修士学位論文

題目

同時更新情報と呼出情報を用いて
機能実装クラス群を抽出する手法の提案

指導教員
井上 克郎 教授

報告者
檜皮 祐希

平成 19 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 18 年度 修士学位論文

同時更新情報と呼出情報を用いて機能実装クラス群を抽出する手法の提案

檜皮 祐希

内容梗概

ソフトウェア保守において，ソフトウェア理解や再利用のために特定の機能が実装されている箇所を抽出することが必要になる．その際に用いられる手法として Feature Location という研究がある．Feature Location の既存手法は，静的手法，動的手法，開発履歴情報を用いた手法の 3 つに分けることができるが，各々の手法にはそれぞれ異なる問題点があり，ソフトウェアの機能によっては適切に抽出できない場合がある．

本研究では，開発履歴情報を用いた手法と静的手法を組み合わせることにより，ある機能を実装しているクラス群を抽出する手法を提案する．具体的にはまず，開発履歴情報のうち同時更新情報を用いて特定の機能を実装しているクラス群の抽出を行う．次に，その抽出結果に対して，静的情報のうち呼出情報を用いて更新履歴情報から誤って抽出したクラス群の除去，及び，更新履歴情報で抽出できなかったクラス群の補完を行う．また，本手法を適用するためのシステムを作成し，作成したシステムを用いてオープンソースソフトウェアに対して手法の適用実験と評価を行い，手法の有用性を示した．

主な用語

Feature Location
ソフトウェア理解 (Software Comprehension)
ソフトウェア再利用 (Software Reuse)
同時更新 (Common Changes)
呼出関係 (Call Relations)

目 次

1 まえがき

ソフトウェアライフサイクルは設計・開発を行うソフトウェア開発とソフトウェア納品後の修正・拡張を行うソフトウェア保守に大別され，そのうちソフトウェア保守はソフトウェアライフサイクルのコストの大部分を占めている重要な部分である．ソフトウェア保守は，ソフトウェア理解とバグ修正や機能拡張などの実際の修正作業に分けることができ，実際に修正作業を行う前には，対象のソフトウェアについての理解が必須である．しかし，ソフトウェアの大規模化や複雑化により，ソフトウェア全体を理解するのには多大な時間やコストがかかる．そこで，ソフトウェア全体を理解するのではなく，これから修正する機能に関連する部分のみを理解することができれば，ソフトウェア理解にかかるコストを大きく削減することができると考えられる [11]．そのため，理解するべき修正に必要な特定の機能に関連する部分を特定する必要があるが，機能に関連する部分の特定にはソフトウェア全体の構成を知る必要があり，大規模かつ複雑なソフトウェアで行うのは困難である [26].

一方，ソフトウェア開発においては，大規模かつ高品質なソフトウェア製品を短期間で開発することが求められている．そうした要求を実現する方法の 1 つとして，再利用がある [23]．再利用とは，過去に作られたソフトウェア部品を同一システム内や他のシステムで用いることで，再利用を行うことにより高品質なソフトウェアを短期間で効率良く開発することができる．しかし，過去に作った他のソフトウェアの一部を再利用する際には，再利用に必要なその機能を実装している部分を抽出する必要があるが，ソフトウェア理解と同様に困難な作業である．

このように，ソフトウェア理解や再利用において，機能が実装されている部分を抽出することが必要になるが，これらは多大な時間やコストがかかる困難な作業である．そこで，ある機能が実装されている部分を効率的に抽出する手法として，Feature Location と呼ばれる手法が研究されており，静的手法 [5][36]，動的手法 [9][10][33][34]，開発履歴情報を用いた手法 [25] の 3 つに大別される．しかし，それぞれ手法によって，適切に抽出できない機能がある，ソフトウェアの知識が必要である，抽出に時間がかかる，などの問題点がある．

そこで本研究では，ある機能が実装されている部分を抽出する手法として，既存手法のうち，開発履歴情報を用いた手法と静的手法を組み合わせた手法を提案する．具体的には，まず開発履歴情報のうち同時更新情報を用いて，同時更新傾向が強いクラス群は同一の機能を実装していると仮定し，同時更新傾向が強いクラス群を抽出する．そして，静的情報のうち呼出情報を用いて，呼出関係の弱いクラスを除去することで誤って抽出したクラス群を除去し，抽出クラス群と特定の呼出関係があるクラスを追加することで開発履歴情報からは抽出できなかったクラス群の補完を行う．この手法により，既存手法における問題点を補い，高い精度で機能を実装しているクラス群を抽出することを目指す．

以降，2 節では本研究の背景となるソフトウェア理解，ソフトウェア部品の再利用，及び，Feature Location について説明し，3 節では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同時更新情報と呼出情報を用いて機能実装クラス群を抽出する手法について説明する．5 節ではその手法を実装したシステムについて述べる．6 節では提案手法の適用実験とその考察を行う．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　ソフトウェア理解と再利用

本節ではソフトウェア理解と再利用について説明した後，ソフトウェア理解と再利用を支援する手法の 1 つである Feature Location について説明する．

2.1　ソフトウェア理解

ソフトウェアライフサイクル *(Software Life Cycle)* は，**ソフトウェア開発** *(Software Development)* と**ソフトウェア保守** *(Software Maintenance)* に大別される．ソフトウェア開発とは，ユーザの要求をソフトウェア・プロダクトに変換する作業であり，要求分析・設計・コーディング・テスト等の工程を含んでいる．一方，ソフトウェア保守とは，納品後にソフトウェア・プロダクトに対して加えられるバグ修正や機能拡張作業である．ソフトウェア保守はソフトウェアライフサイクルにおけるコストの大部分を占め，ソフトウェア保守のコストを下げることは非常に重要であると考えられている．

ソフトウェア理解 *(Software Comprehension)* とは，ソフトウェア資産のソースコードを読み，そのソフトウェア資産が持つ機能がソースコードのどの部分にどのように実装されているかを解読する作業のことであり，ソフトウェア保守におけるバグ修正や機能追加などを行う前にソフトウェア理解を行うことが必要である．

ソフトウェア理解は，他人が書いたソースコードに対して行う場合，特に困難なものになる．しかし実際の開発現場では，開発者と保守担当者が異なることが多々あり，その際，引継ぎが十分に行われない，仕様が文書化されていない，ソフトウェアが変更されており仕様書通りでないなどの問題があり，ソフトウェア理解を困難にしている．さらに，ソフトウェアの大規模化，複雑化により，ソフトウェア全体の理解にかかるコストが増大し，ソフトウェア理解にかかるコストがソフトウェア保守の半分を占めるという報告もされている [7]. そのため，ソフトウェア理解を支援することはソフトウェアライフサイクルにおいて重要な要素である．

ソフトウェア理解の支援手法として，ソフトウェア全体の可視化 [29] など様々な研究がされているが，その中のひとつにソフトウェアの一部を理解するというものがある．ソフトウェア保守を行う上では，ソフトウェア全体についての理解よりも特定の保守作業に応じたソフトウェアの一部の詳細な理解のほうが重要であるとの報告もあり [11]，保守に関係する部分を抽出し，その部分のみを理解することができれば，ソフトウェア理解にかかるコストを大きく削減することができる．

そのため，ソースコード中で保守に関係する機能を実装している箇所を抽出する必要がある．

2.2 ソフトウェア再利用

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 [23]．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ことを再利用といい，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 [3][21]．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 [27] 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．

本研究では，再利用されることを前提として作成されたソフトウェア部品の再利用ではなく，あるプログラム中の部品として提供されていない特定の機能を含むモジュール群を他のソフトウェアで再利用することを考える．このとき，再利用する機能はひとつの部品として提供されていないため，再利用をする開発者は元のソフトウェア中から再利用するために必要な，機能を実装しているモジュール群を抽出する必要がある．

2.3 Feature Location

効率的にソフトウェアの持つ特定の機能がソースコード中で実装されている箇所を抽出する手法として，*Feature Location* についての研究がある [5][9][10][25][33][34][36]．**機能** *(feature)* とは「要求仕様書に明示的，あるいは暗黙に示されているソフトウェアの特徴」と定義されている [18]．Feature Location は，機能とソースコードの部分集合との対応を特定し，抽出する手法である．Feature Location を用いることで，保守作業や開発作業において，特定の機能を実装している箇所を抽出するコストを削減することができ，ソフトウェア理解，ソフトウェア再利用を支援することができる．

既存の Feature Location の手法は大きく静的手法，動的手法，開発履歴情報を用いた手法の 3 つに分けることができる．以下で，それぞれの手法の概要と問題点を述べる．

2.3.1 静的手法

静的手法では，対象とするソフトウェアのソースコードを静的に解析し，モジュールのコールグラフを作成し，そのグラフを解析することで特定の機能を実装している箇所を抽出する．Chen と Rajlich は，作成したコールグラフを開発者自身が辿り，機能を実装しているかを判定していくという手法を提案している [5]．Zhao らは，文書とソースコードの関連を求め，機能を実装しているモジュール群を文書から特定し，特定したモジュール群を元にコールグラフを用いて機能の抽出を行うという手法を提案している [36].

しかし，静的手法はいずれもソースコードの解析結果を元にして特定の機能を実装している箇所を見つけ出すため，実行経路が動的に決定される動的束縛を持つオブジェクト指向プログラムにそのまま適用する事はできない．動的束縛は，オブジェクトが起動するメソッドを変数の型ではなく実行時の実体をもとに決定するため，実際に呼び出されるメソッドがソースコードを解析しただけではわからないためである．

また，開発者自身が機能を実装しているかを判定する必要がある手法では，開発者にソフトウェアに対する知識が要求され，また，手法を自動化することが困難である．

2.3.2 動的手法

動的手法では，あらかじめテストケースを用意しておき，そのテストケースを入力として実行したときの呼び出されたモジュールの履歴を解析することで特定の機能を実装している箇所を抽出する．Wilde と Scully や Wong らは，機能を実行するテストケースと機能を実行しないテストケースの 2 種類を実行し，機能を実行するテストケースで実行されたモジュール群から機能を実行しないテストケースで実行されたモジュール群を除いたモジュール群を機能を実装している箇所としている [33][34]．Eisenbarth らは，テストケースとシナリオを

用意し，概念分析と呼ばれる手法を適用することで，シナリオとモジュールを関連付けし，機能を実装しているモジュール群を抽出する手法を提案している [9]．Eisenberg と Volder は，テストケースの実行履歴からモジュールの呼出関係を取得し，呼出回数などを元に機能との関連の強さを計算し，機能との関連を順位付けする手法を提案している [10]．

しかし，動的手法は最初にテストケースを作る必要があり，また，テストケースの品質に抽出の精度が左右されてしまうため慎重にテストケースを作成する必要があり，テストケース作成のコストが膨大になる．さらに，ソフトウェアを何度も実行する必要があるため，大規模なソフトウェアに適用する場合には非常に時間がかかる．

2.3.3 開発履歴情報を用いた手法

開発履歴情報を用いた手法では，版管理システム等での開発履歴情報から同時更新の傾向を解析することにより特定の機能を実装している箇所を抽出する．楠田らは，開発において同時に更新されることの多いモジュールは同一の機能を実装していると考え，同時に更新されることの多いモジュール同士をクラスタリングすることで，ソフトウェア内のモジュールを機能ごとに分類する手法を提案している [25]．

しかし，更新履歴情報を用いた手法は，開発履歴情報を蓄えてある必要があり，版管理システム等を使って開発履歴を管理していないソフトウェアには適用することができず，管理のされ方によっても精度が大きく変わってしまう．また，開発履歴のみを解析し，ソースコードには全く依存しないため，たまたま他の機能と同時に更新されたが実際には機能とは関係のないモジュールが多く抽出されてしまう可能性がある．

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，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 [1] [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）．

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

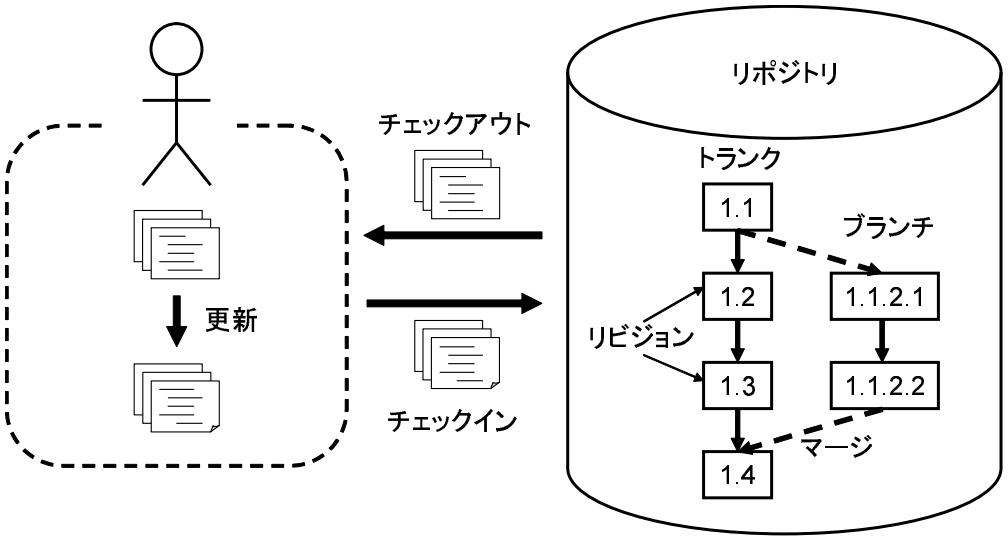

図 1: Checkout/Checkin Model

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．

3.2 主な版管理システム

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[32] や CVS[4] [12] といったシステムが標準で利用可能となっている．ClearCase[8] 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[19] や PVCS[20] 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 [14] 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[32] 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[4] [12] 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[31] や OpenBSD[28] 等のオペレーティングシステムの開発である．

Subversion ：

Subversion[30] は，CVS の持ついくつかの問題点を解決するために開発された版管理システムで，CVS の後継として注目されている．CVS ではプロダクトをファイル単位で管理していたが，Subversion では全てのプロダクトを 1 つのデータベースで管理する．Subversion では，CVS では考慮していなかったファイルの移動の処理を扱うことができたり，リポジトリ内のディレクトリを削除したりすることが可能になっている．またネットワークを介した利用を想定して開発されており，リポジトリへチェックインを行なう際は，新しいリビジョンを作成するのに必要な差分の情報だけが送られるため，効率が良い．

3.3 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 1 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，1 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 1 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 1 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 [16]．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 [38]．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　提案手法

本節では，開発履歴情報の 1 つである同時更新情報と静的情報の 1 つである呼出情報を用いて，ある機能を実装しているクラス群を抽出する手法について述べる．本研究では，インタフェースもクラスと同等に扱い，インタフェースも抽出する．以下で，クラスとしたときはインタフェースも含む．

同時更新情報としてクラス間で同時に更新される割合を表した更新傾向グラフを用いる．版管理システムを用いたソフトウェア開発において同じ機能を実装しているクラスは同時に更新されることが多いと仮定し，クラス間での同時更新される割合を解析し，機能を実装しているクラスと同時更新傾向が強いものを抽出することで機能を実装しているクラスを抽出する．また，呼出情報としてクラス間の呼出関係を表したコールグラフを用いる．呼出関係があるクラス間では呼出元の機能の一部を呼出先が実装している可能性が高いと考えられるため，機能を実装しているクラスからの呼出関係を解析することにより機能を実装している他のクラスを抽出する．

本手法では，まず機能を実装しているクラスを 1 つ開始クラスとして決定し，更新傾向グラフを用いて開始クラスと同時更新傾向が強いクラス群を抽出する．そして，その抽出結果からコールグラフを用いて開始クラスとの呼出関係の弱いクラスを除去し，特定の呼出関係のあるクラス群を補完することによりある機能を実装しているクラス群を抽出する．

本手法は，以下の 4 つの手順から構成されている．

1. 開始クラスの決定
2. 更新傾向グラフ，コールグラフの作成
3. 更新傾向グラフからの抽出
4. 抽出結果とコールグラフの比較

以下，これら 4 つについて詳しく説明する．

4.1　開始クラスの決定

抽出したい機能を実装しているクラスを 1 つ選ぶ．このとき選ぶクラスを開始クラスと呼び，以下の手順ではこの開始クラスを元にクラス群の抽出を行う．開始クラスの決定はユーザが任意の方法で選ぶことができる．

開始クラスの決定方法として，機能から連想されるキーワードを用いた検索，ソフトウェア部品の検索システムを用いるなどが考えられる．ソフトウェア部品検索システムとは，ソ

フトウェア部品特有の性質を利用した検索システムで，クラス名を用いたキーワード検索より機能の特徴を反映した開始クラスを選ぶことができる．

4.2 更新傾向グラフ，コールグラフの作成

ここでは，抽出で用いる更新傾向グラフとコールグラフを作成する．

4.2.1 更新傾向グラフの作成

更新傾向グラフ [25] とは，クラス間の同時に更新される割合を表現するグラフである．本研究では，1 回のチェックイン作業により更新されたクラス群を同時に更新されたクラス群と考える．グラフの頂点はクラスを表現する．頂点間の辺は有向辺であり，辺の始点が頂点 c_1，終点が頂点 c_2 であるとき，同時に更新される傾向を表現するメトリクスである $confidence_{c_1,c_2}$ という値を持っている．

$confidence$ メトリクス [37] は，ある 2 つのクラス c_1，c_2 について，c_1 が更新されたとき，c_2 がどの程度の割合で更新されているか，ということを表現する．$confidence$ メトリクスを以下の式で定義する．ソフトウェア中の全クラス集合を C とし，任意のクラス $c_1, c_2 \in C$ について，

$$confidence_{c_1,c_2} = \frac{c_1\text{と } c_2\text{が同時に更新された回数}}{c_1\text{が更新された回数}} \tag{1}$$

更新傾向グラフの作成では，まず，版管理システムのリポジトリに蓄積されている更新者，更新日時，更新時のコメントの情報を用いて同時に更新されたクラス群を求める．そして，求めた同時更新情報を用いて，クラス間の $confidence$ 値を求め，グラフとして表現する．

更新傾向グラフの作成の具体例を挙げる．クラス A，B，C，D が次のような順に更新されたとする．

1. A，B，C を同時に更新する
2. A，B を同時に更新する
3. C，D を同時に更新する
4. A，B を同時に更新する
5. A，C，D を同時に更新する
6. C，D を同時に更新する

このときの $confidence_{A,B}$ を考える．A が更新されているのは合計 4 回である．その 4 回のうち B も同時に更新されているのは 3 回である．したがって，$confidence_{A,B} = \frac{3}{4} = 0.75$

表 1: confidence 値の例

	A	B	C	D
A	1.0	0.75	0.5	0.25
B	1.0	1.0	0.33	0
C	0.5	0.25	1.0	0.75
D	0.33	0	1.0	1.0

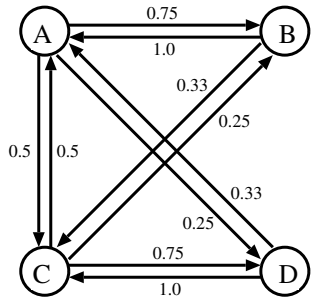

図 2: 更新傾向グラフの例

となる．逆に，$confidence_{B,A}$ を考えると，B が更新されているのは 3 回で，そのすべてにおいて A も同時に更新されている．したがって，$confidence_{B,A} = 1.0$ となる．このように $confidence_{c_1,c_2}$ と $confidence_{c_2,c_1}$ は必ずしも一致しない．同様に，他のクラス間に関しても $confidence$ 値を計算した結果は表 1 のようになる．表 1 の各セルは $confidence_{行,\,列}$ の値を表している．そして，図 2 でこれを更新傾向グラフとして表現した．

4.2.2 コールグラフの作成

コールグラフとは，クラス間の呼出関係を表現したグラフである．本研究では，コールグラフで呼出関係以外に，継承関係，実装関係も表現している．グラフの頂点はクラスを表現する．頂点間の辺は有向辺であり，クラス c_1 がクラス c_2 を呼び出しているとき，始点が頂点 c_1，終点が頂点 c_2 となる有向辺を引く．また，クラス c_1 がクラス c_2 を継承，または，クラス c_1 がインタフェース c_2 を実装しているときも同様に，始点が頂点 c_1，終点が頂点 c_2 となる有向辺を引く．有向辺は，クラス間の関係が呼出関係，継承関係，実装関係のいずれであるかの情報を保持している．

ソースコードを静的に解析し，クラス間の継承関係，実装関係，呼出関係を抽出し，グラフとして表現する．

```
class A extends X {
  public void a1() {
    ...
    B b = new B();
    b.b2();
    ...
  }

  public void a2() {
    ...
    C c = new C();
    c.c1();
    ...
  }
}
```

```
class B {
  public void b1() {
    ...
    C c = new C();
    c.c2();
    ...
  }

  public void b2() {
    ...
  }
}
```

```
class C {
  public void c1() {
    ...
  }

  public void c2() {
    ...
  }
}
```

```
class X {
  public void x1() {
    ...
    B b = new B();
    b.b1();
    ...
  }
}
```

図 3: ソースコード

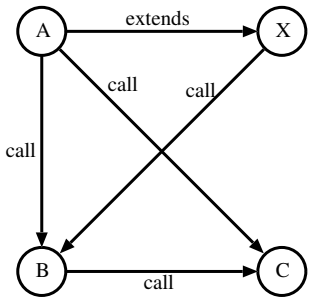

図 4: コールグラフの例

コールグラフの例を示す．図 3 のようなクラス A，B，C，X の 4 クラスがあったとする．クラス A はクラス X を継承しており，クラス B，C を呼び出している．クラス B はクラス C を呼び出している．クラス C はいずれのクラスも呼び出していない．クラス X はクラス B を呼び出している．これをコールグラフで表現すると図 4 のようになる．有向辺の “extends” は継承関係を表しており，“call” は呼出関係を表している．

4.3 更新傾向グラフからの抽出

更新傾向グラフを用いて，開始クラスとの同時更新傾向が強いクラス群を抽出する．同時更新されることが多いクラスは同じ機能を実装していると仮定することができるため，開始クラスとの同時更新傾向が強いクラス群を抽出することで，開始クラスと同じ機能を実装しているクラス群を抽出する．

抽出方法は，あらかじめ閾値 E_{th} を決め，以下のような手順で行う．開始クラスを c_{start}，

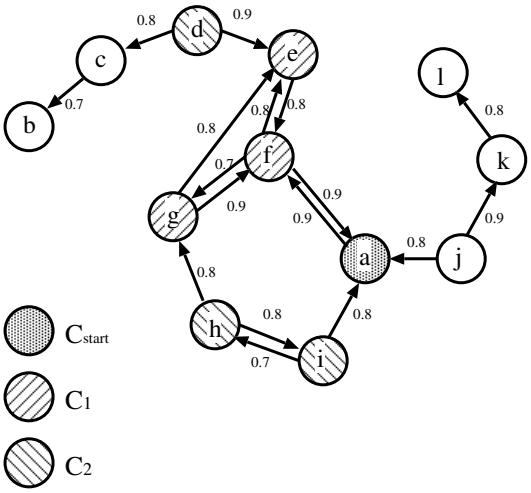

図 5: 更新傾向グラフからの抽出例

全クラス集合を C とする．

1. $(\forall c_i \forall c_j \in C) confidence_{c_i,c_j} < E_{th}$ となる有向辺を削除する．
2. $C_1 = \phi$，$C_2 = \phi$
3. c_{start} から到達可能な頂点を C_1 に加える．
4. 以下の 2 つの条件を満たすクラス c_i をクラス群 C_2 に加える．
   - c_{start}，$C1$ の各頂点のいずれかに到達可能
   - c_i を始点とする有向辺のうち，$confidence_{c_i,c_j}$ が最大となる辺の終点 c_j が c_{start}，C_1，C_2 に含まれる
5. 4. の条件を満たすクラスが存在しなくなるまで 4. を繰り返す．
6. c_{start}，C_1，C_2 の和集合を抽出クラス群とする．

閾値については，対象とするソフトウェアによって更新のされ方等が異なり閾値自体も異なってくる．また，抽出する機能によっても変わるため，抽出を行う利用者によって抽出された数を見たり，抽出結果をコールグラフにマッピングし結果を見ることにより判断する．

更新傾向グラフからの抽出の具体例を図 5 を用いて示す．図 5 は閾値 $E_{th} = 0.7$ とし，更新傾向グラフから E_{th} 以下の有向辺を削除したグラフである．開始クラス a から到達可能な頂点は e, f, g でこれら 3 つのクラスが C_1 となる．次に，a, e, f, g のいずれかの頂点に到達可能な頂点は d, h, i, j である．このうち，d は d を始点とする有向辺のうち $confidence$ が

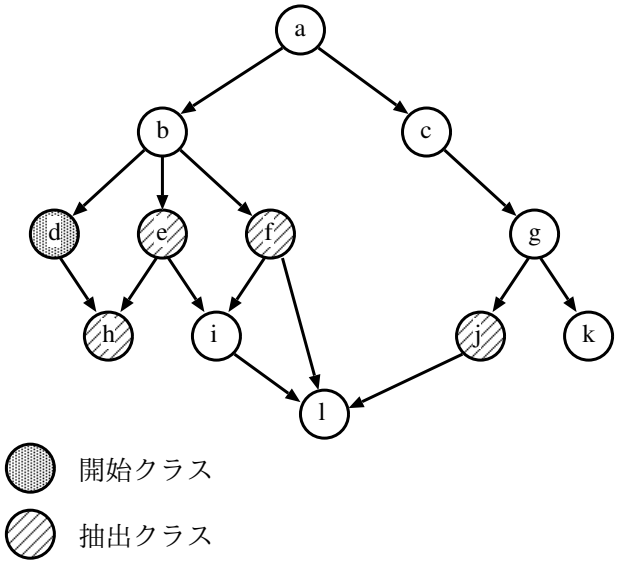

図 6: 開始クラスとの呼出関係が弱いクラスの除去例

最大となるのは，$confidence_{d,e} = 0.9$ となる e で，e は C_1 に含まれる．同様に，h，i を始点とする有向辺の $confidence$ が最大となる辺の終点は c_{start}，C_1，C_2 に含まれる．しかし，j を始点とする有向辺のうち $confidence$ が最大となるのは，$confidence_{j,k} = 0.9$ となる k で，k は c_{start}，C_1，C_2 に含まれない．そのため，j を除く 3 つのクラスが C_2 となる．したがって，このグラフから抽出されるクラス群は $\{a, d, e, f, g, h, i\}$ となる．

4.4 抽出結果とコールグラフの比較

コールグラフを用いて，開始クラスとの呼出関係が弱いクラスの除去，及び，特定の呼出関係があるクラスの補完を行う．まず，更新傾向グラフから抽出されたクラスをコールグラフにマッピングする．

4.4.1 開始クラスとの呼出関係が弱いクラスの除去

ここでは，開始クラスとの呼出関係が弱いクラスを除去する．開始クラスとの呼出関係が弱いクラスは，開始クラスと同じ機能を実装していないと考えることができる．ゆえに，開始クラスとの呼出関係が弱いクラスを除去することで，更新傾向グラフから誤って抽出されたクラスを除去することができる．

クラスの除去にはコールグラフ上の開始クラスからの抽出クラス距離を用いる．抽出クラス c_1 と c_2 間の抽出クラス距離 d_{c_1,c_2} はコールグラフの辺を無向辺と考え，c_1 から c_2 へ到達するとき抽出されていないクラスを少なくともいくつ通るかを表すものと定義する．ただし，

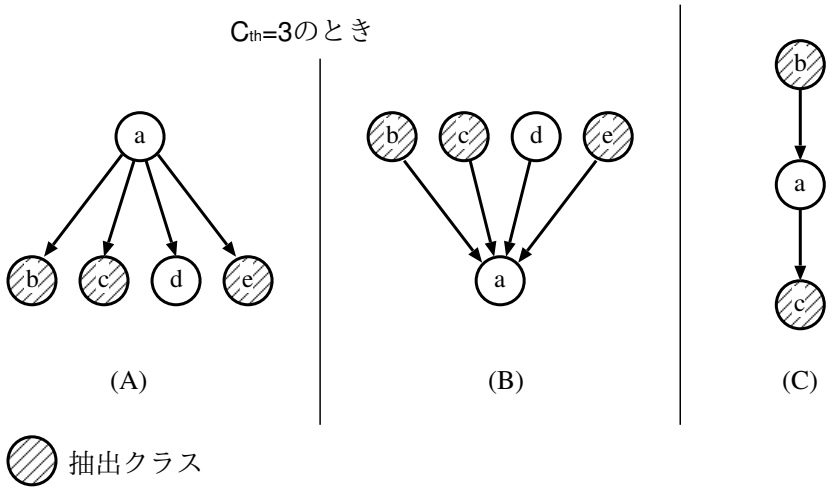

図 7: 特定の呼出関係のあるクラスの補完例

c_1 から c_2 へ到達不可能なときは，$d_{c_1,c_2} = \infty$ とする．例えば，図 6 の d と f の抽出クラス距離 $d_{d,f}$ は，頂点 d と f の最短経路 $d \to b \to f$ には抽出されていないクラス b が存在し 1 つの抽出されていないクラスを通る．また，その他のどのような経路を通っても少なくとも 1 つの抽出されていないクラスを通る．したがって，$d_{d,f} = 1$ となる．d と e の抽出クラス距離 $d_{d,e}$ は $d \to h \to e$ という経路を通ればすべてのクラスが抽出されているので，$d_{d,e} = 0$ となる．

この抽出クラス距離を用い，開始クラス c_{start} とクラス $c \in E$（E は抽出クラス群）について $d_{c_{start},c} \geq 2$ となる c を呼出関係が弱いクラスとして除去する．

図 6 の例では，開始クラスは d，抽出クラス群 $\{e, f, h, j\}$ である．開始クラス d と抽出クラス群の各クラスとの抽出クラス距離を求めると，$d_{d,e} = 0$，$d_{d,f} = 1$，$d_{d,h} = 0$，$d_{d,j} = 2$ となる．したがって，開始クラスとの抽出クラス距離が 2 となった j を除去する．

4.4.2 特定の呼出関係があるクラスの補完

ここでは，コールグラフを用いて特定の呼出関係があるクラス群を抽出することにより，機能を実装しているが更新傾向グラフからの抽出では抽出されなかったクラス群を補う．

まず，抽出クラスを継承しているクラス，抽出インタフェースを実装しているクラスは，元のクラスと同じ機能を実装していると考えられるので，それらのクラスを抽出クラス群に加える．

その上で，次の条件のいずれかを満たすクラスを抽出する．抽出には閾値 C_{th} を用いる．

- 呼び出しているクラスのうち C_{th} 個以上のクラスが抽出クラス（図 7(A)）

- 呼び出されているクラスのうち C_{th} 個以上のクラスが抽出クラス（図 7(B)）
- 呼び出しているクラスのいずれかが抽出クラス，かつ，呼び出されているクラスのいずれかが抽出クラス（図 7(C)）

図 7(A) のクラス a のように呼び出しているクラスのうち C_{th} 個以上が抽出クラスであるクラスは，その機能を統括しているクラスと考えられる．図 7(B) のクラス a のように呼び出されているクラスのうち C_{th} 個以上が抽出クラスであるクラスは，その機能に必要なライブラリを実装していると考えられる．図 7(C) のクラス a のように呼び出しているクラスのいずれかが抽出クラス，かつ，呼ばれているクラスのいずれかが抽出クラスであるクラスは，その機能の一部を実装していると考えられる．これらのクラスは更新傾向グラフからは抽出できなかったが，実際にはその機能の一部を実装していると考えられる．したがって，これらの条件のいずれかを満たすクラス群を抽出クラス群に加える．

これらの手順で最終的に抽出されたクラス群を機能を実装しているクラス群とする．

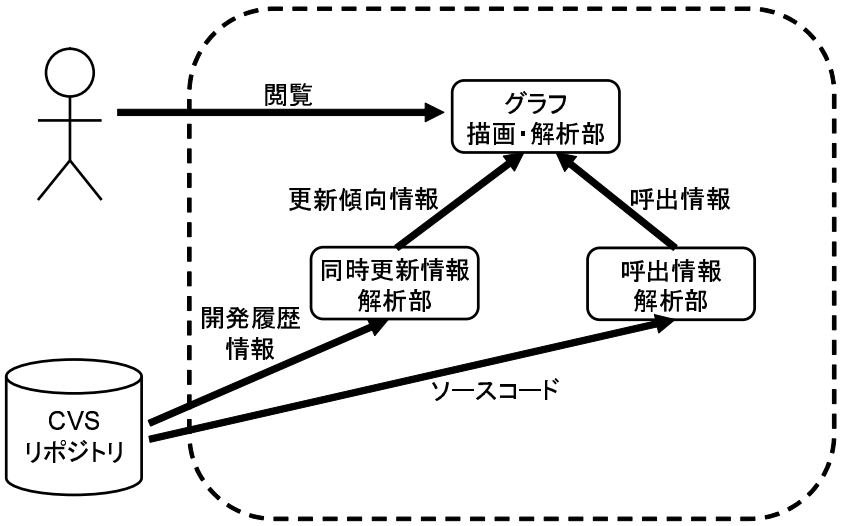

図 8: システムの概要

5 システムの実装

本節では，4 節で述べた同時更新情報と呼出情報を用いて機能を実装しているクラス群を抽出する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，オブジェクト指向言語 Java で記述されており，版管理システム CVS により管理されているソフトウェアを対象とする．

本システムは図 8 のように 3 つのサブシステムに分けることができる．同時更新情報解析部では，CVS のリポジトリから開発履歴を抽出し，同時更新されたクラスを解析し，クラス間の $confidence$ 値を求め，その情報を出力する．呼出情報解析部では，CVS から取得したソースコードを解析し，呼出関係，継承関係，実装関係を抽出し，クラス間の関係の情報を出力する．グラフ表示・解析部では，出力された 2 つの情報から更新傾向グラフ，コールグラフの 2 つのグラフの描画，及び，クラス群の抽出を行う．

以下では，本システムの 3 つのサブシステムについて詳細に説明していく．

5.1 同時更新情報解析部

同時更新情報解析部では，まず CVS リポジトリから開発履歴を抽出する．開発履歴の抽出には cvs2cl [13] を用いる．cvs2cl は，コミットログから変更履歴を記録している ChangeLog ファイルというファイルを生成することができる．ChangeLog ファイルは，1 回のコミット作業によって更新されたファイルの情報を記載している．cvs2cl を用いて ChangeLog を XML 形式で出力する．

次に，出力された情報には Java ソースコードファイル以外のテストデータファイルやドキュメントなどさまざまなファイルの更新情報が含まれるため，Java ソースコードファイルの更新情報のみを抽出し，同時更新されたソースコードファイルのリストを作成する．そして，作成したリストを基にクラス間の $confidence$ 値を計算し，更新傾向情報を求める．更新傾向情報には，すべてのクラスのリストとクラス間の $confidence$ 値が含まれている．

5.2 呼出情報解析部

呼出情報解析部では，ソースコードを解析し，クラス間の継承関係，実装関係，呼出関係を抽出する．解析には，Java ソースコードの XML 表現である JavaML [2] を用いる．

文献 [22] で記述されているクラス間の依存関係から，以下に示す継承関係，実装関係，呼出関係を抽出する．

1. 継承関係
    - クラスの継承
    - インタフェースの継承
2. 実装関係
    - インタフェースの実装
3. 呼出関係
    - コンストラクタの引数の型
    - コンストラクタの例外宣言
    - メソッドの戻り値の型
    - メソッドの引数の型
    - メソッドの例外宣言
    - フィールドの型
    - ローカル変数の型
    - キャスト
    - instanceof テスト
    - クラスの静的メンバアクセス
    - インスタンスの生成

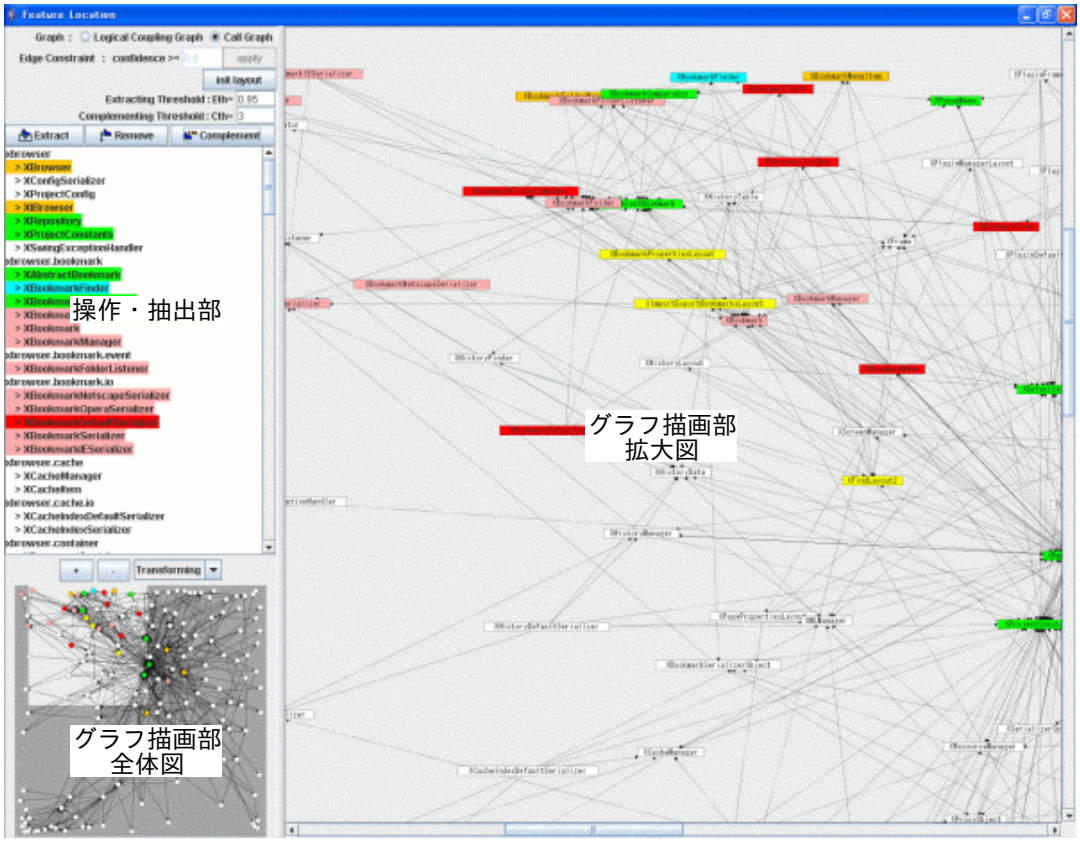

図 9: グラフ描画・解析部

- 配列の生成
- 例外処理

抽出した関係を基に呼出情報を作成する．呼出情報には，すべてのクラスのリストと，関係があるクラスの順序対とそのクラス間の関係の種類が含まれている．

5.3 グラフ描画・解析部

グラフ描画・解析部では，5.1，5.2 節で作成した更新傾向情報と呼出情報を用いてグラフの描画とクラス群の抽出を行う．

グラフの描画･解析には，グラフ描画ライブラリである JUNG（Java Universal Network/Graph Framework） [24] を用いる．JUNG では，グラフの視覚化や解析を行うことができる．

実装したツールの GUI を図 9 に示す．実装したツールは操作・抽出部とグラフ描画部の 2 つから構成されており，さらに，グラフ描画部は，全体図と拡大図の 2 つに分けられる．

操作・抽出部

操作・抽出部では，グラフの頂点であるクラス名をパッケージごとにリスト形式で表示する．また，更新傾向グラフ，コールグラフに対して以下の操作を行うことができる．

- 更新傾向グラフ，コールグラフの切り替え
  グラフ描画部に更新傾向グラフとコールグラフのどちらを表示するかを切り替える．

- 描画する辺の制限
  更新傾向グラフにおいて，$confidence$ 値が設定した値以上の辺のみを描画する．

- 頂点の配置の初期化
  グラフの頂点を再配置し，グラフを再描写する．本ツールでは，グラフのレイアウトアルゴリズムとして，力学的モデルによるレイアウト手法（Force-Directed Method）の 1 つである Fruchterman，Reingold の手法 [15] を用いている．この手法は頂点の配置位置が初期配置に大きく依存するため，配置位置が適切でないと判断したときは頂点の配置を初期化して，再配置する．

- 抽出の閾値の設定
  抽出の際に用いる E_{th}，C_{th} の値を設定する．

- 抽出の実行
  機能を実装しているクラス群を抽出する．クラスリストから開始クラスを選択し，段階的に抽出を行うことができる．“extract” ボタンでは 4.3 節の更新傾向グラフからの抽出，“remove” ボタンでは 4.4.1 節の開始クラスとの呼出関係が弱いクラスの除去，“complement” ボタンでは 4.4.2 節の特定の呼出関係があるクラスの補完を行うことができる．また，抽出されたクラスはクラスリスト，及び，グラフ描画部において色づけされて表示され，抽出された手順によって色分けされている．

グラフ描画部

グラフ描画部では，更新傾向グラフ，及び，コールグラフを描画する．全体図ではグラフ全体を表示し，拡大図では全体図で指定した範囲，拡大率に従い拡大表示を行う．拡大図には，頂点にはクラス名を表示しており，辺にマウスカーソルを持っていくことで，その辺の始点と終点のクラス名，更新傾向グラフであれば $confidence$ 値，コールグラフであれば関係の種類が表示される．

6 評価

提案手法における 2 つの閾値の決定方法に関する 2 種類の実験と，提案手法の妥当性を検討するための 1 種類の実験を 5 節で説明したシステムを用いて 2 つのソフトウェアの 3 つの機能を対象に行った．

実験対象として，オープンソースソフトウェアの XBrowser [35]，HttpUnit [17] を用いた．各ソフトウェアの概要は表 2 の通りである．

XBrowser はウェブブラウザであり，ブックマーク機能，ヒストリ機能の 2 つを抽出した．ブックマーク機能は，何度も訪れる Web サイトの URL を記録しておく機能で，ユーザが指定したページをブックマークとして保存することができ，ユーザはブックマークとして保存したページをブックマークリストから選択することで，ページに簡単にアクセスできるという機能である．ヒストリ機能は，ユーザが過去に閲覧したページの履歴を保持しておき，過去に開いたことのあるページへ戻ったり，その後，戻る前のページに戻ることを可能にする機能である．HttpUnit は Http による通信を擬似的に行うためのフレームワークで，Web での機能テストをサポートする．HttpUnit から JavaScript の実行に関する機能を抽出した．

これら 3 つの機能に対して次の 3 種類の実験を行った．

実験 1 E_{th} と抽出クラス数の関係

更新傾向グラフからの抽出で用いる閾値 E_{th} の値と更新傾向グラフからの抽出クラス数の関係を調査し，E_{th} の決定方法を検討した．

実験 2 C_{th} と抽出クラス数の関係

コールグラフでの補完で用いる閾値 C_{th} の値と抽出クラス数の関係を調査し，C_{th} の最適な値を検討した．

実験 3 各機能の抽出

それぞれの機能に対し，さまざまな開始クラスで抽出を実行し，抽出結果を考察した．

表 2: XBrowser，HttpUnit の概要

	XBrowser	HttpUnit
クラス数	171	164
総行数	28881	30223
総更新回数	165	690
総リビジョン数	1388	1712

実験に対する評価は，クラスごとに各機能を実装しているかをソースコードから判定し，各機能に対する正解集合を作成した上で，適合率，再現率，F 値を用いて評価を行った．適合率は，抽出結果として得られた集合中にどれだけ真の正解と適合した結果を含んでいるかという正確性の指標であり，再現率は真の正解集合のうち，どれだけ抽出できているかという網羅性の指標である．また，適合率，再現率は一方を上げればもう一方が下がるトレードオフの関係にあるため，そのトレードオフを表現する F 値という尺度を用いる．手法によって抽出されたクラスの数を抽出クラス数，ソースコードから実際に機能を実装していると判断した正解クラスの数を正解クラス数，手法によって抽出されたクラスのうち正解クラスと一致したクラスの数を該当クラス数とすると，適合率，再現率，F 値の計算は次の式で行う．

$$\text{適合率} = \frac{\text{該当クラス数}}{\text{抽出クラス数}} \tag{2}$$

$$\text{再現率} = \frac{\text{該当クラス数}}{\text{正解クラス数}} \tag{3}$$

$$F\ \text{値} = \frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}} \tag{4}$$

6.1 実験 1: E_{th} と抽出クラス数の関係

実験内容

更新傾向グラフからの抽出で用いる閾値 E_{th} と更新傾向グラフからの抽出クラス数の関係について次の方法でそれぞれの機能について実験を行った．

まず，開始クラスを 1 つ選び，E_{th} を 1 から 0 まで 0.01 刻みで変化させていき，それぞれの閾値における更新傾向グラフからの抽出クラス数，そのうち正解集合と一致した該当クラス数，適合率を求め，評価する．

開始クラスは，それぞれ“bookmark”，“history”，“javascript”で検索し上位に出現したクラスである xbrowser.bookmark.BookmarkFinder クラス（ブックマーク機能），xbrowser.history. HistoryFinder クラス（ヒストリ機能），com.meterware.httpunit.javascript.JavaScriptEngineFactory クラス（JavaScript 機能）とした．

実験結果を図 10〜12 に示す．

考察

実験の結果，どの機能に関しても，抽出クラス数が急激に増加する点がいくつか存在し，その間ではほとんど抽出クラス数が変化しないという傾向が見られた．そのため，閾値の候補は，急激に抽出クラス数が増加する点の直前の値に絞られる．また，抽出クラス数が全体

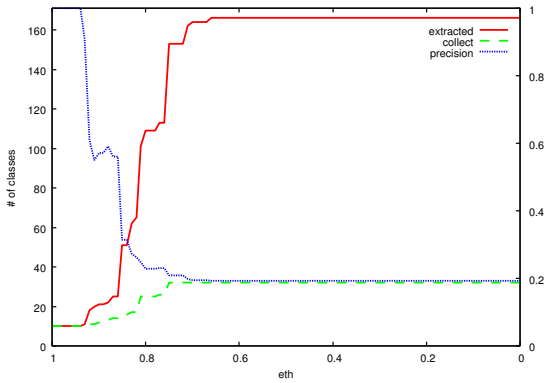

図 10: E_{th} と抽出クラス数の関係:ブックマーク

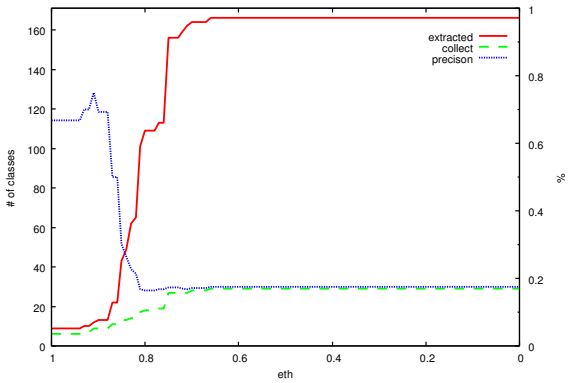

図 11: E_{th} と抽出クラス数の関係:ヒストリ

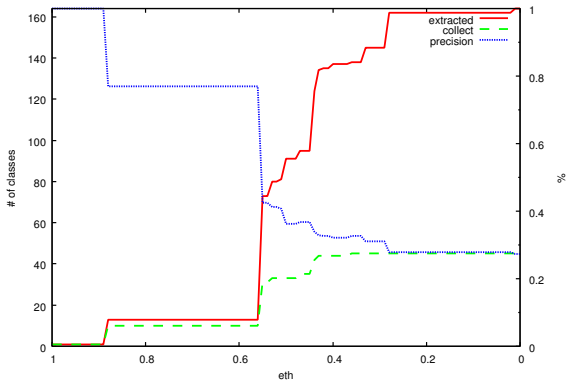

図 12: E_{th} と抽出クラス数の関係:JavaScript

の 5 割を超えているものは意味をなさないと考えられるので，閾値の候補をさらに絞ることができる．

適合率をみると，閾値が大きいときには，適合率は高いものの，閾値を下げるとともに抽出クラス中にノイズが多くなり，抽出クラス数が多くなるにつれ，適合率が著しく小さくなっていく．更新傾向グラフからの抽出の際のノイズは，コールグラフでの補完の精度を下げてしまう原因となり，また，あまりにノイズが多いとコールグラフでの誤って抽出してしまったクラスの除去を適切にできなくなる．しかし一方，抽出クラス数があまりに少なすぎると，コールグラフでの補完をすることができなくなる．そのため，10 個程度のクラスが抽出できている状態で，適合率ができる限り高いのが望ましい．

今回の実験では，ブックマーク機能は 1.0~0.9，ヒストリ機能は 1.0~0.85，Javascript 機能は 0.85~0.6 程度が最適であると考えられる．

実際の抽出では正解がどの程度あり，どのクラスが正解かはわからないが，10 クラス以上抽出されているうち，最初あるいは 2 番目の急激な増加の前を閾値とするのが最適だと考

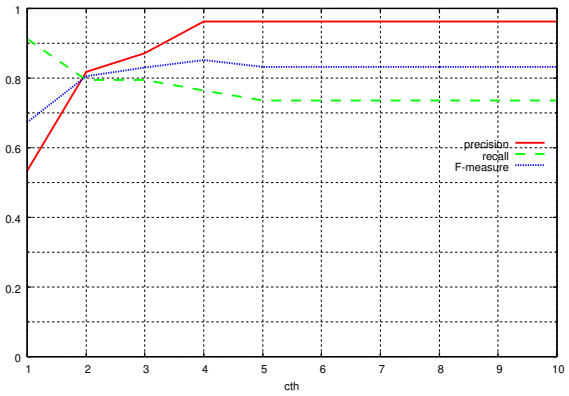

図 13: C_{th} と抽出クラス数の関係:ブックマーク

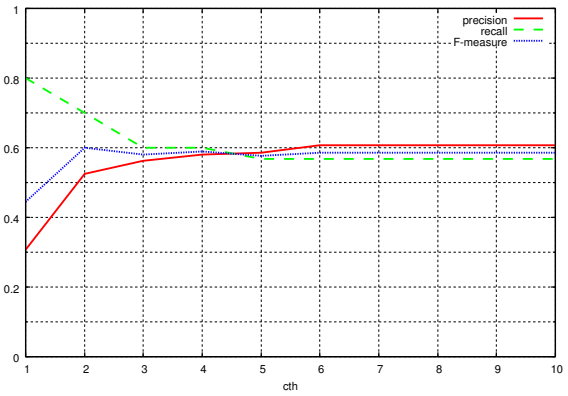

図 14: C_{th} と抽出クラス数の関係:ヒストリ

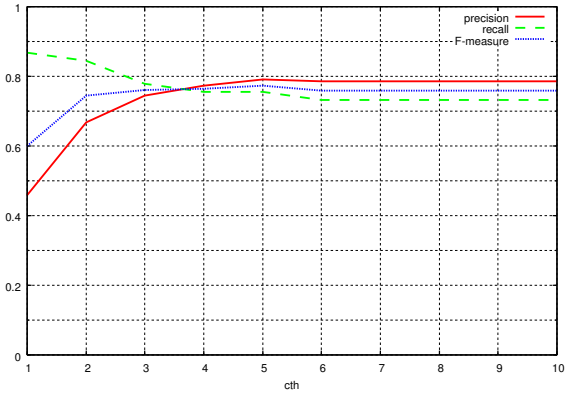

図 15: C_{th} と抽出クラス数の関係:JavaScript

えられる.

6.2 実験 2: C_{th} と抽出クラス数の関係

実験内容

コールグラフでの補完で用いる閾値 C_{th} と抽出クラス数の関係について次の方法でそれぞれの機能について実験を行った.

まず，開始クラスを 1 つ選び，C_{th} を 1 から 10 まで変化させて抽出を行う．そして，それぞれの閾値における適合率，再現率，F 値を求め，評価する.

開始クラスは，実験 1 同様，xbrowser.bookmark.BookmarkFinder クラス（ブックマーク機能），xbrowser.history.HistoryFinder クラス（ヒストリ機能），com.meterware.httpunit.javascript.JavaScriptEngineFactory クラス（JavaScript 機能）とした.

また，E_{th} の値は実験 1 の結果からブックマーク機能は 0.95，ヒストリ機能は 0.9，Javascript

機能は 0.85 とした.

実験結果を図 13~15 に示す.

考察

$C_{th} = 1$ のときは，再現率は高いもののは適合率は著しく低くなっている．以降，C_{th} を増やすにつれ，適合率は増加し，再現率は減少する．$C_{th} = 6$ 以降は適合率，再現率ともに変化しなくなった．$C_{th} = 1$ では，抽出クラスが呼び出しているクラス，抽出クラスを呼び出しているクラスをすべて抽出するため再現率は高いが，多くの呼び出しているだけで機能とは関係のないクラスを誤って抽出してしまうため，適合率が低くなる．

$C_{th} = 2$ 以降では，補完の条件を満たすクラスが減少していき，ノイズとともに正解クラスも抽出されなくなるため，適合率が増加するとともに再現率が若干減少してしまう．$C_{th} = 6$ 以降では,「呼び出しているクラスのうち C_{th} 個以上のクラスが抽出されているクラス」,「呼び出されているクラスのうち C_{th} 個以上のクラスが抽出されているクラス」の条件を満たすクラスがほとんどなくなり，また，閾値の増加によりこの条件を満たさなくなったときも,「呼び出しているクラスのいずれかが抽出されており，かつ，呼ばれているクラスのいずれかが抽出されているクラス」の条件を満たすことが多く，抽出クラスに変化がない．

適合率を見ると，ブックマーク機能では，$C_{th} = 2$ で急激に増加し，$C_{th} = 4$ でほぼ最大値になっている．ヒストリ機能，JavaScript 機能は，同様に $C_{th} = 2$ で急激に増加したあと，$C_{th} = 3$ でほぼ最大になっている．一方，再現率は適合率と逆の関係になっている．

この実験結果より，$C_{th} = 3$ では，適合率が十分に高い状態であると考えられ，それ以上の値にしてもそれほどノイズを取り除けない上，正解クラスも抽出できない可能性が大きくなる．

したがって，$C_{th} = 3$ が最適であると考えられる．

6.3 実験 3: 各機能の抽出

実験内容

各機能に対して，正解集合としたクラスそれぞれを開始クラスとしてクラス群の抽出を行い，適合率，再現率，F 値を用いて評価した．

E_{th} の値は，実験 1 の結果から更新傾向グラフから 10~20 クラス程度が更新傾向グラフから抽出されるように閾値を調整した．しかし，一部の開始クラスでは，E_{th} の値をどのように設定しても 10~20 クラス程度にならない場合があったため，その場合は抽出クラス数が 2 以上でできるだけ小さくなるものを閾値とした．また，C_{th} の値は，実験 2 の結果からすべての機能，開始クラスに対し $C_{th} = 3$ とした．

実験結果を表 3~5 に示す．表の抽出数は抽出されたクラスの数，該当数は抽出されたクラスのうち正解クラスと一致するクラスの数を表す．

考察:ブックマーク機能

一部のクラスで，E_{th} をどのように設定しても全体の半分以上が抽出されてしまうというクラスが存在した．特に，複数の機能を実装しているライブラリや，機能を管理しているクラスなどに多く，これらのクラスはさまざまなクラスと同時に更新されるため，多くのクラスが一度に抽出されたと考えられる．また，開発を通して単独で一度更新された後，一度も変更を加えられなかったため，任意のクラスとの $confidence$ 値が 0 となり，更新傾向グラフからの抽出を行うことができず，結果として抽出を行うことができなかったクラスもあった．このようなクラスは開始クラスにすることができないため，抽出を行うときは別のクラスを選ぶ必要がある．

抽出結果のうち，bookmark パッケージ内の 12 クラスを開始クラスとして抽出した結果は 12 クラス中 11 クラスで適合率，再現率ともに高い結果となった．XBookmarkDefaultSerializer クラスは他の 11 クラスより抽出された正解クラス数が少なく，再現率が低くなった．これは，他の 11 クラスは開発段階において同時更新されることが多く，互いに $confidence$ 値が高くなっているが，XBookmarkDefaultSerializer クラスだけは，別に更新されることが多かったため，更新傾向グラフからの抽出を十分にすることができず，結果として他の 11 クラスに比べて小さい値になったと考えられる．

次に，widget パッケージ内の “bookmark” という単語が入っているクラスを見ると，適合率が高いものの再現率は低くなっている．これらのクラスは GUI に関する機能を実装しており，抽出結果を見ると，ブックマーク機能の GUI に関するクラスしか抽出されていないことがわかった．これは，ブックマーク機能の中でも更にいくつかの細かな機能に細分化されその中でも GUI 機能の中で同時に更新が行われたためだと考えられる．機能の中でも GUI のような直接機能を実装していないクラスを開始クラスにした場合は，全体を捉えることができない可能性がある．

このように，ブックマーク機能を実装しているクラスのうち，核となる部分を実装しているクラスを開始クラスとした場合は，どのクラスを開始クラスとしても，同様の結果を得ることができ，また，高い適合率，再現率を得ることができた．

抽出結果のうち，BookmarkComparator クラスの手法の各手順における抽出結果を詳細に示したのが表 6 である．

表 6 を見ると，更新傾向グラフからの抽出では機能の一部分を高い適合率で抽出できているものの，全体の一部分しか抽出できていない．しかし抽出結果を見ると，bookmark パッ

表 3: ブックマーク機能の抽出

開始クラス	E_{th}	抽出数	該当数	適合率	再現率	F 値
xbrowser.XConfigSerializer	0.72	126	26	0.21	0.76	0.33
xbrowser.XIBrowser	0.90	18	6	0.33	0.18	0.23
xbrowser.XProjectConfig	0.83	73	23	0.32	0.68	0.43
xbrowser.XProjectConstants	0.90	19	6	0.32	0.18	0.23
xbrowser.XRepository	0.72	133	29	0.22	0.85	0.35
xbrowser.bookmark.XAbstractBookmark	0.85	30	26	0.87	0.76	0.81
xbrowser.bookmark.Xbookmark	0.95	26	26	1.00	0.76	0.87
xbrowser.bookmark.XBookmarkComparator	0.85	30	26	0.87	0.76	0.81
xbrowser.bookmark.XBookmarkFinder	0.95	31	27	0.87	0.79	0.83
xbrowser.bookmark.XBookmarkFolder	0.95	26	26	1.00	0.76	0.87
xbrowser.bookmark.XBookmarkManager	0.95	26	26	1.00	0.76	0.87
xbrowser.bookmark.event.XBookmarkFolderListener	0.95	31	27	0.87	0.79	0.83
xbrowser.bookmark.io.XBookmarkDefaultSerializer	0.83	26	21	0.81	0.62	0.70
xbrowser.bookmark.io.XBookmarkIESerializer	0.95	31	27	0.87	0.79	0.83
xbrowser.bookmark.io.XBookmarkNetscapeSerializer	0.95	31	27	0.87	0.79	0.83
xbrowser.bookmark.io.XBookmarkOperaSerializer	0.95	31	27	0.87	0.79	0.83
xbrowser.bookmark.io.XBookmarkSerializer	0.95	31	27	0.87	0.79	0.83
xbrowser.options.XSerializerOptionPage	0.85	9	2	0.22	0.06	0.09
xbrowser.plugin.XPluginContext	0.83	50	29	0.58	0.85	0.69
xbrowser.plugin.defaults.bookmarkpane.XBookmarkPane	0.00	1	1	1.00	0.03	0.06
xbrowser.plugin.defaults.bookmarkpane.XBookmarkTree	0.00	1	1	1.00	0.03	0.06
xbrowser.screen.XBookmarkPropertiesLayout	0.83	60	30	0.50	0.88	0.64
xbrowser.screen.XBrowserLayout	0.90	31	8	0.26	0.24	0.25
xbrowser.screen.XFindLayout2	0.95	11	4	0.36	0.12	0.18
xbrowser.screen.XImportExportBookmarksLayout	0.90	31	9	0.29	0.26	0.28
xbrowser.util.XBookmarkSerializerObject	0.85	102	26	0.25	0.76	0.38
xbrowser.util.XScreenManager	0.77	142	34	0.24	1.00	0.39
xbrowser.widgets.XBookmarkFolderComboBox	0.95	18	15	0.83	0.44	0.58
xbrowser.widgets.XBookmarkFolderMenu	0.95	19	15	0.79	0.44	0.57
xbrowser.widgets.XBookmarkMenu	0.80	154	34	0.22	1.00	0.36
xbrowser.widgets.XBookmarkMenuItem	0.95	19	15	0.79	0.44	0.57
xbrowser.widgets.XBookmarkTable	0.95	16	14	0.88	0.41	0.56
xbrowser.widgets.XHistoryTable	0.80	150	34	0.23	1.00	0.37
xbrowser.widgets.XPersonalToolBar	0.95	27	9	0.33	0.26	0.30

表 4: ヒストリ機能の抽出

開始クラス	E_{th}	抽出数	該当数	適合率	再現率	F 値
xbrowser.XConfigSerializer	0.72	128	17	0.13	0.57	0.22
xbrowser.XIBrowser	0.90	18	6	0.33	0.20	0.25
xbrowser.XProjectConfig	0.83	73	13	0.18	0.43	0.25
xbrowser.XProjectConstants	0.90	19	6	0.32	0.20	0.24
xbrowser.XRepository	0.72	133	19	0.14	0.63	0.23
xbrowser.history.XHistoryComparator	0.95	76	21	0.28	0.70	0.40
xbrowser.history.XHistoryData	0.90	33	18	0.55	0.60	0.57
xbrowser.history.XHistoryFinder	0.90	32	18	0.56	0.60	0.58
xbrowser.history.XHistoryManager	0.80	158	27	0.17	0.90	0.29
xbrowser.history.event.XHistoryListener	0.77	159	30	0.19	1.00	0.32
xbrowser.history.io.XHistoryDefaultSerializer	0.85	22	9	0.41	0.30	0.35
xbrowser.history.io.XHistorySerializer	0.85	16	9	0.56	0.30	0.39
xbrowser.options.XGeneralOptionPage	0.70	170	30	0.18	1.00	0.30
xbrowser.options.XSerializerOptionPage	0.85	9	3	0.33	0.10	0.15
xbrowser.plugin.XPluginContext	0.83	50	15	0.30	0.50	0.38
xbrowser.plugin.defaults.historypane.XHistoryPane	0.00	1	1	1.00	0.03	0.06
xbrowser.renderer.XRenderer	0.90	31	17	0.55	0.57	0.56
xbrowser.renderer.custom.XCustomRenderer	0.80	29	7	0.24	0.23	0.24
xbrowser.renderer.custom.js.XJSHistory	0.85	13	4	0.31	0.13	0.19
xbrowser.renderer.custom.js.XJSWindow	0.85	13	4	0.31	0.13	0.19
xbrowser.renderer.event.XRendererAdapter	0.80	164	29	0.18	0.97	0.30
xbrowser.renderer.event.XRendererListener	0.90	31	17	0.55	0.57	0.56
xbrowser.renderer.event.XRendererListenerSupport	0.65	170	30	0.18	1.00	0.30
xbrowser.screen.XBrowserLayout	0.90	31	17	0.55	0.57	0.56
xbrowser.screen.XFindLayout2	0.95	10	8	0.80	0.27	0.40
xbrowser.screen.XHistoryLayout	0.90	31	17	0.55	0.57	0.56
xbrowser.util.XHistorySerializerObject	0.85	102	17	0.17	0.57	0.26
xbrowser.util.XScreenManager	0.77	145	28	0.19	0.93	0.32
xbrowser.widgets.XBackForwardHistoryPopup	0.95	8	2	0.25	0.07	0.11
xbrowser.widgets.XHistoryTable	0.80	153	28	0.18	0.93	0.31

表 5: JavaScript 機能の抽出

開始クラス	E_{th}	抽出数	該当数	適合率	再現率	F 値
com.meterware.httpunit.BlockElement	0.80	42	30	0.71	0.67	0.69
com.meterware.httpunit.Button	0.70	32	21	0.66	0.47	0.55
com.meterware.httpunit.FormControl	0.53	92	39	0.42	0.87	0.57
com.meterware.httpunit.FormParameter	0.70	32	21	0.66	0.47	0.55
com.meterware.httpunit.FrameHolder	0.53	103	44	0.43	0.98	0.59
com.meterware.httpunit.GetMethodWebRequest	0.65	40	14	0.35	0.31	0.33
com.meterware.httpunit.HTMLElement	0.67	55	36	0.65	0.80	0.72
com.meterware.httpunit.HTMLElementBase	0.67	55	36	0.65	0.80	0.72
com.meterware.httpunit.HTMLElementScriptable	0.70	63	37	0.59	0.82	0.69
com.meterware.httpunit.HTMLPage	0.70	21	11	0.52	0.24	0.33
com.meterware.httpunit.HttpUnitOptions	0.36	141	44	0.31	0.98	0.47
com.meterware.httpunit.HttpUnitUtils	0.36	141	44	0.31	0.98	0.47
com.meterware.httpunit.ParsedHTML	0.65	41	31	0.76	0.69	0.72
com.meterware.httpunit.RequestContext	0.53	1	1	1.00	0.02	0.04
com.meterware.httpunit.ScriptException	0.80	47	35	0.74	0.78	0.76
com.meterware.httpunit.TableRow	0.70	43	32	0.74	0.71	0.73
com.meterware.httpunit.WebApplet	0.62	45	33	0.73	0.73	0.73
com.meterware.httpunit.WebClient	0.65	24	14	0.58	0.31	0.41
com.meterware.httpunit.WebForm	0.65	35	22	0.63	0.49	0.55
com.meterware.httpunit.WebFrame	0.56	96	43	0.45	0.96	0.61
com.meterware.httpunit.WebImage	0.65	78	39	0.50	0.87	0.63
com.meterware.httpunit.WebLink	0.54	87	38	0.44	0.84	0.58
com.meterware.httpunit.WebList	0.80	42	30	0.71	0.67	0.69
com.meterware.httpunit.WebRequest	0.60	40	14	0.35	0.31	0.33
com.meterware.httpunit.WebRequestSource	0.65	55	36	0.65	0.80	0.72
com.meterware.httpunit.WebResponse	0.60	45	34	0.76	0.76	0.76
com.meterware.httpunit.WebTable	0.59	59	38	0.64	0.84	0.73
com.meterware.httpunit.WebWindow	0.65	24	14	0.58	0.31	0.41
com.meterware.httpunit.dom.DomBasedScriptingEngineFactory	1.00	115	44	0.38	0.98	0.55
com.meterware.httpunit.dom.HTMLCollectionImpl	1.00	40	15	0.38	0.33	0.35
com.meterware.httpunit.dom.HTMLDocumentImpl	1.00	40	15	0.38	0.33	0.35
com.meterware.httpunit.dom.HTMLElementImpl	1.00	40	15	0.38	0.33	0.35
com.meterware.httpunit.dom.NodeImpl	1.00	40	15	0.38	0.33	0.35
com.meterware.httpunit.dom.ScriptingSupport	1.00	40	15	0.38	0.33	0.35
com.meterware.httpunit.javascript.JavaScript	0.80	41	32	0.78	0.71	0.74
com.meterware.httpunit.javascript.JavaScriptEngineFactory	0.80	47	35	0.74	0.78	0.76
com.meterware.httpunit.javascript.ScriptingEngineImpl	1.00	115	44	0.38	0.98	0.55
com.meterware.httpunit.parsing.DocumentAdapter	0.80	21	11	0.52	0.24	0.33
com.meterware.httpunit.parsing.NekoDOMParser	0.60	85	41	0.48	0.91	0.63
com.meterware.httpunit.parsing.ScriptFilter	0.80	14	7	0.50	0.16	0.24
com.meterware.httpunit.parsing.ScriptHandler	1.00	96	42	0.44	0.93	0.60
com.meterware.httpunit.scripting.ScriptableDelegate	0.70	47	33	0.70	0.73	0.72
com.meterware.httpunit.scripting.ScriptingEngine	0.70	47	33	0.70	0.73	0.72
com.meterware.httpunit.scripting.ScriptingEngineFactory	0.80	47	35	0.74	0.78	0.76
com.meterware.httpunit.scripting.ScriptingHandler	1.00	115	44	0.38	0.98	0.55

表 6: ブックマーク機能の抽出例:xbrowser.bookmark.BookmarkComparator

	抽出数	該当数	適合率	再現率	F 値
全体	30	26	0.87	0.76	0.81
更新傾向グラフでの抽出	13	11	0.85	0.32	–
コールグラフでの除去	1	0	0.00	–	–
コールグラフでの補完	18	15	0.83	0.44	–

表 7: ヒストリ機能の抽出例:xbrowser.history.HistoryFinder

	抽出数	該当数	適合率	再現率	F 値
全体	32	18	0.56	0.60	0.58
更新傾向グラフでの抽出	13	9	0.69	0.30	–
コールグラフでの除去	0	0	0.00	–	–
コールグラフでの補完	19	9	0.47	0.30	–

ケージ内の 12 クラス中 11 クラスが抽出されており，ブックマーク機能の中心的な部分をほぼ抽出できている．

さらに，コールグラフを用いることで，誤って抽出したクラスの一部を取り除くことができ，また，更新傾向グラフでは抽出できなかったクラスの多く補うことができた．

このように，同時更新情報に呼出情報を組み合わせることで，機能を実装しているクラス群を高い適合率，再現率で抽出することができた．

考察:ヒストリ機能

開始クラスによる抽出結果の違いはブックマーク機能の抽出と同じ傾向が見られたが，ブックマーク機能と比較して全体的に適合率，再現率ともに低い値となった．

抽出結果のうち，HistoryFinder クラスの手法の各手順における抽出結果を詳細に示したのが表 7 である．

ブックマークと比べ更新傾向グラフでの適合率が低くなっており，また，コールグラフでの除去ができていない．ヒストリ機能は，開いたページの履歴を取得する際，他の多くの機能と直接関係しており，開発や保守においてヒストリ機能を実装，修正する際，同時に他の機能を実装しているクラスも更新することが多く，ヒストリ機能を実装していないクラスと

表 8: Javascript 機能の抽出例:com.meterware.httpunit.javascript.JavaScriptEngineFactory

	抽出数	該当数	適合率	再現率	F 値
全体	47	35	0.75	0.78	0.76
更新傾向グラフでの抽出	13	10	0.77	0.22	–
コールグラフでの除去	0	0	0.00	–	–
コールグラフでの補完	34	25	0.74	0.56	–

の間の confidence も高くなったため抽出されてしまったためと考えられる．また，それらのクラスは直接ヒストリ機能との間に呼出関係があったため，除去することができなかった．そのため，コールグラフでの補完において誤って抽出したクラスを元にさらに別の誤ったクラスを抽出してしまい，全体の適合率が下がったと考えられる．

再現率に関しては，直接ヒストリ機能には関係ないが，一部でヒストリを参照しているようなクラスを主に抽出することができなかった．しかし，ヒストリ機能の核となる部分を実装しているクラス群の多くを抽出することができており，バグ修正等の際のソフトウェア理解においては，手法は有効であると考える．

考察:JavaScript 機能

javascript パッケージ内の 3 クラスのうち閾値を適切に決めることができなかった 1 クラスを除く 2 クラスから高い適合率，再現率でクラスを抽出することができ，また，JavaScript を含む script について実装している scripting パッケージ内のクラスからの抽出も概ね高い適合率，再現率を得ることができた．

JavaScriptEngineFactory クラスの手法の各手順における抽出結果を詳細に示した表 7 と抽出結果から，前の 2 つの実験同様に更新傾向グラフから JavaScript に関する核となるクラスの大部分を抽出し，コールグラフで，抽出できなかったクラスと周辺の機能を実装しているクラスを補完できていることがわかった．

以上，3 つの機能抽出の実験結果から，同時更新情報と呼び出し情報を用いることで，開始クラスを適切に決めることができれば機能実装クラス群を高い精度で抽出できることを確認できた．

7　まとめ

本研究では，ある機能が実装されているクラス群を抽出する手法として，開発履歴情報を用いた手法と静的手法を組み合わせる手法を提案した．具体的には，開発履歴情報として，クラス間の同時更新の傾向を表現する更新傾向グラフを用い，同時に更新されるものが多いクラスを抽出した．さらに，静的情報として，クラス間の呼出関係を表現するコールグラフを用い，開始クラスとの呼出関係が弱いクラスの除去と特定の呼出関係があるクラスの補完を行った．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実験を行った結果，同時更新情報と呼出情報を組み合わせることで，高い精度で特定の機能を実装しているクラス群を抽出できることを確認した．

今後の課題として，さらなる手法の改良，既存手法との比較実験，クラス単位以外の粒度への拡張が挙げられる．

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] 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, 1999.

[2] Greg J. Badros. JavaML: A Markup Language for Java Source Code. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*, pp. 159–177, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[3] 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, 1992. ACM Press.

[4] 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.

[5] Kunrong Chen and Václav Rajlich. Case Study of Feature Location Using Dependence Graph. In *IWPC '00: Proceedings of the 8th International Workshop on Program Comprehension*, p. 241, Washington, DC, USA, 2000. IEEE Computer Society.

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

[7] T. A. Corbi. Program understanding: challenge for the 1990's. *IBM Syst. J.*, Vol. 28, No. 2, pp. 294–306, 1989.

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

[9] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating Features in Source Code. *IEEE Trans. Softw. Eng.*, Vol. 29, No. 3, pp. 210–224, 2003.

[10] Andrew David Eisenberg and Kris De Volder. Dynamic Feature Traces: Finding Features in Unfamiliar Code. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pp. 337–346, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Katalin Erdös and Harry M. Sneed. Partial Comprehension of Complex Programs (enough to perform maintenance). In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, pp. 98–105, Washington, DC, USA, 1998. IEEE Computer Society.

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

[13] Karl Fogel, Melissa O'Neill, and Martyn J. Pearce. cvs2cl.pl: CVS-log-message-to-ChangeLog conversion script, 2002. `http://www.red-bean.com/cvs2cl/`.

[14] 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, 1997. Springer-Verlag.

[15] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, Vol. 21, No. 11, pp. 1129–1164, 1991.

[16] 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, 2003. IEEE Computer Society.

[17] HttpUnit. `http://httpunit.sourceforge.net/`.

[18] IEEE Std 1008-1987, IEEE Standard for Software Unit Testing.

[19] Microsoft Inc. Visual SourceSafe. `http://msdn.microsoft.com/ssafe/`.

[20] Serena Software Inc. PVCS Version Control Software. `http://www.serena.com/pvcs_version_control_software.html`.

[21] 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, 1992.

[22] 岩田英丈, 阿萬裕久, 山田宏之. 変更履歴情報に着目した依存関係分析. 信学技報, 第 106 巻 of *KBSE2006-26*, pp. 7–12, 2006.

[23] 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.

[24] JUNG - Java Universal Network/Graph Framework. `http://jung.sourceforge.net/`.

[25] 楠田泰三. メソッドの同時更新履歴を用いたクラスの機能別分類法. Master's thesis, 大阪大学 情報科学研究科, 2006.

[26] John J. Marciniak, editor. *Encyclopedia of software engineering*. John Wiley & Sons, 1994.

[27] 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, 1997. ACM Press.

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

[29] Blaine A. Price, Ian S. Small, and Ronald M. Baecker. A taxonomy of software visualization. In *Proc. 25th Hawaii Int. Conf. System Sciences*, 1992.

[30] Subversion. `http://subversion.tigris.org/`.

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

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

[33] Norman Wilde and Michael C. Scully. Software reconnaissance: mapping program features to code. *Journal of Software Maintenance*, Vol. 7, No. 1, pp. 49–62, 1995.

[34] W. Eric Wong, Joseph R. Horgan, Swapna S. Gokhale, and Kishor S. Trivedi. Locating Program Features using Execution Slices. In *ASSET '99: Proceedings of the 1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, pp. 194–203, Washington, DC, USA, 1999. IEEE Computer Society.

[35] XBrowser. `http://xbrowser.sourceforge.net/`.

[36] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. SNIAFL: Towards a Static Non-Interactive Approach to Feature Location. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pp. 293–303, Washington, DC, USA, 2004. IEEE Computer Society.

[37] Thomas Zimmermann, Stephan Diehl, and Andreas Zeller. How History Justifies System Architecture (or Not). In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution*, pp. 73–83, Washington, DC, USA, 2003. IEEE Computer Society.

[38] 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, 2004. IEEE Computer Society.