修士学位論文

題目

ソースコードの差分を用いた関数呼び出しパターンの抽出手法の提案と実装

指導教員
井上 克郎 教授

報告者
中山 崇

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

ソースコードの差分を用いた関数呼び出しパターンの抽出手法の提案と実装

中山 崇

内容梗概

ソフトウェア部品の再利用はソフトウェアの生産性や品質の向上に役立つため，非常に重要である．実際に再利用を行なう際には，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，利用法の理解には部品に附属するドキュメントやサンプルプログラムなどが用いられる．しかし，こうしたものが附属していないソフトウェア部品も多く，そのような部品では利用法の学習が困難となり，その結果，部品の再利用が難しくなっていた．

このような問題を解決する手法の一つとして，コーディングパターンの抽出というものが挙げられる．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のことである．開発者はこれを閲覧することで，再利用対象のソフトウェア部品の利用法を学ぶことができる．しかし，既存のコーディングパターン抽出手法では，複数の機能が 1 つのまとまりとして認識されてしまい，個別の機能に対するパターンが抽出できないという問題点が存在していた．そのため，開発者はある機能を実装するのに関係のないコード記述も同時に学習するおそれがあった．

そこで本研究では，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コーディングパターン抽出手法を提案する．この手法では，版管理システムに蓄積されているソースコードの差分が個別の機能に関連したものであるという事に着目している．また，実際のオープンソースプロジェクトに対して提案手法を実装したシステムを適用した．その結果，本手法がソフトウェア部品の再利用に有用なコーディングパターンを抽出し，かつ既存手法の問題点を解決している事を確認した．

主な用語

ソフトウェア部品
ソフトウェア再利用
コーディングパターン
版管理システム
Sequential pattern mining

目 次

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．プログラムソースの公開と再配布を許可しているオープンソースソフトウェア開発コミュニティでは，ソフトウェアプロダクトが増加するにつれて開発基盤としての価値だけでなく，ソフトウェアを構成する様々な部品を発見して他のソフトウェア開発に再利用したり参考にするといった，ソフトウェア資産を有する大規模なライブラリとしての価値も高まっている．SourceForge[23] ではソースコードのストレージやコミュニケーションに必要なメーリングリストなどソフトウェア開発に必要とされる物的資源を提供しており，2006 年 2 月現在 11,000 以上ものプロジェクトを保持し，開発者として登録されているユーザ数は 120 万を越える．

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 [4][16]．実際に再利用を行なう際には，まず，再利用するソフトウェア部品の利用法を理解しなければならない．ここでいうソフトウェア部品の利用法とは，部品の初期化の仕方や，処理を行なうために必要な関数の呼び出し順などである．一般に，ソフトウェア部品の利用法の理解は，そのソフトウェア部品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なう．例えば，ソフトウェア部品のドキュメントを閲覧することでその仕様を学んだり，さらにサンプルプログラムを閲覧することで詳細なコーディングの仕方を学んだりする．大規模なオープンソースのソフトウェア部品や，商用のソフトウェア部品などにはこのようなドキュメントやサンプルプログラムが附属する事が多い．しかし，規模の小さいソフトウェア部品ではこうしたものが附属していない場合も多く，そのような部品の利用法を学習するのは困難であった．

このような問題を解決する手法の一つとして，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中に頻繁に現れる構造のよく似たコード記述の事である．開発者はこれを閲覧する事で，再利用対象のソフトウェア部品を利用するためにはどのような関数をどのように呼び出せばよいかを学ぶ事ができる．しかし，既存のコーディングパターン抽出手法は，同じ箇所で利用される事の多いコード記述には関連がある，という仮定をおいているため，特に関連のないコード記述同士でも何箇所かでまとめて利用されていると，それらの間には関連があると見なされてしまうという問題点が存在していた．そのため，開発者はある機能の利用法を学ぶ際に，関係の無いコード記述も同時に学習してしまうおそれがあった．

そこで本研究では，個別の機能に限定したソフトウェア部品利用法を理解する事を目的としたコーディングパターン抽出手法を提案する．この手法では，版管理システムに蓄積され

ているソースコードの差分が個別の機能に関わるものである，という事に着目している．このことは，版管理システムへのソースコードの変更の保存が，一般に個別の機能毎に行なわれるという事にもとづいている．

以降，2 節では本研究の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説明し，3 節で版管理システムについて述べる．4 節では本研究で提案するコーディングパターン抽出手法について説明し，5 節ではその実装について述べる．6 節では提案手法の評価を行なう．最後に，7 節で本研究の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

2.1　ソフトウェア部品と再利用

一般に**ソフトウェア部品** *(Software Component)* とは**再利用** *(reuse)* できるように設計されたソフトウェアの実体とされている [16]．過去のソフトウェア開発における成果物などからなるソフトウェア部品の集合を**ライブラリ** *(library)* という．以下，ソフトウェア部品を単に部品と呼ぶ．ライブラリ中の部品を同一システム内や他のシステムで用いる事を再利用といい，ソフトウェアの生産性と品質を改善し，結果としてコストを削減するという報告が多く出されている [4][15]．部品という概念はプログラムソースコードやバイナリ実行ファイル，その他に設計仕様や構造，テスト項目，そしてそれらの文書などであったりと，ソフトウェア開発過程で生成されたあらゆる成果物に適用され，その種類は様々である．部品の再利用可能な度合を**再利用性** *(reusability)* といい，特定の設計やコーディングの標準に従う事で，部品の再利用性を向上させる事ができる．ソフトウェアの再利用は，部品の形式や再利用の目的をもとに，**ホワイトボックス再利用** *(white-box reuse)* と**ブラックボックス再利用** *(black-box reuse)* という分類をすることがある．

ホワイトボックス再利用 ：

> 仕様や文書，プログラムソースコードの再利用を指す．部品の内部構造に基礎を置くため，部品の詳細を把握する事が可能であり，用途に応じて修正可能でプラットフォームに非依存である．ホワイトボックス再利用における再利用者は主にソフトウェア開発者である．特にソースコード再利用に関しては，ライブラリ内の部品を利用者の再利用環境に応じて洗練する手法に関する研究 [18] などが存在する．

ブラックボックス再利用 ：

> 主にバイナリ実行ファイルの再利用を指す．具体的には JavaBeans や ActiveX/DCOM, CORBA などがあり，部品の詳細を知らずにその機能だけを再利用したいときに有効である．プログラムに詳しくないエンドユーザのソフトウェア開発で行なわれるもので，開発形態としてはビジュアルプログラミングなどがある．コンパイル不要で迅速に再利用可能な反面，プラットフォームに依存しており，詳細な解析は困難である．

本研究で扱う再利用とは，ホワイトボックス再利用，特にソースコードに対象を絞った再利用のことである．ソースコードは，それが実現する個別の機能毎にクラスや関数と呼ばれる単位に分けられる．開発者はこれらの個別の機能を実現する部品群を組み合わせる事で大きな機能を実現するプログラムを書いていく．また，これらのクラスや関数には，それ単体

では意味をなさず，複数のものをある一定の手順で組み合わせる事で初めて望んだ機能を実現するものも存在する．

2.2 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際には，まず再利用対象となる部品の利用法を学習しなければならない．例えば，再利用対象となる部品が関数である場合は以下のような事を学習しなければ望んだ機能を実現できない．

- 実現したい処理に必要な関数群
- 必要な関数群の呼び出し順
- 各関数の引数や返り値の扱い方

一般に，部品の利用法の学習はその部品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して行われる．ドキュメントには主に部品が実現する機能や仕様などが詳細に書かれている．また，サンプルプログラムには部品を利用して目的の機能を実現するためのソースコードの例が書かれている．開発者は部品のドキュメントを見る事で，その部品についての理解を深め，それからサンプルプログラムを読んだり模倣したりする事で，基本的な部品の利用法を学習していく．更に高度な利用法を学習するには，それまでに得られた基本的な知識とドキュメントのさらなる熟読が必要になる．

商用のソフトウェア部品や大規模なオープンソースのソフトウェア部品等にはこのようなドキュメントやサンプルプログラムが附属しているので，それを用いて利用法の学習を行なう事ができる．しかし，個人で開発した部品や規模の小さい部品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を学ぶ事が困難であるため，再利用が難しくなる．

2.3 コーディングパターン

ドキュメントやサンプルプログラムが附属していない部品の利用法の理解を支援するために，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し，それを開発者に提示する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] [12] [19][27]．その中の一つとして，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理解に役立てるという研究が存在する [20]．コーディングパターンとはソースコードに頻繁に出現する構造のよく似たコード記述の事である．

ソフトウェア部品を再利用するにはある一定の手順の踏まなければならない．例えば，部品に対して初期化作業を行ない，それから主要な処理を実行し，最後に終端処理を施して初

めて機能を果たす部品も存在する．そのため，ある部品をソースコード内のいくつかの箇所で利用しているということは，ソースコード内によく似たコード記述がいくつか現れる事になる．そこで，ソースコード内に頻繁に出現するコード記述であるコーディングパターンを抽出する事で，部品の利用例を取り出し，それを利用法の理解に役立てるのである．

コーディングパターンを閲覧する事で開発者は以下の事を学習する事ができる．

実現したい処理に必要な関数群 ：

一般のソフトウェア開発においては，一つの機能は複数の関数の組み合わせからなる事が多い．そこで，部品を再利用するにはまず，利用したい機能を実現している関数はどれであるかという事を理解しなければならない．コーディングパターンには一連の処理に必要な関数呼び出しが含まれているため，これを閲覧する事で処理に必要な関数群を理解する事ができる．

関連のある関数の呼び出し順 ：

一つの機能を実装する関数群の呼び出し順序にも一定の決まりがある．初期化作業を行なう関数は一番始めに呼び出さなければならず，ある関数によって得られる値を参照する関数はその関数より後に呼び出さなければならない．コーディングパターンは単なる関数の列挙ではなく，ソースコードの構造を持ったものであるため，そこに現れる関数呼び出しの順序を閲覧する事で具体的な呼び出し順を学習する事ができる．

関数の引数や返り値の扱い方 ：

コーディングパターンは部品の具体的な利用法を抽象化して取り出したものであるため，それだけでは具体的なソースコードの書き方までは理解できない．そこで，対象のコーディングパターンを実際に利用しているソースコードを閲覧する事で，引数の指定の仕方や返り値の扱い方といった詳細なコーディングの仕方を学習する事ができる．

2.4 既存のコーディングパターン抽出手法の問題点

既存のコーディングパターン取得手法は，同じ箇所に現れる関数呼び出しは同じ機能を実現するもの同士であると見なす，という特徴をもつ．一つの機能を実装するソースコードは一箇所にまとめて書いた方が理解もしやすく，実際にそのようにしてコーディングを行なう事が一般的となっている．また，この仮定にもとづいて関連する関数群を抽出したところ，良好な結果が得られたという研究も存在する事から [19]，この仮定は妥当なものであると考えられる．

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ことで，実際には関連のない関数間に関連があるとされてしまうおそれがある．例えばあるソフトウェアのソースコード

内の幾つかの箇所で，何らかの計算を行なうコード記述とデータをファイルに書き出す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呼び出されているので，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ファイルへの書き出し処理が含まれてるため，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．

この，関連の無い機能に対する関数呼び出しが一括りにされるおそれがあるという問題点は，開発者による部品の利用法理解を妨げ，その結果ソフトウェア開発の効率を下げていると考えられる．この問題点を解決するためには，個別の機能に限定したコーディングパターンの抽出を行なう事が必要となる．本研究ではこの問題点を解決するために，版管理システムに蓄積されているソースコードの差分が個別の機能に関連するコード記述であるという事に注目した．3 節では版管理システムについて説明した後，なぜ版管理システム内のソースコードの差分が一つの機能に関連するコード記述であるかについて述べる．

3　版管理システムとその性質

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，版管理システムへのソースコードの変更の保存が個別の機能毎に行なわれるという事について述べる．

3.1　版管理システム

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

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** *(Repository)* 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** *(Revision)* 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．

また，リポジトリとのデータ授受をするために，開発者はシステムに依存したオペレーションを利用する必要がある．

版管理手法を述べるに辺り，その基礎となるモデルが幾つか存在する [2] [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．なお，以降本文において，プロダクトのある時点における状態の事をリビジョンと呼ぶことにする．

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制御に関して定義されている（図 1 参照）．

リビジョン管理下にあるコンポーネン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はなく，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なう．リポジトリから特定のリビジョンのコンポーネントを取得する操作を**チェックアウト** *(Checkout)* という．逆に，データをリビジョンに格納し，新たなリビジョンを作成する操作を**チェックイン** *(Checkin)* という．

単純にリビジョンを作成するのみでは，シーケンシャルなリビジョン列を生成する事になる．しかし，過去のリビジョンに遡り，別の工程（デバッグ等）で開発を行なう場合のため

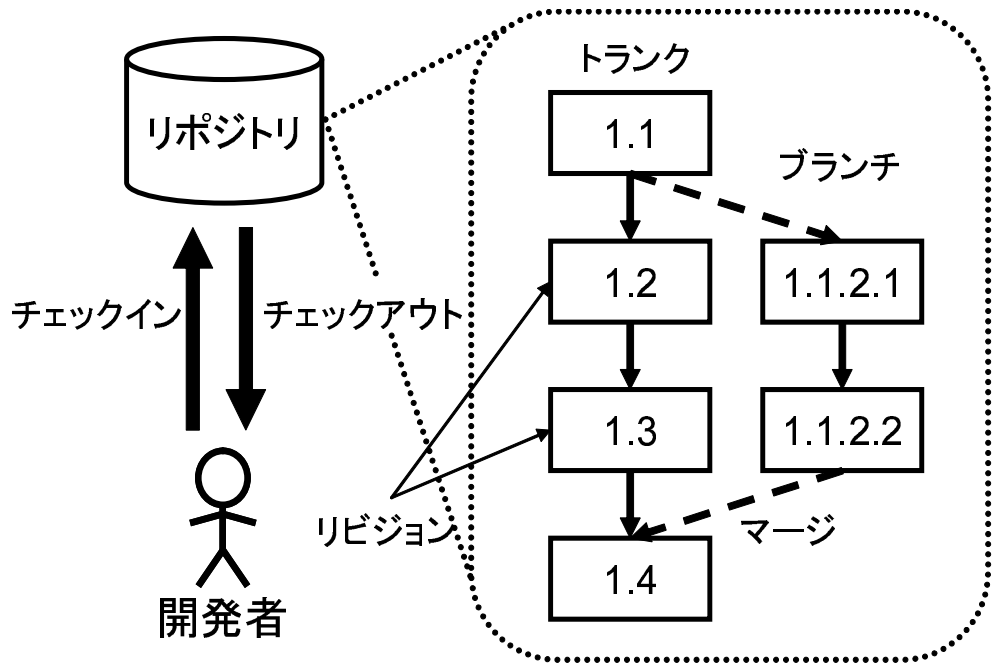

図 1: Checkin/Checkout Model

に，リビジョン列を**ブランチ** *(Branch)* という操作によりブランチを生成し，その上にリビジョンを作成するという手法を採る．このブランチに対して，元のリビジョン列の事を**トランク** *(Trunk)* という．また，ブランチ上での作業内容（デバッグ修正部分等）を，別のリビジョンに統合する作業を**マージ** *(Merge)* という．このように，リビジョン列は木構造になる事から**リビジョンツリー** *(Revision Tree)* と呼ばれる．

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[26] や CVS[5] [9] といったシステムが標準で利用可能となっている．ClearCase[7] 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，Visual SourceSafe[13] や PVCS[14] 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 [10] も存在する．

ここでは，版管理システムの内の幾つかを紹介する．

RCS ：

RCS[26] 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，版管理機構をもたせるなどの用途もある．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 つのファイルで行なわれる．

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 の持つツールを使用する．

CVS ：

CVS[5] [9] 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，近年最もよ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用する事も考慮し，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）を利用する事も可能である為，オープンソースによるソフトウェア開発の場面で活躍の場が多い．その最たる例が FreeBSD[25] や OpenBSD[21] 等のオペレーティングシステムの開発である．

3.2 ソースコード更新の性質

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版管理システムにチェックインしていく事で開発作業を進めていく．このとき，一度のチェックインで変更されるソースコードはある一つの機能についてのソースコードである事が一般的となっている．以下に，版管理システムへのチェックインが機能毎に行なわれることの根拠について述べる．

不完全なソースコードをチェックインする事はないため ：

版管理システムに変更内容を保存しながら開発を行なうのは，保存された開発状態を後で復元して利用するからである．例えば，開発途上のソースコードの既存機能の部分にバグが存在する事が分かったとする．ここで開発中のソースコードに対してバグ修正を行なうと，不完全なソースコードを扱う事になるため，新たなバグを埋め込む可能性が高くなる．そこで，過去の安定版のソースコードからブランチを作成し，そこでバグ修正をした結果をトランクへマージするという事を行なう．しかし，この過去の状態のソースコードが不完全な状態だと，このような作業を行なうことも困難になる．そのため，後でソースコードの状態が復元される事を考えて，一つ以上の機能が正しく更新された状態でソースコードをチェックインする事が慣習となっている．

開発状態の復元を柔軟に行なうため ：

チェックインした機能に問題が発生した場合，一旦そのチェックインの前の状態に戻るという事がある．このとき，複数の機能を同時にチェックインしていると，その内の

一つだけに問題があった場合でも，それら全ての機能への変更が無かった状態へと戻らなければならない．細かくチェックインを繰り返す事で，このような開発状態の復元作業に柔軟性を持たせる事ができる．

大規模なチェックインには困難が伴うため ：

現在広く使われている版管理システムには複数の開発者が同時に開発作業を行なう事を支援する機能を持つものが多い．それらの機能の一つに，複数の開発者間でのソースコード変更の競合に対処するための機能が挙げられる．しかし，競合による問題は全て版管理システムが解決するわけではなく，最終的には人間が解決することになる．そこで誤りが起きてしまう可能性もあるため，開発においてはできるだけ変更の競合は起こさないほうがよい．もし，一度に大量のソースコードをチェックインしていると，競合が起きたときに影響を受ける範囲が広くなってしまう．そのため，チェックインを行なうときは出来るだけ細かく行う方がよい．

また，既存の研究からも，このチェックインの性質が一般的であることが伺える．Gall らは版管理システム内の開発履歴から，ソースコードの構造からは分からない 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 [11]．他にも，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 [30]．これらの研究に共通しているのは，一度のチェックインで扱うソースコードは共通の機能を実装しているという事を仮定している点である．そしてこれらの研究が良い結果を残している事を併せて考えると，実際にソフトウェア開発においても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問題無く受け入れられるものである事が分かる．

4 節では，この版管理システムへのチェックインの性質を用いて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．

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，版管理システムに蓄積された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．なお，本手法では C 言語を対象としたコーディングパターンの抽出を行なう．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とらえているため，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ことにする．

本手法は大きく分けて 3 つのステップに分類される．まず始めに行なわれるのがソースコードの差分からソースコードの特徴と本研究で呼んでいるものを取得する事である．次に，取り出されたソースコードの特徴から特徴シーケンスと本研究で呼んでいるものを生成する．最後に，生成された全ての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．図 2 は本手法における関数呼び出しパターン抽出の流れを表したものである．

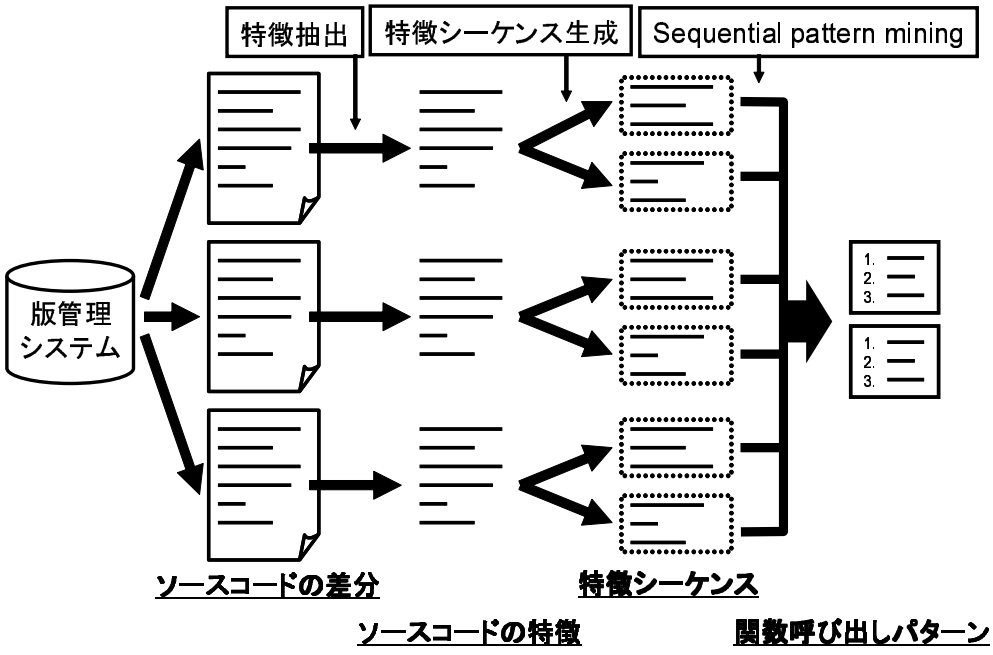

図 2: 関数呼び出しパターン抽出の流れ

以下では，これら 3 つのステップを 1 つずつ詳細に説明していく．

4.1 ソースコードの特徴の取得

まず始めにソースコードの差分からソースコードの特徴と呼ばれるものを取得する．本研究ではソースコードの特徴を以下のように定義している．

- ソースコード中に存在する，以下に挙げる要素をソースコードの特徴とする
  - 関数呼び出し
  - 条件文の開始・終了位置
  - 繰返し文の開始・終了位置

開発者はある機能を実装した関数を複数組み合わせる事で，より大きな機能を実現していく．よって関数呼び出しはそのプログラムがどういった事を実現しようとしているかを知るのに良い指標となることから，ソースコードの特徴として選んだ．また，関数呼び出しだけではなく，条件文の開始・終了位置や繰り返し文の開始・終了位置といったような，プログラムの制御構造を決定するような要素もソースコードの特徴とする事にした．関数呼び出しを組み合わせてある機能を実現する際に，単純に関数呼び出しを並べるだけで実行できるという事は少ない．ある関数呼び出しの返り値の結果によって処理を変えたり，処理が終るまで同じ関数呼び出しを繰り返すといった事がよく行なわれる．つまり，部品の利用法を学ぶときは制御構造も含めた部品利用例を閲覧する必要がある．そこで，ソースコードの特徴に条件文の開始・終了位置といった制御構造を決定付ける要素を含める事で，最終的に抽出される関数呼び出しパターンに制御構造の情報も持たせている．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特に制御文要素と呼ぶ事にする．

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に対して行なわれる．版管理システムではソースコードの差分を diff プログラムの出力で表しており，それらは新しいリビジョンにおいて追加された行，削除された行，そして編集された行の 3 種類に分類できる．特に diff プログラムでは編集された行は行の削除と追加を組み合わせて実現されているので，実質的にはソースコードの差分は追加された行と削除された行の 2 種類で構成されていると言える．本手法でソースコードの特徴の取得の対象となっているのは，それらの内の新しいリビジョンにおいて追加された行である．

ソースコードの特徴を取り出す対象を差分のみに限定する理由は，抽出する関数呼び出しパターンが個別の機能に限定されたものにする為である．3.2 節で述べたように，版管理システムへのチェックインは個別の機能についてのソースコードの変更を基本として行なわれる．つまり，隣接するリビジョン間の差分に注目する事は，全体のソースコードからある一

つの機能に関わるソースコードを抜き出す事と言い替える事が出来る．そのため，ソースコードの差分から取得した関数呼び出しパターンは個別の機能に限定されたものとなる．

次に，取得したソースコードの特徴から不要な要素を取り除いていく．取得した特徴の数は莫大となる為，不要な要素を取り除かないで関数呼び出しパターンを抽出しようとすると膨大な時間とメモリ空間が必要となる．そこで，不要な要素を除去してデータサイズを抑える事で，計算にかかる時間とメモリ空間を少なくしている．本研究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している．標準関数はドキュメントやサンプルプログラムも充実しており，本手法を用いて関数呼び出しパターンを抽出せずとも十分に利用法を学習できるとの判断から除去の対象とした．同じ構造の制御文要素のみの繰り返しとは例えば「*if* → *end* → *if* → *end* → *if* → *end*」といったものである．このようなものは関数の呼び出しパターンに寄与しているとは考えにくいので除去の対象とした．

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を特徴シーケンスという単位にまとめていく．特徴シーケンスとは関数の利用例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ので，特徴シーケンスを構成するソースコードの特徴は新しいリビジョンにおいて追加された行のみに限定される．

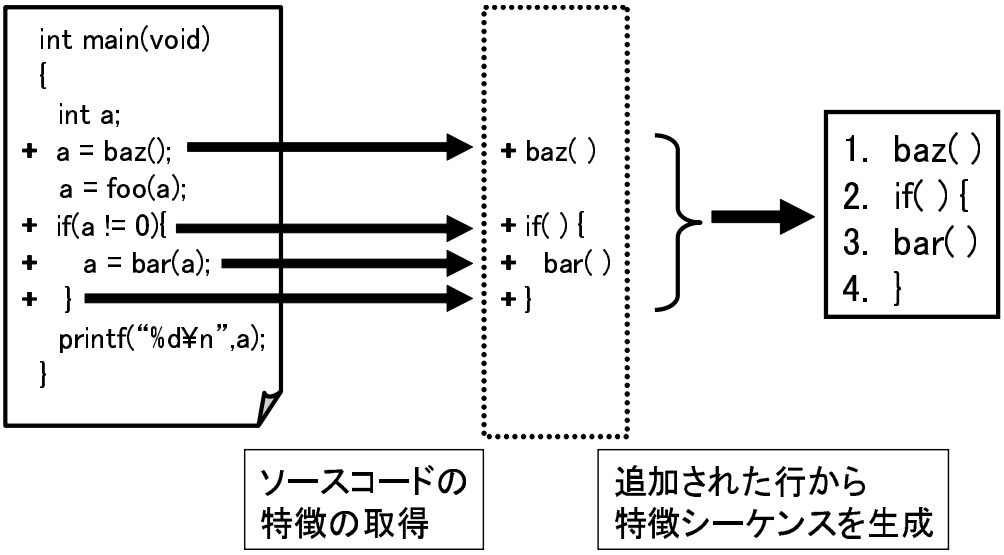

図 3: ソースコードの差分からの特徴シーケンス抽出の流れ

図 3 はソースコードの差分から特徴シーケンスを抽出するまでの流れを図として表したも

のである．図の左側はソースコードの差分を表しており，先頭に'+' がついている行は新しいリビジョンにおいて追加された行を表している．そこから，ソースコードの特徴を取得した後，追加された行に存在する特徴から 1 つの特徴シーケンスを生成している．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なう事で，全特徴シーケンスの取得が出来る．

次に，生成された特徴シーケンスから不要なシーケンスを除去する．ここでいう不要なシーケンスとは，関数呼び出し要素を 1 つも含まない特徴シーケンスの事である．本手法では主に関数呼び出しの手順を理解する為の呼び出しパターンを取得する事を目的としている．そのため，関数呼び出しを含まない特徴シーケンスは目的の関数呼び出しパターン抽出に必要はないと判断し，除去の対象とした．

4.3 Sequential pattern mining によるパターン抽出

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なう．

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]．Sequential pattern mining を行なうアルゴリズムには，AprioriAll[1]，GSP[24]，SPADE[28]，Spam[3] などが存在する．本研究では一般的に用いられることが多い PrefixSpan[22] を採用した．

PrefixSpan は射影と呼ばれる操作を繰り返すことでシーケンシャルパターンを抽出するアルゴリズムである．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．図 4 は要素 a についての射影を行なう様子を表している．

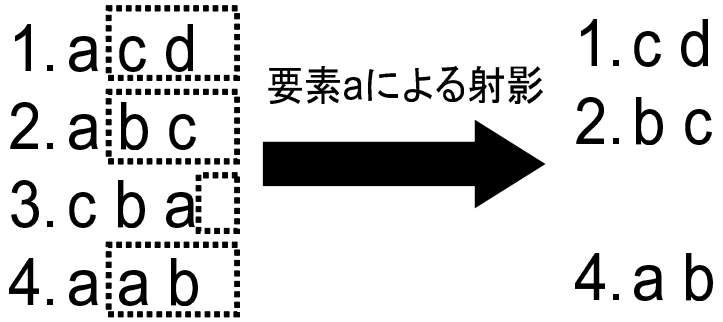

図 4: 要素 a についての射影

PrefixSpan アルゴリズムによる具体的な sequential pattern mining の流れを図 5 を用いて説明する．ここでは，パターン抽出の閾値である最小サポート値を 2 としてマイニングを行

なっている．まず，各シーケンスを構成している全ての要素について，そのサポート値を計算する．サポート値とは要素が出現しているシーケンスの数である．次に，サポート値が最小サポート値以上の要素をシーケンシャルパターンとして出力する．図 5 の例では a，b，c を長さ 1 のシーケンシャルパターンとして出力する．それから，最小サポート値を越える各要素について射影を行なう．そして射影を行なったシーケンスに対してまた各要素のサポート値を計算する．サポート値が最小サポート値以上ならば，その要素を 1 つ前に出力したパターンの末尾につけたものをシーケンシャルパターンとして出力する．図 5 の例では，要素 a についての射影を行なったシーケンスにおいて最小サポート値以上のサポート値を持つ b と c を，要素 a の後につけた ab と ac をシーケンシャルパターンとして出力する．この，射影，サポート値計算，シーケンシャルパターン出力という操作を最小サポート値を満たす要素が無くなるまで繰り返すことで，全てのシーケンシャルパターンを抽出する．

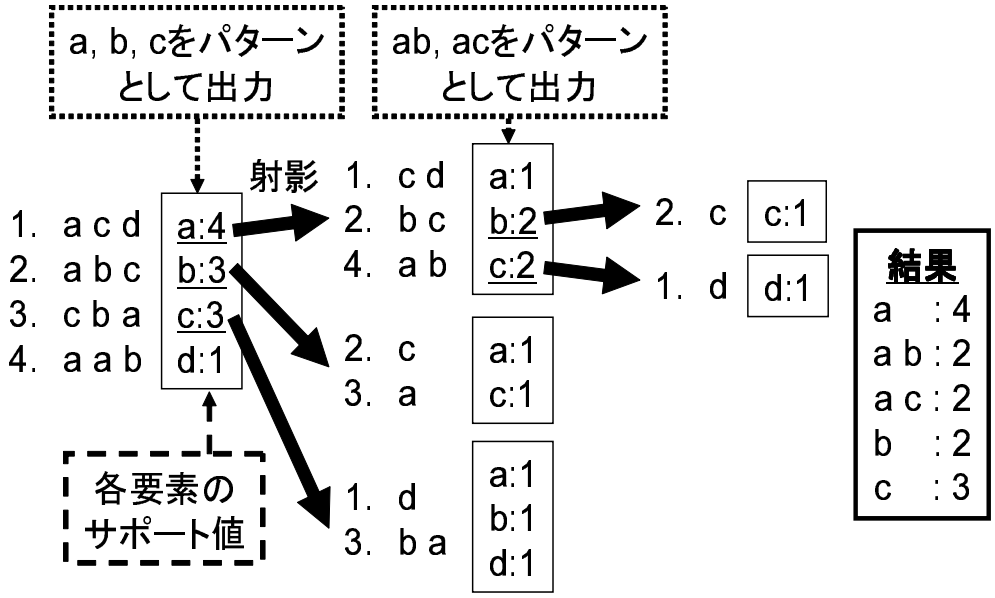

図 5: 最小サポート値 2 における PrefixSpan による sequential pattern mining

4.2 節のステップで生成した特徴シーケンスは部品の利用実績を抽象化したものと言える．そこで，この特徴シーケンス群に共通して存在するソースコードの特徴列を PrefixSpan アルゴリズムで抽出する事によって，頻繁に用いられる部品の利用手順を抽出する．

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，sequential pattern mining の性質により，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．部品利用法の理解に用いることのできない関数呼び出しパターンには以下のようなものが存在する．

制御文要素の対応が取れていないパターン :

パターンの要素を前から順に見ていったときに，if 文の開始位置が出現する前に if 文の終了位置や else 文が出現するようなパターンの事である．このようなパターンは再利用する事が出来ないので，抽出する価値はない．

制御文要素の数がパターン全体の要素数の 3 分の 2 を越えるパターン :

パターンを占める全要素の内，制御文要素の占める割合が 3 分の 2 を越えるようなパターンとは例えば，「*if* → *else* → *end if* → *if* → *call func1* → *call func2* → *end if*」のようなものである．このようなパターンでは大半の制御文要素は関数呼び出しの利用パターンとは関係が無く，パターン全体としての利用価値も低い．

関数呼び出し要素が 1 個以下しか無いパターン :

本研究では部品の利用パターンとして関数同士の相互呼び出し関係を対象としている．そのため，パターンの中に関数呼び出し要素が 0 個，ないしは 1 個しかないパターンはそのような相互呼び出し関係を表しているとは考えられない．そのため，関数呼び出し要素が 1 個以下しかないパターンは利用法理解に用いることができないと考えられる．

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資源を浪費させたり，開発者が欲する関数呼び出しパターンの検索を困難にしたりする．したがって，こうした不要な関数呼び出しパターンは除去する必要がある．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している．

対応する要素がない制御文要素を持つパターンを抽出しない :

PrefixSpan の各ステップでは，最小サポート値を持つ要素をその前のパターンの末尾に付加したうえでパターンとして出力している．このとき，図 6 のように付加する制御文要素に対応するものが存在しない場合にそのままパターン出力を行なうと，制御文要素の対応がとれていないパターンを抽出することになる．そこで，このような要素の付加を制限することで，制御文要素の対応がとれていないパターンの抽出を抑制する．

この制限では制御文要素の終了位置が欠けているだけのパターンの抽出は抑制できないが，本手法ではそのようなパターンは除去せずにそのまま出力することにしている．本手法では関数呼び出しパターンの抽出元がソースコードの差分となっているため，特徴シーケンス自体の制御文要素の対応がとれていないことも多い．そのような特徴シーケンスからパターン抽出を行なっているため，全ての制御文要素の対応がとれて

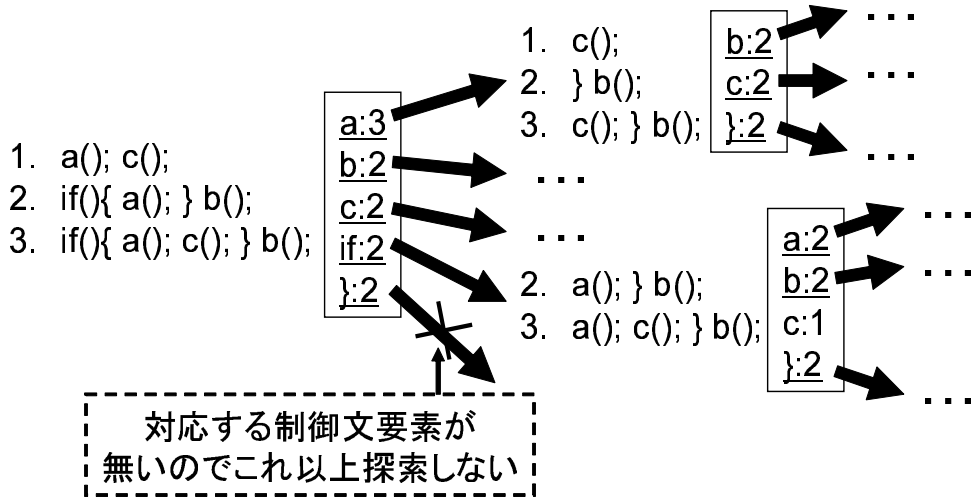

図 6: sequential pattern mining 時における不要なパターンの除去

いないパターンを除去するとほとんどパターンが残らないことになる．制御文要素の終了位置が欠けているだけのパターンではパターンを閲覧する側でそれを補うことができるため，このようなパターンは除去せずに残すことにしている．

一方，この制限をかけるとで PrefixSpan アルゴリズムの探索空間を削減できるため，計算に必要な時間やメモリ量を大幅に軽減することができるという副作用もある．

同じ種類の制御文要素を持つパターンを抽出しない：

一般に 1 つの関数呼び出しパターン中に同じ制御文要素が幾つも出現する事はない．そこで，同じ種類の制御文要素を持つパターンの抽出を制限することで，不要なパターンの除去を行なっている．具体的なパターンの除去手段は，対応する要素がない制御文要素を持つパターンと同様に，sequential pattern mining 中に随時行なう．この制限によって制御文要素が大半を占めるパターンの抽出が抑制される．

制御文要素が連続するパターンを抽出しない：

間に関数呼び出し要素をはさまない制御文要素の並びにはあまり意味が無い．そこで，パターン探索時に制御文要素が連続するような探索を抑えることで，制御文要素が連続するパターンが抽出されないようにする．この制限によって制御文要素が大半を占めるパターンの抽出が抑制される．

同じ関数呼び出し要素が連続するパターンを抽出しない：

一般に，同じ関数呼び出し要素が連続する場合，その関数呼び出し要素はそれ単体で役割をなしている事が多い．これらの要素の除去を行なわない場合，関数呼び出しパターンが必要以上に長くなって利用法の理解の妨げとなるため，sequential pattern mining 時の制限の対象とする．

制御文要素の数がパターン全体の 3 分の 2 を越えるパターンを除去する ：

全てのパターン抽出が終った後に，各パターンに対して検査を行ない，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 を越えているものを除去する．

関数呼び出し要素の数が 1 個以下であるパターンを除去する ：

全てのパターン抽出が終った後に，各パターンに対して検査を行ない，関数呼び出し要素数が 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取得手法を実装したシステムの詳細について述べる．

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出来る．1 つ目は，4.1 節と 4.2 節で説明したソースコードの特徴シーケンス生成を行なう部分である．2 つ目は，4.3 節で説明した関数呼び出しパターンの抽出を行なう部分である．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ブラウザ部である．システム全体の概要を図 7 に示す.

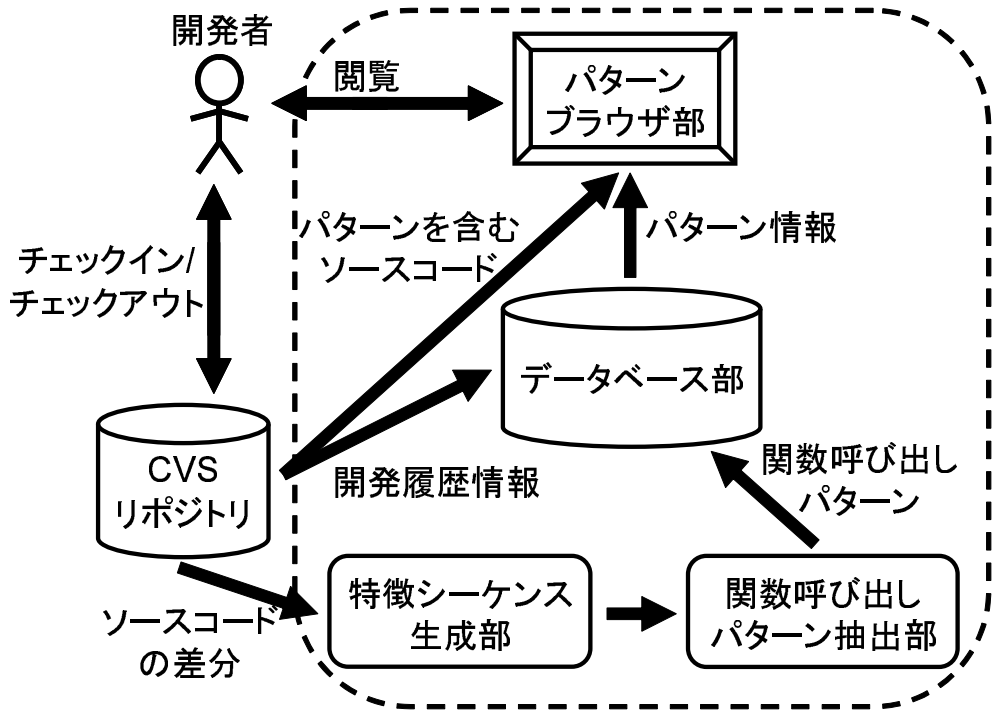

図 7: システムの概要

以下では本システムの 3 つのサブシステムと，それらが用いるデータの保存，取得を行なうデータベース部について説明していく．

なお，本システムの実装に用いた環境は以下の通りである．

- CPU:Pentium4 3.20GHz
- RAM:1GB
- OS:FreeBSD 6.0-STABLE

- データベース:Postgresql 7.3.13
- Web サーバ:Jakarta Tomcat 5.5.12
- 版管理システム：CVS
- JDK 1.4.2

5.1 データベース部

データベース部は表 1 のように，6 つのデータベースからなる．以下では各データベースの役割とそれがどのような情報を保持しているかを説明する．

表 1: システムが用いるデータベース

リビジョン DB	チェックイントランザクション DB	特徴シーケンス DB
ID ファイルパス リビジョン番号 更新日時 更新作業者 ログメッセージ	ID 更新日時 更新作業者 ログメッセージ 所属リビジョン ID のリスト	ID 特徴シーケンス 取得元リビジョンの ID 所属関数定義名
特徴番号 DB	**関数呼び出しパターン DB**	**パターン検索 DB**
ID 関数呼び出しパターン パターンが存在した特徴シーケンスの ID サマリ カテゴリ番号	ソースコードの特徴名 対応する整数値	関数名 その関数を持つパターン ID

リビジョンデータベース :

リビジョンデータベースは解析対象となるソフトウェアプロジェクトの全てのチェックイン履歴情報を保持するデータベースである．各チェックイン履歴情報には，チェックイン対象のファイルパス，リビジョン番号，更新日時，更新作業者，ログメッセージが含まれている．また，各リビジョン情報はそれらを特定するための ID を持つ．

チェックイントランザクションデータベース :

チェックイントランザクションデータベースは解析対象となるソフトウェアプロジェクトの全てのチェックイントランザクション情報を保持するデータベースである．チェックイントランザクションとは，一度の更新作業で扱った全てのファイルのチェックイン作業をまとめて表現したものである．各チェックイントランザクション情報には，更新日時，更新作業者，ログメッセージ，更新されたリビジョンの ID のリストが含まれている．また，各チェックイントランザクション情報はそれらを特定するための ID を持つ．

特徴シーケンスデータベース ：

特徴シーケンスデータベースは解析対象となるソフトウェアプロジェクトから取得した全ての特徴シーケンスの情報を保持するデータベースである．各特徴シーケンス情報には，取得元となったリビジョンの ID，取得元となった関数定義名，そして特徴シーケンス自身である．また，各特徴シーケンス情報はそれらを特定するための ID を持つ．

特徴シーケンスはソースコードの特徴のリストとなっており，各ソースコードの特徴は，ソースコードの特徴のタイプ，それが存在した行番号，そしてその特徴の名前を保持している．ソースコードの特徴のタイプには関数呼び出し型と制御文要素型が存在する．ソースコードの特徴の名前は，関数呼び出し型の場合はその関数名に，制御文要素型ではその制御文要素の名前に相当する．

特徴番号データベース ：

関数呼び出しパターン抽出時にソースコードの特徴をそのまま扱っていたのではデータ量と計算速度の両面において大きなロスとなる．そこで，各ソースコードの特徴を整数値の番号として表し，sequential pattern mining 時にはその番号を用いて計算を行なうようにしている．特徴番号データベースは，その番号とソースコードの特徴との対応関係を保持しているデータベースである．

関数呼び出しパターンデータベース ：

関数呼び出しパターンデータベースは抽出された関数呼び出しパターンに関する情報を保持するデータベースである．関数呼び出しパターンは特徴番号データベースにおける特徴番号のリストとして保持される．また，その関数呼び出しパターンがどの特徴シーケンスの中に存在したかという情報も保存する．そして，パターンを 1 行の文字列として表したサマリや，そのパターンが所属するカテゴリ番号なども同時に保存する．各関数呼び出しパターン情報はそれらを特定するための ID を持つ．

パターン検索データベース ：

パターン検索データベースは関数呼び出しパターンを関数名から検索する為に必要な情報を保持している．具体的には，各関数名から，その関数呼び出しを含んでいる関数呼び出しパターン情報の ID が保存されている．

5.2 特徴シーケンス生成部

特徴シーケンス生成部は 4.1 節と 4.2 節で述べた処理を実現している．

まず始めに全てのチェックイン情報を取得する為に，CVS から取得できるログを解析する．解析して得られたチェックイン情報は全てリビジョンデータベースに保存される．

そして取得したチェックイン情報からチェックイントランザクション情報を復元する．チェックイントランザクションの復元には，[29] で述べられている手法を用いている．復元された情報は全てチェックイントランザクションデータベースに保存される．

次に各ソースコードの差分から提案手法の通りに特徴シーケンスを生成していく．全リビジョン情報について，対象のファイルのリビジョンとその直前のリビジョンとの diff の出力を解析し，新しいリビジョンのどの行が特徴シーケンスの取得対象となるかを計算する．次に，新しいリビジョンのソースコードを解析し，追加された行か否かに関わらず全てのソースコードの特徴を取得する．そしてこれらを先に計算した追加された行がどこかという情報とすりあわせる事で，新しいリビジョンにおいて追加された行に存在するソースコードの特徴のリストを得る．さらに，この特徴のリストと，各関数定義が何行目から何行目までに渡っているかという情報を合わせることで，各関数定義内に存在するソースコードの特徴のリスト，すなわち特徴シーケンスを得る事が出来る．取得された各種情報は特徴シーケンスデータベースに保存される．

この特徴シーケンス生成処理は基本的に全てのリビジョンに対して行なわれる．ただし，マージと考えられるチェックイントランザクションに属するリビジョンに対しては行なわれない．マージとはブランチに蓄積されたソースコードの変更をトランクへと反映する作業であり，この場合は多くの機能についての変更が一度にチェックインされる事が多い．そのため，このチェックインから得られる特徴シーケンスは個別の機能に関する関数呼び出しパターンを抽出するという目的にそぐわない．したがって，マージと考えられるチェックインからは特徴シーケンスを取得しないようにしている．

CVS はどのチェックイントランザクションがマージであるかを保持していない．そこで，本システムでは一度に大量のファイルを変更しているチェックイントランザクションをマージであると判定している．

5.3 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は 4.3 節で述べた処理を実現している．

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．

Sequential pattern mining アルゴリズムは PrefixSpan[22] を採用している．このアルゴリズムを Java 言語で実装したものを用いて，特徴シーケンスデータベース中の全特徴シーケンスと，ユーザが指定する閾値から関数呼び出しパターンを抽出する．

抽出された関数呼び出しパターンはカテゴリ分けされる．ここでは，同じ種類の関数呼び出し要素を持つパターン同士を同じカテゴリとして分類した．このことによって，同じ関数群が少し違う呼び出し方をされているという状況を把握しやすくした．また，各パターンを一行の文字列として簡単に表現するサマリも生成する．このサマリはパターンブラウザにおける関数呼び出しパターンの簡易表示に用いられる．そして，各パターンがどのリビジョンのどの箇所に存在したかも調べる．これらの情報は全て各パターンと共に関数呼び出しパターンデータベースへと保存される．また，各関数呼び出し要素がどのパターンに存在したかという情報も計算してパターン検索データベースへと保存する．これらの情報は 5.4 節で述べる関数呼び出しパターンブラウザによって用いられる．

5.4 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，抽出されたコーディング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 8 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．

画面の上部には抽出された関数呼び出しパターンがカテゴリ別にリストされている．この中から一つのパターンを選択してクリックすると，左下部に関数呼び出しパターンとそれを実現しているリビジョンのリストが表示される．リビジョンのリストの内の一つを選択してクリックする事で，右下部に関数呼び出しパターンを実現しているソースコードが表示される．このソースコードのうち，頭に’ +’ がついている行が新たに追加された行を表しており，色がついた行は該当パターンが存在する行を表している．開発者はこれらを見ながら部品の利用法について学習していく．

また，画面上部のリンクをクリックする事でパターン検索画面へ移る事が出来る．パターン検索画面では，開発者が利用パターンを学びたい関数名を入力する事で，その関数呼び出しを含むコーディングパターンを検索する事ができる．検索された各コーディングパターンに張られているリンクをクリックする事で，そのパターンの内容とリビジョンのリストを閲覧する事ができる．図 9 は検索画面のスクリーンショットである．

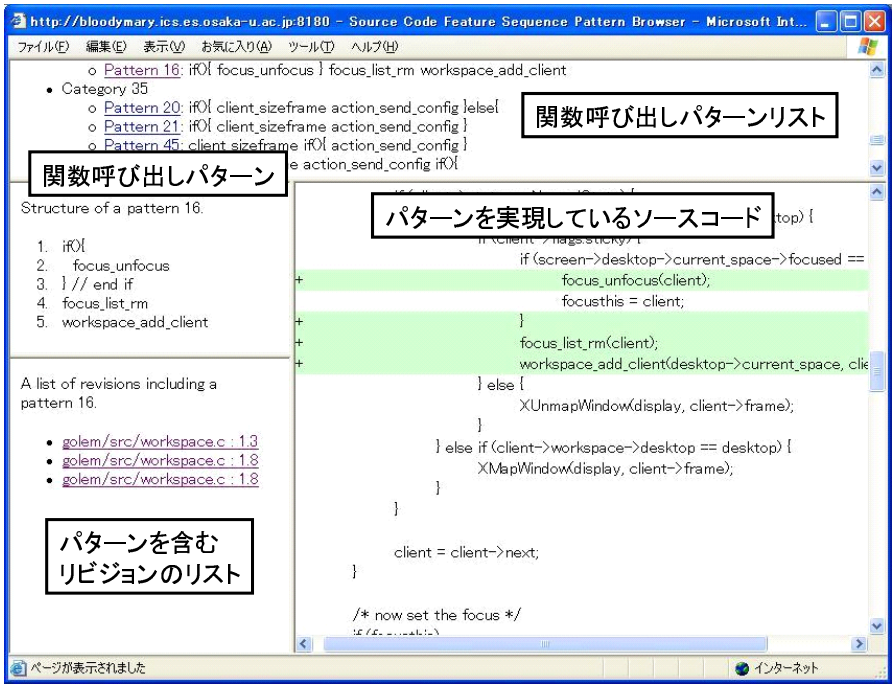

図 8: 関数呼び出しパターンブラウザ

図 9: パターン検索画面

6 評価

本節では，本研究で提案した関数呼び出しパターン取得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，既存の関数呼び出しパターン抽出手法の問題点を解決できているかどうかについて評価していく．

評価に用いた環境はシステム作成におけるものと同じなので，ここでは説明を省略する．

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は The Golem X11 Window Manager[17]（以下，Golem と表記）を選択した．Golem の概要は表 2 の通りである．

表 2: The Golem X11 Window Manager の概要

開発期間	2001/03/28 〜 2005/05/29
開発者数	3
総リビジョン数	2680
総チェックイントランザクション数	607
ソースファイル数（最新版）	180
総行数（最新版のソースファイルのみ）	55758

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 個となった．また，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見ていく事で，それらの部品の利用法理解における有用性を評価していく．

図 10 は画像の描画に関する関数呼び出しパターンである．パターンを詳しく見ると，`XCreatePixmap` 関数で画像を取得し，`image_put` 関数で画像を描画，そして `image_destroy` 関数で取得した画像の解放という流れを理解することができる．また，パターンを実現しているソースコードを見る事で `DefaultDepth` 関数は `XCreatePixmap` 関数の，`DefaultGC` 関数は `image_put` 関数の引数として用いられている事が分かる．

図 11 はクライアントのリサイズを行な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`

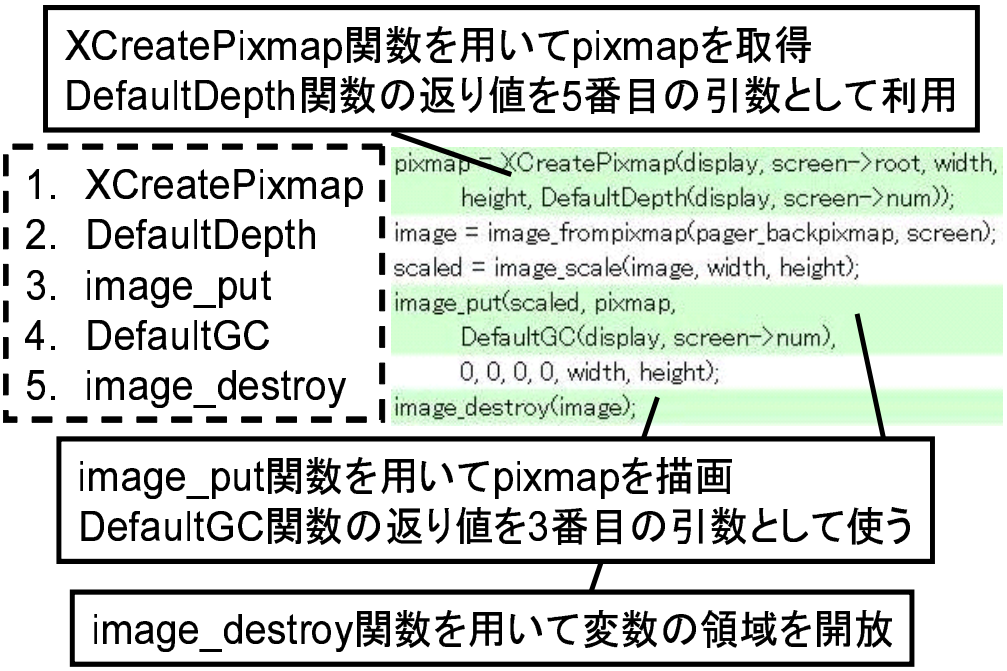

図 10: 関数呼び出しパターンの例 1

関数を用いて結果をサーバに送信していることが分かる．このパターンからは，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．

このように，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見る事で，機能を実現する為にどのような関数をどのように用いればよいのかが理解できた．また，これら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，本研究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取得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点である．

評価を行なう為に，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に対して提案手法を適用した

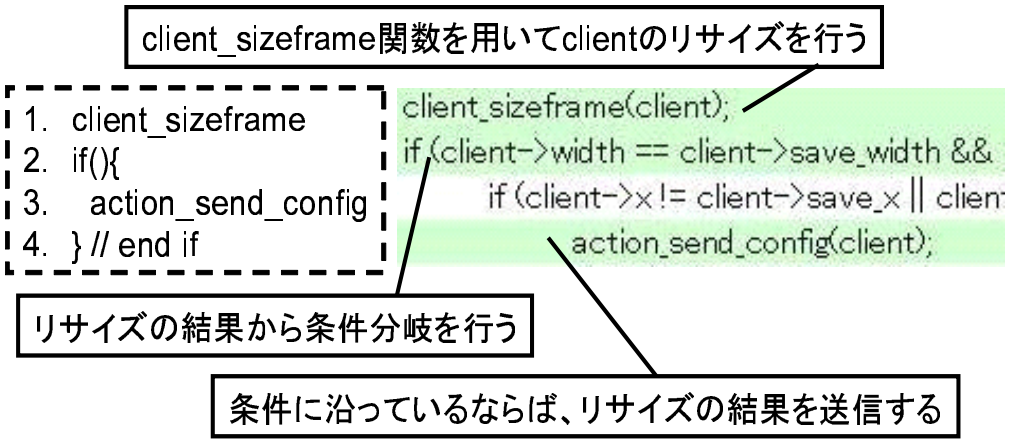

図 11: 関数呼び出しパターンの例 2

ものである．この手法は「同じ箇所で呼び出されている関数に関連があるという仮定のもとで，最新版のソースコードからパターンを抽出する」という既存手法の思想にマッチしている．そこで，これら 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を評価する．

表 3 は最新版のソースコードから抽出した関数呼び出しパターンと，ソースコードの差分から抽出した関数呼び出しパターンの違いの一例である．これを見ると，差分から抽出したパターンはほぼ最新版から抽出したパターンのサブセットであるといえる．

次に，最新版から抽出した関数呼び出しパターンを実現しているソースコードを調査した．その中の一つは `screen_manage` という関数定義内に存在しており，パターンの各要素は関数定義内の全体に渡って分布していた．つまり，最新版から抽出したパターンは `screen_manage` という処理に関する関数呼び出しパターンであるといえる．

一方，差分から抽出した関数呼び出しパターンが `screen_manage` 関数のどの部分に分布しているかを調べたところ,「スクリーンのカラーを取得する」という，`screen_manage` 関数の処理の中の更に細かい機能にあたる部分のパターンである事が分かった．

これらのことから，差分から抽出した関数呼び出しパターンは，最新版から抽出した関数呼び出しパターンに含まれる複数の機能の内の 1 つを抜き出している事が分かる．

表 4 も同様に最新版から抽出したパターンと差分から抽出したパターンの違いの一例である．どちらのパターンも画像を取得し，それを描画する処理に関するパターンである．最新版から抽出した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が分かる．また，2 つのパターンを見比べる事で，`image_scale` 関数の呼び出しが必須処理でなさそうである事が分かる．事実，`image_scale` 関数は画

表 3: 最新版と差分から抽出したパターンの違い 1

最新版から取得したパターン	差分から取得したパターン

DefaultColormap
WhitePixel
if(){
　XParseColor
　DefaultColormap
}else{
　XAllocColor
　DefaultColormap
　BlackPixel
}
XCreateGC
	if(){
	　XParseColor
	　DefaultColormap
	　BlackPixel
	}else{
	　XAllocColor
	　DefaultColormap
	}

像のスケーリング処理を行なう関数であり，画像の描画自体には必須とは言えないものである．よって，差分から抽出した関数呼び出しパターンの方が，画像の描画という一つの機能の利用法を学びやすい事が分かる．

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内，およそ半分が最新版から抽出したパターンを更に細かい単位に分割していた．以上のことから，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という本研究で提案する手法が，複数の関連の無い機能が 1 つのパターンとされるおそれがあるという既存手法の問題点を改善していることが分かる．

しかし，この比較実験によって提案手法の問題点も明らかになった．差分から抽出したパターンの数は 74 個であったのに対して，最新版から抽出したパターンは 440 個であった．つまり，差分から抽出する手法では利用パターンが抽出できない部品が相当数あるという事である．これは，ソースコードの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状況が現れない部品が多いからであると思われる．

また，差分から抽出したものには存在するが，最新版から抽出したものには存在しないパターンも幾つか見られた．これは過去のソースコードで用いられた関数呼び出しパターンがソースコードの変更によって消えたことが原因であると思われる．このようなパターンは最新版のソースコードを編集する場合，利用法の理解に活用することができない．

そこで，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，

表 4: 最新版と差分から抽出したパターンの違い 2

最新版から取得したパターン	差分から取得したパターン
PDEBUG	
if(){	
XCreatePixmap	XCreatePixmap
DefaultDepth	DefaultDepth
image_scale	
image_put	image_put
DefaultGC	DefaultGC
image_destroy	image_destroy

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

本研究では，ソフトウェア部品の利用法理解を支援するため，版管理システムに蓄積された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手法を提案した．本手法では，関連の無い複数の機能が 1 つのコーディングパターンとしてまとめられるおそれがあるという既存手法の問題点を解決する為に，個別の機能に限定した関数呼び出しパターンを得ることを目的としている．この目的を達するために本研究では，一度のチェックインで扱うソースコードは 1 つの機能に関する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プロセス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プロジェクト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．そして，最新版のソースコードのみから抽出した関数呼び出しパターンと比較を行なう事で，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できた．

今後の課題としては，パターン取得の対象となるソフトウェアプロジェクトのスケーラビリティの向上，関数呼び出しパターンブラウザのさらなる改良，そして，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．

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.

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] Rakesh Agrawal and Ramakrishnan Srikant. Mining Sequential Patterns. In *ICDE '95: Proceedings of the 11th International Conference on Data Engineering*, pp. 3–14, Washington, DC, USA, March 1995. IEEE Computer Society.

[2] Ulf Asklund, Lars Bendix, Henrik Bærbak Christensen, and Boris Magnusson. The Unified Extensional Versioning Model. In *SCM-9: Proceedings of the 9th International Symposium on System Configuration Management*, pp. 100–122, September 1999.

[3] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential PAttern Mining using a Bitmap Representation. In *KDD '02: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 429–435, New York, NY, USA, July 2002. ACM Press.

[4] Victor Basili, Gianluigi Caldiera, Frank McGarry, Rose Pajerski, Gerald Page, and Sharon Waligora. The Software Engineering Laboratory - an Operational Software Experience Factory. In *Proceedings of the Fourteenth International Conference on Software Enigineering*, pp. 370–381, New York, NY, USA, May 1992. ACM Press.

[5] Brian Berliner. CVS II: Parallelizing Software Development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pp. 341–352, Berkeley, CA, January 1990. USENIX Association.

[6] Reidar Conradi and Bernhard Westfechtel. Version models for software configuration management. *ACM Computing Surveys*, Vol. 30, No. 2, pp. 232–282, 1998.

[7] Rational Software Corporation. Rational clearcase. `http://www.rational.com/products/clearcase/`.

[8] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. Bugs as Deviant Behavior: A General Approach to Inferring Errors in System Code. In *SOSP '01: Proceedings of the 18th ACM symposium on Operating Systems Principles*, pp. 57–72, New York, NY, USA, October 2001. ACM Press.

[9] Karl Franz Fogel. *Open Source Development with CVS*. Coriolis Group Books, Scottsdale, AZ, USA, 1999.

[10] Peter Fröhlich and Wolfgang Nejdl. WebRC: Configuration Management for a Cooperation Tool. In *ICSE '97: Proceedings of the SCM-7 Workshop on System Configuration Management*, pp. 175–185, London, UK, May 1997. Springer-Verlag.

[11] Harald Gall, Mehdi Jazayeri, and Jacek Krajewski. CVS Release History Data for Detecting Logical Couplings. In *IWPSE: '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, pp. 13–23, Washington, DC, USA, September 2003. IEEE Computer Society.

[12] Reid Holmes and Gail C. Murphy. Using Structural Context to Recommend Source Code Examples. In *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pp. 117–125, New York, NY, USA, May 2005. ACM Press.

[13] Microsoft Inc. Visual sourcesafe. `http://msdn.microsoft.com/ssafe/`.

[14] Serena Software Inc. Pvcs version control software. `http://www.serena.com/pvcs_version_control_software.html`.

[15] Sadahiro Isoda. Experience Report on Software Reuse Project: Its Structure, Activities, and Stastical Results. In *Proceedings of the Fourteenth International Conference on Software Engineering*, pp. 320–326, May 1992.

[16] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software reuse: architecture, process and organization for buisiness success*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

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

[18] Katsuhisa Maruyama and Ken ichi Shima. A Mechanism for Automatically and Dynamically Changing Software Components. In *Proceedings of the 1997 symposium on Software reusability*, pp. 169–180, New York, NY, USA, May 1997. ACM Press.

[19] Amir Michail. Data Mining Library Reuse Patterns Using Generalized Association Rules. In *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, pp. 167–176, New York, NY, USA, June 2000. ACM Press.

[20] 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, July 2004.

[21] OpenBSD. `http://www.openbsd.org/`.

[22] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. PrefixSpan: Mining Sequential Patterns by Prefix-Projected Growth. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, pp. 215–224, Washington, DC, USA, April 2001. IEEE Computer Society.

[23] SourceForge. `http://sourceforge.net/`.

[24] Ramakrishnan Srikant and Rakesh Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*, Vol. 1057 of *Lecture NOtes in Computer Science*, pp. 3–17. Springer, March 1996.

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

[26] Walter F. Tichy. RCS - A System for Version Control. *Software - Practice and Experience*, Vol. 15, No. 7, pp. 637–654, 1985.

[27] Chadd C. Williams and Jeffrey K. Hollingsworth. Recovering System Specific Rules from Software Repositories. In *MSR '05: Proceedings of the 2005 International Workshop on Mining Software Repositories*, pp. 7–11, New York, NY, USA, May 2005. ACM Press.

[28] Mohammed J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, Vol. 42, No. 1-2, pp. 31–60, 2001.

[29] Thomas Zimmermann and Peter Weisgerber. Preprocessing CVS Data for Fine-Grained Analysis. In *MSR 2004: Proceedings of International Workshop on Mining Software Repositories*, pp. 2–6, May 2004.

[30] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. Mining Version Histories to Guide Software Changes. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pp. 563–572, Washington, DC, USA, May 2004. IEEE Computer Society.

付録

A Sequential pattern mining

ここでは，sequential pattern mining という問題の定義について述べる．

A.1 用語の解説

$I = \{i_1, i_2, \cdots, i_n\}$ を全ての**アイテム (item)** のセットとし，**アイテムセット (itemset)** を I のサブセットとする．また，**シーケンス (sequence)** とは順序付けられたアイテムセットのリストである．シーケンス s は $< s_1 s_2 \cdots s_l >$ とも表現する．このとき，$s_j \subseteq I (1 \leq j \leq l)$ である．シーケンスの各要素は**要素 (element)** とも呼ばれ，$x_k \in I (1 \leq k \leq m)$ とした時に $(x_1 x_2 \cdots x_m)$ と書くこともある．1 つのアイテムは 1 つの要素の中には 1 つしか持てないが，1 つのシーケンスの各要素に渡って複数回現れることはある．

あるシーケンスが持つアイテムの数をそのシーケンスの**長さ (length)** と呼ぶ．長さが l であるシーケンスのことを ***l*-sequence** と書く．

2 つのシーケンス，$\alpha =< a_1 a_2 \cdots a_n >$ と $\beta =< b_1 b_2 \cdots b_m >$ があったとき，$a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, \cdots, a_n \subseteq b_{jn}$ かつ，$1 \leq j_1 < j_2 < \cdots < j_n$ となるような整数が存在するならば，α は β の**サブシーケンス (subsequence)** であるといい，β は α の**スーパーシーケンス (super sequence)** であるという．また，$\alpha \sqsubseteq \beta$ とも書く．

シーケンスデータベース (Sequence database) S とは，$< sid, s >$ という二項組のセットである．*sid* は**シーケンス ID(sequence-id)** と呼ばれるものであり，s はシーケンスである．シーケンス α がシーケンス s のサブシーケンスであるとき，二項組 $< sid, s >$ はシーケンス α を含んでいるという．

シーケンスデータベース S におけるシーケンス α の**サポート (support)** とは，α を含む二項組の数のことである．S におけるシーケンス α のサポートのことを $support_S(\alpha)$ と書く $(support_S(\alpha) = |\{< sid, s > | (< sid, s > \in S) \wedge (\alpha \sqsubseteq s)\}|)$．文脈としてシーケンスデータベースが明らかな場合は簡単に $support(\alpha)$ とも書く．

ある正整数 ξ が**サポート閾値 (support threshold)** として与えられたとき，$support_S(\alpha) \geq \xi$ となるようなシーケンス α のことを**シーケンシャルパターン (sequential pattern)** と呼ぶ．サポート閾値のことは**最小サポート値 (minimum support)** とも呼ぶ．長さが l であるようなシーケンシャルパターンのことを ***l*-pattern** と呼ぶ．

A.2 問題文

Sequential pattern mining とは，シーケンスデータベースと最小サポート値が与えられた時に，シーケンスデータベース中に存在する全てのシーケンシャルパターンを見つけるという問題である．

B PrefixSpan

ここでは，sequential pattern mining を行なう一般的なアルゴリズムである **PrefixSpan** について説明する．

B.1 用語の解説

シーケンス $\alpha =< e_1 e_2 \cdots e_n >$ とシーケンス $\beta =< e'_1 e'_2 \cdots e'_m >$ が与えられたとき，以下の条件が成り立つときかつそのときに限り β は α の**プリフィックス (prefix)** であるという．

- $e'_i = e_i (i \leq m-1)$
- $e'_m \subseteq e_m$
- $(e_m - e'_m)$ 内の全てのアイテムがアルファベット順で e'_m 内の要素より後のものである

シーケンス α とそのサブシーケンスである β があったとき，シーケンス α' のことを以下の条件が成り立つときかつそのときに限りプリフィックス β に関する α の**射影 (projection)** であるという．

- α' がプリフィックス β を持つ
- プリフィックス β を持つ α のサブシーケンスで，かつ α' のスーパーシーケンスとなるような α'' が存在しない

$\alpha' =< e_1 e_2 \cdots e_n >$ をプリフィックス $\beta =< e_1 e_2 \cdots e_{m-1} e'_m > (m \leq n)$ に関する α の射影であるとする．このとき，$\gamma =< e''_m e_{m+1} \cdots e_n >$ をプリフィックス β に関する α についての**ポストフィックス (postfix)** と呼び，$\gamma = \alpha /\lrcorner \beta$ と書く．ただし，$e''_m = (e_m - e'_m)$ である．$\alpha = \beta \cdot \gamma$ とも書く．

もし β が α のサブシーケンスでないならば，β に関する α の射影もポストフィックスも存在しない．

α をシーケンスデータベース S におけるシーケンシャルパターンであるとする．このとき，S 中の各シーケンスの α に関するポストフィックスの集合のことを **α-projected database** と呼び，$S|_\alpha$ のように書く．

B.2 アルゴリズムの説明

入力：シーケンスデータベース *S* と最小サポート値である *min_sup*

出力：全てのシーケンシャルパターンからなるセット

手続き : $PrefixSpan(<>, 0, S)$ を呼び出す．ただし，$<>$ は長さが 0 のシーケンシャルパターンを表している．

サブルーチン **PrefixSpan** は以下のようなものである．

引数 :

α : シーケンシャルパターン

l : α の長さ

$S|_\alpha$: α の長さが 0 でないときは $alpha$-projected database を表し，0 であるときはシーケンスデータベース S を表す．

手続き :

1. $S|_\alpha$ 内を走査し，以下の条件に合うような最小サポート値以上のサポート値を持つアイテム b のセットを見つける．
   - シーケンシャルパターンを生成するために，α の最後の要素に b を組み合わせることが出来る．
   - シーケンシャルパターンを生成するために，α の末尾に $< b >$ を付け加えることが出来る．
2. 見つかった各アイテム b を α に付け加えて新たなシーケンシャルパターン α' を生成し，出力する．
3. 各 α' について α'-projected databse $S|_{\alpha'}$ を生成し，$PrefixSpan(\alpha', l+1, S|_{\alpha'})$ を呼び出す．

C　抽出された関数呼び出しパターン

ここでは，6.1 節の実験において抽出された関数呼び出しパターンを全てカテゴリ毎に掲載する．カテゴリ番号の横に記されているのはそのカテゴリ内の関数呼び出しパターンが持つ関数名である．

- Category 1: [DefaultColormap, DefaultColormap, XAllocColor, XParseColor]
  - if(){ XParseColor(); DefaultColormap(); XAllocColor(); DefaultColormap(); }else{
  - XParseColor(); DefaultColormap(); XAllocColor(); DefaultColormap(); if(){
- Category 2: [XMoveWindow, draw_winbox]
  - if(){ draw_winbox(); }else{ XMoveWindow(); }
  - draw_winbox(); if(){ XMoveWindow(); }
- Category 3: [DHEIGHT, DWIDTH]
  - if(){ DWIDTH(); DHEIGHT(); }
  - DWIDTH(); if(){ DHEIGHT(); }
  - DWIDTH(); DHEIGHT(); if(){
- Category 4: [XSetWindowBackgroundPixmap, pager_getpagedbg]
  - if(){ XSetWindowBackgroundPixmap(); pager_getpagedbg(); }
- Category 5: [OPTIONAL_PARAM, PWARN]
  - OPTIONAL_PARAM(); if(){ PWARN(); }
  - OPTIONAL_PARAM(); PWARN(); if(){
- Category 6: [XUngrabServer, draw_winbox]
  - if(){ draw_winbox(); XUngrabServer();
  - draw_winbox(); XUngrabServer(); if(){
- Category 7: [client_setstate, plugin_iconify_notify]
  - plugin_iconify_notify(); client_setstate();
  - client_setstate(); plugin_iconify_notify();

- Category 8: [XGetGeometry, XMapWindow]
  - if(){ XGetGeometry(); XMapWindow(); }
- Category 9: [image_frompixmap, image_scale]
  - if(){ image_frompixmap(); image_scale(); }
  - image_frompixmap(); image_scale(); if(){
- Category 10: [BlackPixel, WhitePixel]
  - if(){ BlackPixel(); WhitePixel(); }else{
  - if(){ BlackPixel(); WhitePixel(); }
  - BlackPixel(); WhitePixel(); if(){
- Category 11: [DHEIGHT, DWIDTH, FULLHEIGHT, FULLWIDTH]
  - FULLWIDTH(); FULLHEIGHT(); DWIDTH(); DHEIGHT();
- Category 12: [workspace_add_client, workspace_rm_client]
  - if(){ workspace_rm_client(); workspace_add_client();
- Category 13: [execlp, screen_shutdown]
  - screen_shutdown(); execlp();
- Category 14: [DefaultColormap, DefaultColormap, DefaultColormap, XAllocColor, XParseColor, XParseColor]
  - if(){ XParseColor(); DefaultColormap(); XParseColor(); DefaultColormap(); }else{ XAllocColor(); DefaultColormap(); }
  - XParseColor(); DefaultColormap(); if(){ XParseColor(); DefaultColormap(); }else{ XAllocColor(); DefaultColormap(); }
- Category 15: [XMapWindow, XReparentWindow]
  - if(){ XReparentWindow(); XMapWindow(); }
- Category 16: [XUngrabPointer, XUngrabServer, plugin_geometry_change]
  - XUngrabPointer(); XUngrabServer(); plugin_geometry_change();

- Category 17: [action_send_config, client_sizeframe, workspace_add_bypos]
  - if(){ client_sizeframe(); action_send_config(); workspace_add_bypos();
- Category 18: [focus_client, workspace_add_client]
  - if(){ workspace_add_client(); focus_client(); }else{
- Category 19: [XMaskEvent, XQueryPointer]
  - XQueryPointer(); XMaskEvent();
- Category 20: [XFreeStringList, XGetWMName, XmbTextPropertyToTextList, strdup]
  - if(){ XGetWMName(); }else{ XmbTextPropertyToTextList(); strdup(); XFreeStringList(); }
  - if(){ XGetWMName(); XmbTextPropertyToTextList(); strdup(); XFreeStringList(); }else{
  - XGetWMName(); if(){ XmbTextPropertyToTextList(); strdup(); XFreeStringList(); }else{
  - XGetWMName(); if(){ XmbTextPropertyToTextList(); strdup(); XFreeStringList(); }
  - XGetWMName(); XmbTextPropertyToTextList(); if(){ strdup(); XFreeStringList(); }else{
  - XGetWMName(); XmbTextPropertyToTextList(); if(){ strdup(); XFreeStringList(); }
- Category 21: [XGrabPointer, XGrabServer]
  - if(){ XGrabPointer(); } XGrabServer();
- Category 22: [FULLHEIGHT, FULLWIDTH]
  - FULLWIDTH(); if(){ FULLHEIGHT(); }else{
  - FULLWIDTH(); if(){ FULLHEIGHT(); }
- Category 23: [OPTIONAL_PARAM, pier_additem]
  - OPTIONAL_PARAM(); if(){ pier_additem(); }
- Category 24: [DefaultDepth, DefaultGC, XCreatePixmap, image_destroy, image_put]
  - XCreatePixmap(); DefaultDepth(); image_put(); DefaultGC(); image_destroy();

- Category 25: [XGrabServer, draw_winbox]
  - if(){ XGrabServer(); draw_winbox(); }
  - XGrabServer(); draw_winbox(); if(){
- Category 26: [DHEIGHT, DHEIGHT, DWIDTH, DWIDTH]
  - DWIDTH(); DHEIGHT(); DWIDTH(); DHEIGHT();
- Category 27: [XQueryPointer, draw_winbox]
  - XQueryPointer(); draw_winbox(); loop(){
- Category 28: [FULLHEIGHT, FULLHEIGHT, FULLWIDTH]
  - if(){ FULLWIDTH(); FULLHEIGHT(); }else{ FULLHEIGHT(); }
  - FULLWIDTH(); FULLHEIGHT(); if(){ FULLHEIGHT();
- Category 29: [focus_list_rm, focus_unfocus, workspace_add_client]
  - if(){ focus_unfocus(); } focus_list_rm(); workspace_add_client();
- Category 30: [XMapWindow, XUnmapWindow]
  - if(){ XUnmapWindow(); }else{ XMapWindow();
  - XUnmapWindow(); if(){ XMapWindow();
  - XUnmapWindow(); XMapWindow(); if(){
- Category 31: [DHEIGHT, DWIDTH, draw_winbox]
  - draw_winbox(); DWIDTH(); DHEIGHT();
- Category 32: [BlackPixel, BlackPixel, WhitePixel]
  - BlackPixel(); WhitePixel(); BlackPixel();
- Category 33: [XMapWindow, client_setstate]
  - XMapWindow(); client_setstate();
- Category 34: [RootWindow, XDeleteProperty]
  - if(){ XDeleteProperty(); RootWindow(); }

- Category 35: [XFindContext, pager_sizepaged]
  - XFindContext(); pager_sizepaged();
- Category 36: [FULLHEIGHT, FULLWIDTH, FULLWIDTH]
  - if(){ FULLWIDTH(); }else{ FULLWIDTH(); } FULLHEIGHT();
  - if(){ FULLWIDTH(); }else{ FULLWIDTH(); FULLHEIGHT(); }
  - FULLWIDTH(); if(){ FULLWIDTH(); FULLHEIGHT();
- Category 37: [DisplayHeight, DisplayWidth]
  - if(){ DisplayWidth(); } DisplayHeight();
  - if(){ DisplayWidth(); DisplayHeight(); }
  - DisplayWidth(); if(){ DisplayHeight(); }
- Category 38: [RootWindow, XCreateWindow]
  - if(){ XCreateWindow(); RootWindow();
- Category 39: [XSync, usleep]
  - if(){ XSync(); usleep();
- Category 40: [action_send_config, client_sizeframe]
  - if(){ client_sizeframe(); action_send_config(); }else{
  - if(){ client_sizeframe(); action_send_config(); }
  - client_sizeframe(); if(){ action_send_config(); }
  - client_sizeframe(); action_send_config(); if(){
- Category 41: [XMaskEvent, draw_winbox]
  - draw_winbox(); loop(){ XMaskEvent(); }
- Category 42: [image_destroy, image_frompixmap, image_scale]
  - image_frompixmap(); image_scale(); image_destroy();
- Category 43: [log_enter, log_exit]
  - log_enter(); if(){ log_exit();

- Category 44: [BlackPixel, DefaultColormap, DefaultColormap, XAllocColor, XParseColor]
  - if(){ XParseColor(); DefaultColormap(); BlackPixel(); }else{ XAllocColor(); DefaultColormap(); }
- Category 45: [plugin_unzoom_notify, plugin_zoom_notify]
  - if(){ plugin_zoom_notify(); }else{ plugin_unzoom_notify(); }
- Category 46: [XGrabServer, XUngrabServer]
  - if(){ XGrabServer(); XUngrabServer();