽象化したものであり，具体的には，1 つの関数定義内におけるソースコードの特徴のリストである．ただし，本手法ではソースコードの差分から関数呼び出しパターンを抽出するため，特徴シーケンスを構成する各要素は編集後のソースコードにおいて追加，編集された行にあるもののみに限定される．

図 3 は，追加，編集が起こった行に存在するソースコードの特徴から特徴シーケンスを生成する様子を表している．この操作を全てのソースコードの全てのリビジョン，全ての関数定義に対して行う事で，全ての特徴シーケンスの生成が出来る．このとき，関数呼び出しを一つも含まないような特徴シーケンスは関数呼び出しパターンの抽出に用いる事ができないので除去する．

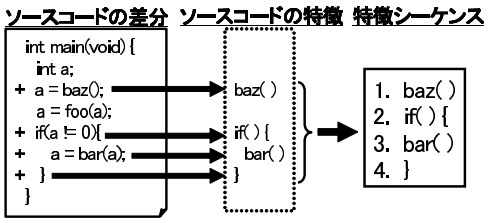

図 3: 差分からの特徴シーケンス抽出の流れ

4.3 Sequential pattern mining を用いたパターン抽出

最後に，生成した全ての特徴シーケンスを対象にして sequential pattern mining と呼ばれる手法を適用する事で，関数呼び出しパターンの抽出を行う．Sequential pattern mining とは与えられた複数のリストから，ユーザが指定した閾値以上の頻度で共通して現れる部分リストを求める手法である [1]．4.2 節で述べたように，特徴シーケンスは関数の利用実績を表すので，これらの間に共通するシーケンスを抽出する事で，関数呼び出しパターンとしている．

Sequential pattern mining を行うアルゴリズムはいくつか存在するが，本研究では一般的に用いられることが多い PrefixSpan[9] を採用した．PrefixSpan はサポート値計算と射影と呼ばれる操作が中心となるアルゴリズムである．サポート値計算とは，与えられたシーケンス群を構成する各要素のサポート値を求める操作である．サポート値とは対象の要素を含むシーケンスの数である．射影とは，全てのシーケンスから特定の要素からの接尾辞を取

り出す操作である．

PrefixSpan アルゴリズムでは図 4 のように，射影したシーケンス群の各要素についてサポート値を計算し，閾値以上の各要素について更に射影を行うといった事を繰り返す事で sequential pattern mining を行う．

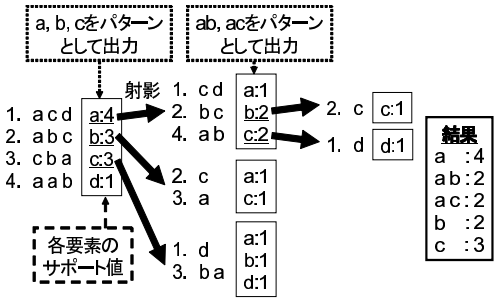

図 4: PrefixSpan によるマイニング

しかし，単純に特徴シーケンスに対して sequential pattern mining を適用するだけでは，有用な関数呼び出しパターンと共に，非常に多くの部品利用法の理解に用いることのできないパターンが抽出される．具体的には，制御文要素の対応がとれていないパターン，制御文要素の数がパターン全体の 3 分の 2 を越えるパターン，そして関数呼び出し要素を 1 個以下しか持たないパターンが挙げられる．

このような関数呼び出しパターンは利用法の理解に用いることができないどころか，パターン抽出にかかるコストが大きくなり，抽出されたパターンの検索を困難にする．そこで，本手法ではこれらの不要な関数呼び出しパターンを以下のようにして除去する．

マイニング時における除去:

PrefixSpan では，閾値以上のサポート値を持つ要素に対して射影を行なってパターンを出力するが，その結果制御文要素の対応がとれていないパターンが出力される場合はその射影を実行しないようにする．この結果，不要なパターンが出力されないだけでなく，PrefixSpan アルゴリズムの探索空間が大幅に削減され，計算時間と消費メモリ量が軽減される．

また，1 つのパターンに同じ制御文要素を 2 つ以上持たせない，制御文要素を連続させない，同じ関数呼び出し要素を連続させないという制限も射影操作に加える．

パターン抽出後における除去:

全てのパターン抽出が終った後に，各パターンに対して条件判断を行ない，制御文要素の数がそのパターン全体の要素数の 3 分の 2 を越えている，もしくは関数呼び出し要素が 1 個以下であるパターンを除去する．

5 実装

本節では，4 節で述べたソースコードの差分からの関数呼び出しパターン抽出手法を実装したシステムについて述べる．

本システムは大まかに 3 つのサブシステムに分ける事ができる．1 つ目は，4.1 節と 4.2 節で説明した処理を行なう特徴シーケンス生成部である．2 つ目は，4.3 節で説明した処理を行なう関数呼び出しパターン抽出部である．3 つ目は，抽出した関数呼び出しパターンと，それを実現しているソースコードを閲覧する為のパターンブラウザ部である．システム全体の概要を図 5 に示す．

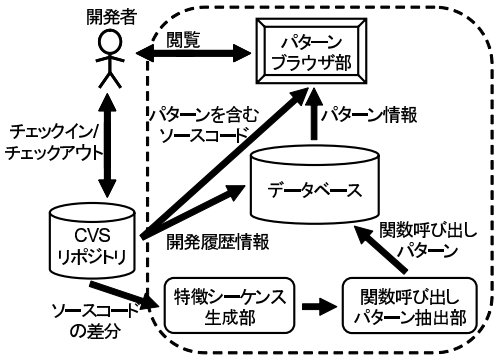

図 5: システムの概要

以下では本システムの 3 つのサブシステムについて説明する．

5.1 特徴シーケンス生成部

特徴シーケンス生成部では，版管理システム CVS から全てのリビジョンにおけるソースコードの差分を取得し，4.1 節と 4.2 節で述べた手法にしたがって特徴シーケンスを生成する．ただし，対象のリビジョンが CVS によるマージ操作によって自動的に更新されたものと判断した場合は特徴シーケンスの生成を行なわない．ここでマージ動作とは，ブランチ上に蓄積された更新を他のブランチに適用する操作である．この操作では一般に複数の機能が同時に更新されるので，個別の機能に関する関数呼び出しパターンの取得には向かない．そのため特徴シーケンスの生成を行わない．

5.2 関数呼び出しパターン抽出部

関数呼び出しパターン抽出部では，特徴シーケンス生成部によって生成された特徴シーケンスから sequential pattern mining を用いて関数呼び出しパターンを抽出する．

Sequential pattern mining の対象となる特徴シーケンスは，計算時間と消費メモリ量削減の為，全て整数値の配列に変換した上で計算を行なっている．

抽出された関数呼び出しパターンは，同じ関数呼び出し要素を持つもの同士でカテゴリ分けされる．これによって，同じ関数群が少し違う呼び出し方をされている状況を把握しやすくした．また，パターンを一行の文字列として簡単に表現するサマリや，各パターンがどのリビジョンのどの箇所に存在したかという情報も抽出し，データベースへと格納する．これらの情報は 5.3 節で述べる関数呼び出しパターンブラウザ部によって用いられる．

5.3 関数呼び出しパターンブラウザ部

関数呼び出しパターンブラウザ部は，抽出された関数呼び出しパターンとそれを実現しているソースコードを提示する事で開発者による部品利用法の理解を支援する．図 6 に関数呼び出しパターンブラウザのスクリーンショットを示す．

画面上部のパターンリストと左下部のリビジョンリストから項目を選択する事で，開発者は関数呼び出しパターンとそれを利用しているソースコードを同時に閲覧する事ができる．右下部のソースコードのうち，行頭に'+' が付いているものは追加，編集が起こった行であり，背景に色が付いた行はパターンに該当する行である．また，パターン検索画面からユーザが指定した関数呼び出しを含むパターンを検索する事もできる．

6 評価

本節では，提案手法が実際に部品の利用法理解に有用なパターンを抽出できるかどうか，そして，既存手法の問題点が改善されているかどうかについて評価する．

評価に際して，関数呼び出しパターンを抽出する対象のソフトウェアプロジェクトには The Golem X11 Window Manager[10] を選択した．Golem の CVS リポジトリ中のソースコードの差分から本手法を用いて関数呼び出しパターンを抽出したところ，計算時間は約 1 分半，抽出された関数呼び出しパターン数は 74 個，パターンのカテゴリ数は 45 個となった．

6.1 関数呼び出しパターンの評価

ここでは，抽出された関数呼び出しパターンを実際に確認する事で，それらの部品の利用法理解における有用性を評価する．

図 7 は，提案するシステムによって抽出されたクライアントのリサイズを行う際の関数呼び出しパターンである．パターンを詳しく見ると，`client_sizeframe` 関数を用いてクライアントのリサイズを行なった後，`client` 変数の状態から条件分岐を行ない，条件に合致しているならば `action_send_config` 関数を用いてリサイズ結果を X サーバに送信していることが分かる．このパターンから，本手法を用いることで制御構造も含めた関数呼び出しパターンを取得できることが分かる．

また，この例と同じように抽出された全てのパターンを閲覧していったところ，抽出された 45 個のパターンカテゴリのうち，33 個のパターンカテゴリが利用法の理解に有用な関数呼び出しパターンを含んでいた．これらのことから，本稿での提案手法は部品の利用法理解に有用な関数呼び出しパターンを抽出できることが分かる．

6.2 既存手法との比較

ここでは，既存の関数呼び出しパターン抽出手法に存在した問題点が本手法によって改善されているかどうかについて評価する．既存手法の問題点とは，一つの関数呼び出しパターンの中に複数の機能に関する処理が含まれるおそれがあるという事である．

評価を行う為に，2 種類の方法で関数呼び出しパターンの抽出を行なった．一方は本研究で提案したソースコードの差分からの関数呼び出しパターン抽出手法である．もう一方は対象をソースコードの差分でなく，最新版のソースコード全体として提案手法を適用したものである．この手法は「同じ箇所で呼び出される事の多い関数群を 1 つのパターンにまとめる」という既存手法の特徴を持ちつつ，出力されるパターンの形式が提案手法のものと同じになるようにしている．この 2 種類の方法で抽出した関数呼び出しパターンを比較する事で，既存手法との比較を行ない，問題点が改善されているかどうか評価する．

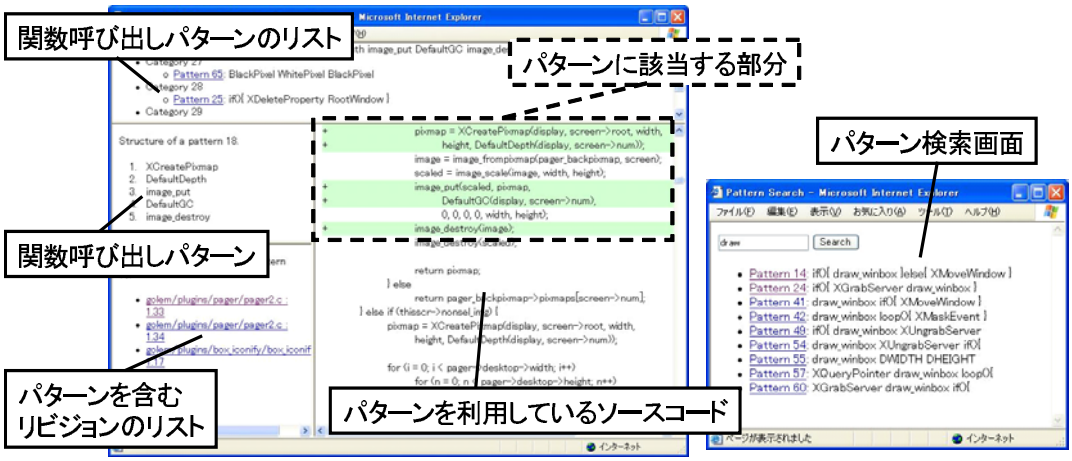

図 6: 関数呼び出しパターンブラウザとその検索画面

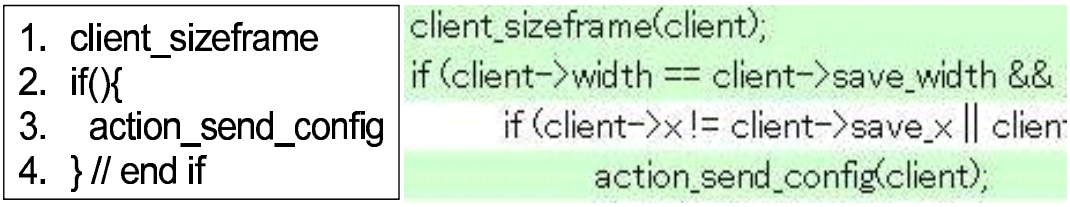

図 7: 関数呼び出しパターンとそれを利用しているソースコードの例

表 1: 最新版と差分から抽出したパターンの違い

既存手法によるパターン	提案手法によるパターン
PDEBUG	
if(){	
XCreatePixmap	XCreatePixmap
DefaultDepth	DefaultDepth
image_scale	
image_put	image_put
DefaultGC	DefaultGC
image_destroy	image_destroy

表 1 は画像の描画に関する関数呼び出しパターンの，既存手法と提案手法による違いを表している．既存手法のパターンを見てみると，明らかに画像の描画に関係の無い PDEBUG というデバッグ出力マクロが含まれている事や，画像の描画に必須ではない image_scale 関数の呼び出しが含まれている事が分かる．しかし，提案手法のパターンではそうした要素は含まれていない．よって，提案手法によるパターンは既存手法によるパターンから余分な要素を除去したものであるといえる．

他にも，抽出された利用法理解に有用なパターンを含むパターンカテゴリのうち，およそ半分が既存手法によるパターンを更に細かい単位に分割するパターンを含んでいた．以上の事から，複数の関連の無い機能が 1 つのパターンにされるという既存手法の問題点を提案手法が改善している事が分かる．

しかし，既存手法では利用パターンが抽出できるが，提案手法では抽出できないような部品が多く存在することもわかった．これは，初期状態から変更が加わっていない部品など，ソースコードの差分に利用実績が現れない部品も存在しているからである．また，提案手法からは最新版に存在しない部品に関するパターンも抽出されていた．このようなパターンは最新版の部品を用いる場合には活用する事ができない．

これらのことから，提案手法と既存手法を組み合わせた関数呼び出しパターン抽出手法の確立が今後の課題として挙げられる．これによって，それぞれのパターンが持つ問題点が相互に補われ，より質の高い関数呼び出しパターンを得ることができると考えられる．

7 まとめ

本稿では，ソフトウェア部品の利用法理解を支援するため，版管理システム内のソースコードの差

分から個別の機能に限定した関数呼び出しパターンを抽出する手法を提案した．本手法では，一度の更新で扱うソースコードは 1 つの機能のものであるという版管理システムを用いたソフトウェア開発の特性に注目して関数呼び出しパターンの抽出を行なった．また，提案手法を実装し，実際のオープンソースソフトウェアに対して適用したところ，部品の利用法理解に有用な関数呼び出しパターンを抽出することができた．さらに，既存手法との比較を行い，提案手法が実際に個別の機能に限定した関数呼び出しパターンを抽出し，既存手法の問題点を改善している事が確認した．

今後の課題としては，関数呼び出しパターンブラウザのさらなる改良や，既存手法と提案手法を組み合わせた関数呼び出しパターン抽出手法の確立などが挙げられる．

参考文献

[1] Agrawal, R. and Srikant, R.: Mining Sequential Patterns, *ICDE '95: Proceedings of the 11th International Conference on Data Engineering*, pp. 3–14 (1995).

[2] 渥美紀寿，山本晋一郎，結縁祥治，阿草清滋：FCDG に基づいたコーディングパターン，日本ソフトウェア科学会 コンピュータソフトウェア， Vol. 21, No. 4, pp. 27–36 (2004).

[3] Basili, V., Caldiera, G., McGarry, F., Pajerski, R., Page, G. and Waligora, S.: The Software Engineering Laboratory - an Operational Software Experience Factory, *Proceedings of the Fourteenth International Conference on Software Enigineering*, pp. 370–381 (1992).

[4] Dijkstra, E. W.: Notes on Structured Programming. `http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF`.

[5] Gall, H., Jazayeri, M. and Krajewski, J.: CVS Release History Data for Detecting Logical Couplings, *IWPSE: '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, pp. 13–23 (2003).

[6] Jacobson, I., Griss, M. and Jonsson, P.: *Software reuse: architecture, process and organization for buisiness success*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1997).

[7] 松本吉弘：ソフトウェアの考え方・作り方，電気書院 (1981).

[8] Michail, A.: Data Mining Library Reuse Patterns Using Generalized Association Rules, *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, pp. 167–176 (2000).

[9] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.: PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth, *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, pp. 215–224 (2001).

[10] The Golem X11 Window Manager: . `http://golem.sourceforge.net/`.

[11] Williams, C. C. and Hollingsworth, J. K.: Recovering System Specific Rules from Software Repositories, *MSR '05: Proceedings of the 2005 International Workshop on Mining Software Repositories*, pp. 7–11 (2005).

[12] Zimmermann, T., Weisgerber, P., Diehl, S. and Zeller, A.: Mining Version Histories to Guide Software Changes, *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pp. 563–572 (2004).