特別研究報告

題目

関数の変更履歴と呼出し関係に基づいた
開発履歴理解支援システムの実現

指導教官
井上 克郎 教授

報告者
中山 崇

平成 16 年 2 月 19 日

大阪大学 基礎工学部 情報科学科

平成 15 年度 特別研究報告

関数の変更履歴と呼出し関係に基づいた開発履歴理解支援システムの実現

中山 崇

内容梗概

ソフトウェアの開発を行う際には, 開発されたプロダクトを効率良く管理する為のリビジョン管理システムが用いられており, このシステムに存在するリポジトリと呼ばれるデータベースには将来の開発に活用することの出来る開発履歴情報が多く含まれる. ソフトウェア開発において, この履歴情報は以前の開発についての理解や, 新たな開発の手助けとなることが期待されるが, 開発が進むにつれてそれらの情報は膨大になるため, 必要な情報を的確に取得することは容易ではない.

そこで本研究では開発蓄積情報を取得する為のソフトウェア開発支援環境の構築を行う. 具体的には, リポジトリに蓄積されたソフトウェアの開発履歴を解析し, 開発者が必要とする開発履歴情報の検索・閲覧を行うシステムの試作を行なった. さらに実際のソフトウェア開発で用いられた版管理システムの開発履歴情報を用いて本システムの評価を行なった. その結果, 本システムを用いることで, 開発者は有益な情報を入手することが可能となり, 現状の問題点が解決されることが確認できた. このことにより, ソフトウェア開発者の負担が軽減され, ソフトウェア生産性の向上が期待できる.

主な用語

リビジョン管理 (Revision Control)
クロスリファレンス (Cross Reference)
リポジトリ (Repository)
検索 (Search)

目 次

1　はじめに

オープンソースソフトウェアの開発規模の増大に伴い, その開発形態は多人数化, 分散化している. 大規模なオープンソースソフトウェア開発では, 複数の開発者が互いにソースコードを共有しながら同時に一つの開発作業に携わることが一般的になりつつある. またインターネットに代表されるネットワーク環境の発展に伴い, 分散した多くの開発者が異なる場所で開発作業を行うことも多い.

このように複雑化しているソフトウェアシステムを効率よく管理する為, 近年のオープンソースソフトウェア開発では, リビジョン管理システムや, 電子メールを用いたメーリングリストシステムを用いることが多くなっている. リビジョン管理システムは, プロダクトの開発履歴をリポジトリと呼ばれるデータベースに格納して管理する. メーリングリストでは, 開発者相互の意思疎通や進捗状況の報告などが行なわれる. これらのシステムでは開発者が行なったプロダクトへの変更履歴や送信した電子メールは全て個別のアーカイブとして保存されており, その履歴の中には, 将来の開発に活用することの出来る情報が多く蓄積されている.

ソフトウェアを再利用する際, これらのアーカイブを閲覧することによって, 以前の開発についてより深い理解が得られ, 開発の手助けになることが知られている [18]. 具体的には, あるファイルの修正点について, 変更内容からだけではその修正意図について完全な理解が出来ないときに, その修正と関連のあるメールアーカイブを閲覧することで, 修正内容に対するより深い理解を得ることが出来る. しかし, 蓄積された膨大な情報の中から, 開発者が必要とする情報を的確に取得することは容易ではない. 開発者が必要とするプログラムは複数のファイルに分かれて蓄積されている可能性もある. 例えば, ある機能の欠陥を修正する為に, 一見無関係に見える複数のファイルを同時に修正した, などという状況も考えられる. このため, 開発者はどのファイルの履歴を見れば良いかを知ることは困難となる.

本研究ではこの問題を解決する為のソフトウェア開発支援環境の構築を行う. 具体的には, リポジトリに蓄積されたソフトウェアの開発履歴を解析し, 開発者が必要とする開発履歴情報の検索・閲覧を行うシステムの試作を行う. 本システムは, 開発履歴情報の特定の為のリビジョン検索, 及びヒストリイベント検索, そしてソフトウェア開発の具体的な手法を閲覧する為のリビジョン関係を考慮した関数クロスリファレンサなどを持つシステムであり, これらの機能を提供することでオープンソースソフトウェア開発の支援を行う. 例えば, リビジョン検索やヒストリイベント検索を用いての修正箇所の特定や, 関数クロスリファレンサを用いての実装の詳細の理解における支援を行う.

また, 実際のオープンソースソフトウェア開発で用いられたリポジトリを用いて本システムの評価を行なった. 具体的には, 実際のソフトウェア開発で蓄積された開発履歴情報を用いて, 過去の開発情報の検索・閲覧及び理解が出来るか実験した. その結果, 本システムを用い

ることで, 開発者は有益な情報を入手することが可能となり, 現状の問題点が解決されることが確認できた. このことにより, ソフトウェア開発者の負担が軽減され, ソフトウェア生産性の向上が期待できる.

以降,2 節ではオープンソースソフトウェア開発とその開発環境, 及びその問題点について述べ, それから既存の開発履歴閲覧システムとその問題点について述べる.3 節ではシステムの概要と設計について述べ,4 節ではそのシステムの実装について述べる.5 節では本システムに対して検証を行う. 最後に 6 節で本研究のまとめと今後の課題について述べる.

2　オープンソースソフトウェア開発

本節では，オープンソースとその開発環境について触れ，オープンソースソフトウェア開発環境の持つ問題点を説明する．

2.1　オープンソースソフトウェア

開発中のソースコードやドキュメント等のプロダクトを広く公開して複数の開発者が並列的にソフトウェアの開発作業を行う開発手法はオープンソースソフトウェア開発と呼ばれ[26][7]，高品質で多機能なソフトウェアを開発できるとして注目を集めている．そのオープンソースソフトウェア開発によって開発されたソフトウェアをオープンソースソフトウェアと言う．

オープンソースソフトウェア開発では，世界中に分散した各開発者が，インターネットに代表される大規模ネットワークを使って開発作業を行う．そのため，開発者はいつでも自由に開発作業に参加することが可能である．FreeBSD[21] や Linux[12]，GNU[8]，Apache[1] 等は，オープンソースソフトウェアの代表である．

2.2　開発環境

オープンソースソフトウェア開発では，各開発者がそれぞれ分散して並列的に開発作業を行うことが可能である．その一方で，開発中のソースコードやドキュメント等のプロダクトを広く公開するため，それらの管理を行う必要がある．そこで，オープンソースソフトウェア開発に参加する開発者は，オープンソースソフトウェア開発環境と呼ばれる環境の中でプロダクトの管理を行う．

オープンソースソフトウェア開発環境の構成例を図 1 に示す．オープンソースソフトウェア開発環境は，一般に複数の既存システムから構成される．図 1 の構成例の場合，ソースコードやドキュメント等のプロダクトは，版管理システム [10] の一つである CVS(Concurrent Versions System)[3] [11][27] を用いて管理される．それらのプロダクトは，rsync や ftp を利用して，各開発者に複製，配布される．また，開発者間で相互に行われる意志疎通の手段として，電子メールやメーリングリストが用いられる．その内容はアーカイブとして保存され，WWW(World Wide Web) を用いた検索エンジンによって自由に検索や閲覧が可能である．開発者からのバグ報告等フィードバックは，GNATS(GNU Problem Report Management System) を用いたバグデータベースによって管理される．

以下では，これらのシステムの中から，オープンソースソフトウェア開発で広く用いられる版管理システムと電子メールについて説明する．

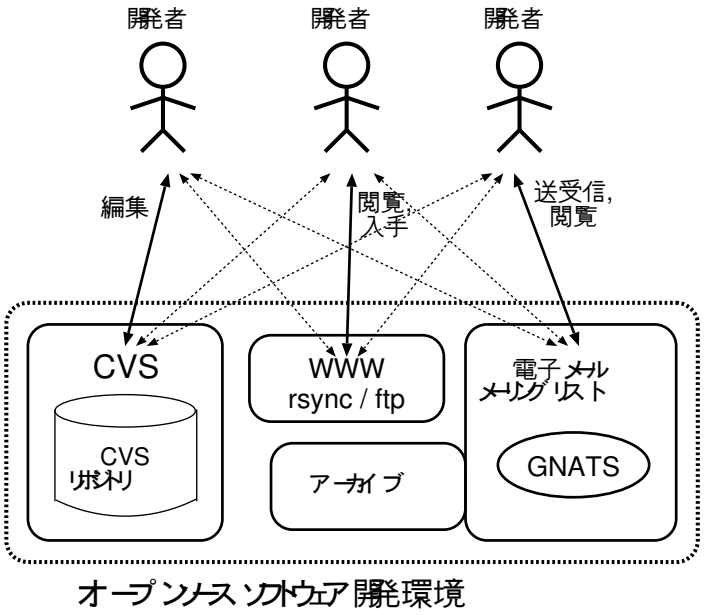

図 1: オープンソースソフトウェア開発環境の構成例

2.2.1 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．

各プロダクト (ソースコード，リソースなど) 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，作成日時やメッセージログ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為に，開発者はシステムに依存したオペレーション (operation) を利用する必要がある．

版管理手法を述べるにあたり，その基礎となるモデルが数多く存在する [22][20] ．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ことをリビジョンと呼ぶことにする．

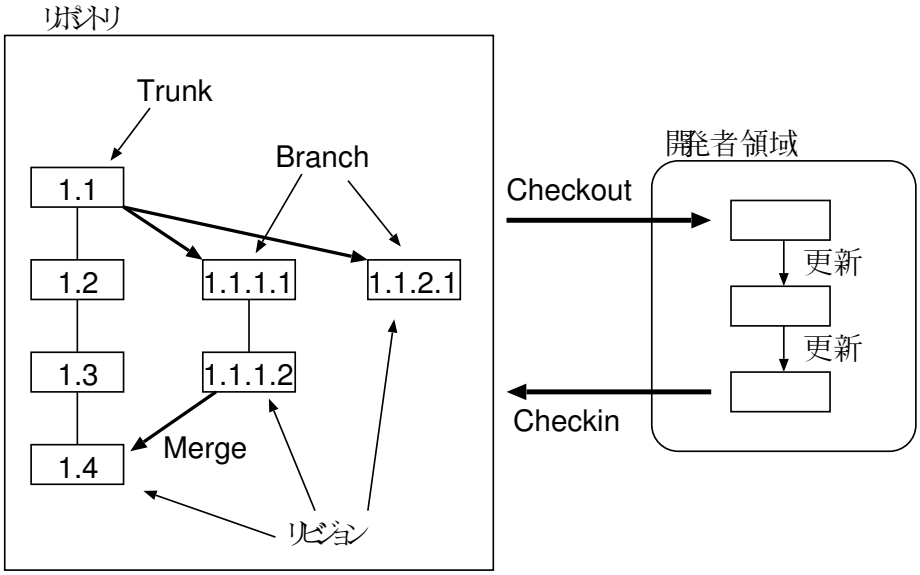

図 2: Checkout/Checkin Model

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 2 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．開発者のそれらのファイルを直接操作するのでなく，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う．リポジトリより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ことになる．しかし，過去のリビジョンに遡り，別の工程で開発を行う場合 (例えば，デバッグ) 等の為，リビジョン列を分岐させるには，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ことを**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ことから リビジョンツリー (RevisionTree) という.

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[25] や CVS[3][11] といったシステムが標準で利用可能となっている．ClearCase[19] のように商用ものも存在する．また，UNIX 系 OS だけではなく，Windows 系 OS においても，SourceSafe[14]

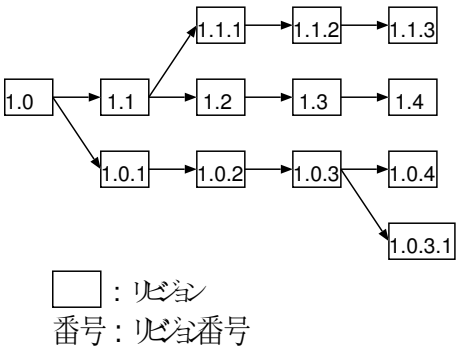

図 3: リビジョンツリーの例

や PVCS[13] 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 [17] も存在する．

ここでは，版管理システムのうちのいくつかを紹介する．

- RCS

RCS[25] 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，版管理機構を持たせる場合などの用途もある．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つのファイルに行われる．

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 の持つツールを利用する．

- CVS

CVS[3][11][27] 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，近年最も良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ことも考慮し，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) を利用することも可能である為，オープンソースによるソフトウェア開発やグループウェアの場面で活躍の場が多い．その最たる例が，FreeBSD や OpenBSD 等のオペレーティングシステムの開発である．

2.2.2 電子メール

電子メールとは，インターネットやイントラネット等のネットワークを通じて, 文書や画像等のデータをやりとりするためのシステムである．今日では，ネットワーク環境の充実に伴い，容易に利用することが可能となり，単に「メール」と称されることも多い．

オープンソースソフトウェア開発では，開発者が世界中に分散して存在し，開発作業を行うことが多い．そのため，互いの意志疎通のための手段として，インターネットを介した電子メールが一般的に利用される．また，開発者間での電子メールのやりとりを一括して管理するために，メーリングリスト (Mailing List) と呼ばれるシステムが利用されることも多い．メーリングリストでは，例えば，ある参加者がメーリングリスト宛に電子メールを送信すると，同じ電子メールが参加者全員に配信される．また，誰かがその電子メールに対して返信すると，その電子メールも参加者全員に配信される．このため，他の開発者間での議論内容を，各開発者が容易に捕捉することが可能となる．さらに，これらの電子メールが膨大な量になると，アーカイブ (archive) として管理される．また，WWW を用いて，アーカイブの中から電子メールによる議論の内容を検索するシステムも存在する．

これらのシステムを利用することにより，分散している開発者の間でさまざまな情報を共有することが可能となり，開発作業の促進につながる．

2.3 問題点

オープンソースソフトウェア開発では，分散した複数の開発者が自由に各々の開発作業を行う．これらの開発者は，普段から開発作業に専念するのではなく，個人的な時間を利用して作業を行うことが多い．従って，並列的に作業を行う他の開発者の進捗状況を綿密に追跡する時間が十分に確保できず，その把握が難しい．あるいは，開発者間での意志疎通が不足してしまう傾向がある．すなわち，この種の開発においては，開発者間での意志疎通の支援が不十分であることが，重要な問題点の一つである．

以下では，本研究において，この問題点を解決する上で着目すべき要因について説明する．

2.3.1 システム固有の情報蓄積

現状のオープンソースソフトウェア開発は，複数のシステムを組み合わせた開発環境で作業が行われる．これらのシステムは，その目的に応じて，開発に関する情報をシステム内部に蓄積し，開発者に提供する．ところが，各システムから提供された情報は，あくまでもそのシステム固有の情報でしかない．従って，それらの情報が開発者に対して個々に提供された場合には，開発者が必要とする情報が含まれているとは限らない．そのため，開発者の立場から見ると，各システムの情報量が不足していると考えられてしまう．

例えば，多くの開発者は，版管理システムのリポジトリからリビジョン情報を取得し，参照する．ところが，版管理システムのリビジョン情報は，ファイル単位でリビジョンが管理される．このため，ファイル単位での履歴を取得することは容易に可能であるが，その他の視点から履歴を取得することは困難である．例えば，ある開発者がこれまでに行った開発作業の履歴を取得したり，特定の日時に行われた更新作業の履歴を取得することは非常に難しい．

具体的には，以下のような問題が生じる．

CVS リポジトリ内にある，`src/bin/ln/ln.c,v` というファイルについて，ファイル単位でリビジョン情報を取得するのは容易である．例えば，CVS コマンドの一つである `rlog` コマンドを利用することで，図 4 に示すように，ファイル単位でリビジョン情報を簡単に取得可能である．

しかし，リビジョン `1.19` の更新作業者 `sobomax` がこれまでに行った開発作業の履歴を取得する方法や，リビジョン `1.19` の更新日時 `2001/04/26 17:15:57` と同時に行われた更新作業の履歴を取得することは非常に難しい．

2.3.2 システム間の連係不足

オープンソースソフトウェア開発では，開発管理を効率良く行うことを目的として，SourceForge[23] や SourceCast[4]，OSDL(Open Source Development Lab.)[15] 等のサービスが提供されている．これらのサービスでは，電子メールや会議システム等の汎用的な CSCW ツールや，その内容を記録したアーカイブ，WWW，版管理システム等，多くのシステムをまとめて開発者に提供する．しかし，提供される各システムは互いに独立したものであり，単一の環境として何らかの連係を持っているわけではない．すなわち，電子メールで行われた意志疎通の内容はそのシステムの内部にアーカイブ化されて記録されるが，それらの情報は版管理システム CVS の情報と連係を持つことはない．そのため，CVS リポジトリを参照しながら，それに関連した電子メール上での話題を取得したい場合，あるいは，電子メールを参照しながら，それに関連した CVS リポジトリの情報を取得したい場合には，開発者が二つのシステムから個々に情報を取得して，それらを結びつける必要がある．しかし，そのように関連付けを開発者が行う場合のコストは，開発規模が増大するにつれて非常に大きくなる．

具体的には，以下に示すような問題が生じる．

あるオープンソースソフトウェア開発では，版管理システム CVS と，電子メールとそのアーカイブを利用するとする．ある開発者が，CVS リポジトリ内の `src/bin/ln/ln.c,v` というファイルのあるリビジョン `1.19` の情報を参照した際に，それに関連した電子メール上での話題を取得したい場合には，ファイルパスやリビジョン番号，ログメッセージからい

```
% rlog ln.c,v

RCS file: ln.c,v
Working file: ln.c
            .
            .
         (中略)
            .
            .
----------------------------
revision 1.20
date: 2001/04/26 17:22:48;  author: sobomax;  state: Exp;  lines: +1 -1
Previous commit should read:

style(9) Reviewed by:   bde
----------------------------
revision 1.19
date: 2001/04/26 17:15:57;  author: sobomax;  state: Exp;  lines: +28 -9
Bring in ‘-h’ compatability option and its alias ‘-n’ to match NetBSD and GNU
semantics.

style(9) Reviewed by:
Obtained from:  NetBSD
----------------------------
            .
            .
         (以下略)
            .
            .
```

図 4: ファイル単位のリビジョン情報

くつかのキーワードを抽出して，電子メールの全文検索を行うなどの方法を利用する必要がある．あるいは，ある電子メールを参照しながら，それに関連した CVS リポジトリのリビジョン情報を取得したい場合には，電子メールのサブジェクトや本文からいくつかのキーワードを抽出して，それを手がかりにして CVS リポジトリを検索しなければならない．

2.4 関連研究

本節では CVS リポジトリに蓄積された開発履歴を閲覧する為の既存のシステムの紹介と，そのシステムが抱える問題点について指摘する．

2.4.1 開発履歴閲覧システム

- CVSWeb

CVSWeb [6] は CVS リポジトリ内に保存されている開発履歴情報を Web ブラウザ経由で視覚的に閲覧できるシステムである．

このシステムは, プロジェクト内のファイル階層の閲覧や, 各ファイルのチェックインの履歴の確認, 各リビジョンの内容の表示, そしてあるファイルの任意のリビジョン間の差分を色付きで表示する機能なども持っている．

またネットスケープ等の Web ブラウザを通して情報を表示するので, GUI の為の新たなシステム導入が不必要であるという特徴をも持つ．

- ViewCVS

ViewCVS [24] は CVSWeb と同様に CVS リポジトリ内の開発履歴情報を Web ブラウザ経由で閲覧できるシステムである．

このシステムは CVSWeb の機能に加え, 以下の機能を兼ね備えている．

- 正規表現やワイルドカードを用いたリビジョン検索
- 任意のタグの時点におけるプロダクトの tar ファイルの自動生成
- 正規表現を用いたファイルの検索
- CVSGraph [5] を用いたリビジョンツリーの視覚的表示
- Bonsai

Bonsai [2] は CVS リポジトリのコンテンツに対して検索を実行できるツールである. 検索条件の指定方法は多岐に渡り, モジュールやディレクトリを指定しての検索や, 正規表現を用いた, ブランチ・ファイル名・更新作業者を指定しての検索などが行える. また時間を指定しての検索も行うことが出来, その指定の仕方も何時間・何日以内だとか, いつからいつまでのリビジョンを検索などということも出来る.

2.4.2 既存の開発履歴閲覧システムの問題点

以下に既存の開発履歴閲覧システムが抱える問題点を挙げる.

- リビジョン検索を持たないものがある

  CVSWeb などはリビジョン検索機能を持っていないが, その場合ユーザがプロジェクトに精通していない限り, あるトピックに関する変更がどこでどのように行われているかを捜し当てることが困難である.

- ファイル単位での開発履歴の追跡

  既存の履歴閲覧システムでは, ファイル単位での開発履歴の追跡しか行うことが出来ず, 複数のファイルやディレクトリを対象として開発の流れを追うことが困難である.

- 時間によるリビジョン間に繋がりを認識するのが困難

  複数のファイルを同時にチェックインしたときのように, 複数のリビジョンが関連を持つことがあるが, 既存の履歴閲覧システムではそのようなリビジョン間の関連性を把握するのが困難である. またソースコード間の参照関係のように複数のファイル間のつながりを, リビジョン関係も考慮した上で把握することは非常に困難である.

以上の問題点を解決することを目的とした, 開発履歴閲覧システムを設計, 試作した. 詳細は次節で述べる.

3　開発履歴理解支援システム CREBASS

本研究では版管理システム CVS のリポジトリを対象として,Web ブラウザを通しての内容の閲覧, 及び開発者が必要とする情報の取得のための開発支援環境の構築を目指している. 本節では, 試作した開発履歴理解支援システム CREBASS (Cvs REpository Browse And Search System) について説明する.

3.1　システムの概要

本システムでは以下の情報を Web ブラウザを通して閲覧することができる.

- ディレクトリ構造の表示

  各ディレクトリ内に存在するサブディレクトリとファイルを表示する.

- 任意のリビジョン間の差分の表示

  あるファイル内の任意のリビジョン間の差分を表示する. このとき修正内容が分かりやすいように変更された内容に応じて色別に表示を行う.

- プロジェクト内のファイルのログ

  ファイルに対するチェックインの履歴を表示する. 各チェックインの履歴には更新作業者, 更新日時, ログメッセージなどがある.

- 各リビジョンの内容

  各リビジョンにおけるファイルの内容を, そのリビジョンの情報とともに表示する. リビジョンの情報には更新作業者, 更新日時, ログメッセージなどがある. またファイルの内容が C 言語のソースファイルであった場合は, その中で関数を参照している箇所をクリックすることで, 現在閲覧中であるファイルのリビジョンのチェックイン時点での関数の定義元へとジャンプすることもできる.

- リビジョン情報の検索

  各リビジョンのチェックインに関する情報を指定した条件の元で検索することができる. 検索結果には対象リビジョンのファイル名, リビジョン番号, 所属ブランチ名, 更新日時, 更新作業者, ログメッセージ, そしてキーワードが含まれる. また条件の指定は以下に示す 5 つの方法を組み合わせて行う.

  – 更新日時

指定した日時と同時刻にチェック印されたリビジョンの情報を検索する. ただし, 複数ファイルのチェックイン時に処理時間によってチェックイン日時がずれてしまうことを考慮し, 指定した日時から前後 1 分以内にチェックインされたリビジョンを検索結果として表示する.

- 更新作業者

  指定したユーザがチェックインを行なったリビジョンを検索結果として表示する.

- キーワード

  リビジョンのチェックイン作業を象徴するキーワードを指定し, それと同じものを持つリビジョンを検索結果として表示する.

- リビジョンが所属するブランチ名

  ブランチ名を指定し, それと同じ名前のブランチに所属するリビジョンを検索結果として表示する.

- ディレクトリ名

  指定されたディレクトリ以下のファイルのリビジョンを検索結果として表示する

- プロジェクトデータの変遷の折れ線グラフによる表示

  プロジェクト内にあるファイルの行数, 編集量, 累計 commiter 数の変遷を折れ線グラフにして表示する. 表示の際はファイル・ディレクトリ名とブランチ名を指定することができる. ファイル名を指定した時はそのファイルを, ディレクトリ名を指定した時はそのディレクトリ下にある全てのファイルを対象とする. またブランチ名を指定した時はそのブランチに所属するリビジョンだけを対象とする. グラフ生成の際は対象を複数選択することも出来る. この場合, 複数の対象が 1 つのグラフ上に同時に表示される. このグラフを用いることで各対象間の遷移の違いを確認することが出来る.

- cvs history 情報の表示

  CVS には history というコマンドがあり, それを用いることによって過去, リポジトリに対して起きたイベントのヒストリを得ることができる. ここで言うイベントとはファイルのチェックイン, チェックアウト, タグ付けなどのリポジトリに対するアクションのことを指す. また各イベント情報はイベントを起こしたユーザ名, 日時やイベントに固有の情報などを持つ. 本システムではこのイベント情報をデータベース化し, イベントの種類やユーザ名, 期間を指定して検索, 表示する機能を持つ. また,cvs history コマンドでは複数ファイルの同時チェックインといったような複数イベントの同時発生を直接検出することはできないが, 本システムではそれを認識し, ユーザに同時発生したイベントだとわかるように表示する.

3.2 システムの構成

本システムは以下のサブシステムから構成される.

- データベース
  - 関数定義・参照情報データベース
  - リビジョン情報データベース
  - プロジェクトデータベース
  - cvs history 情報データベース
- ユーザインターフェース
  - ディレクトリ構造表示部
  - ファイルログ表示部
  - 差分表示部
  - コード表示部
  - リビジョン情報検索部
  - プロジェクトデータ表示部
  - cvs history 表示部

3.3 データベース

本システムでは以下の 4 つのデータベースを用いることによりユーザに有用な情報の提示を行う.

- 関数定義・参照情報データベース

  あるリビジョンの何行目に何の関数の定義があるのか, また何行目で何の関数を参照しているかをデータベース化したもの.C 言語のソースコードを表示する際のクロスリファレンスの構築の際に用いる.

- リビジョン情報データベース

  各リビジョンのチェックインの情報をデータベース化したもの. リビジョン検索の際に用いる.

- プロジェクトデータベース

  各ファイルの各リビジョンの総行数, 編集量, 累計 commiter 数をデータベース化したもの. これらの情報の折れ線グラフの描画の際に用いる.

- cvs history 情報データベース

  リポジトリに対してこれまでに起きたヒストリイベントをデータベース化したもの. ヒストリイベントの検索, 表示の際に用いる.

3.4 ユーザインターフェース

この部分はユーザと各データベースを結ぶインターフェースの役割を果している. 開発者から要求を受けると CVS リポジトリやデータベースと連結し, 要求されたデータを表示する.

- ディレクトリ構造表示部

  あるディレクトリの中にどのようなファイルやサブディレクトリが存在するかを確認できる. ここでは各ファイルのリビジョン番号, 最終チェックインからの経過時間, 最終更新者, 及びログメッセージが附属情報として閲覧でき, それらをキーとしてファイルの並びをソートすることも出来る. また, リストボックスからタグを選択することで, そのタグを持つファイルのみを表示するということも出来る.

  ディレクトリ名をクリックするとそのディレクトリの構造表示部へ, ファイル名をクリックするとそのファイルのファイルログ表示部へ, リビジョン番号をクリックするとそのリビジョンの内容の表示へとジャンプする.

- ファイルログ表示部

  あるファイルのコミットログの履歴を確認することが出来る. 新しいリビジョンから順番にリビジョン番号, 更新日時, 更新作業者, 所属ブランチ名, タグ名, 前リビジョンからの編集量, ログメッセージが表示される. ブランチ名, もしくはタグ名をクリックすることでそのタグ名を持つリビジョンのみのコミットログの履歴を表示することも出来る. また任意の 2 リビジョンを選んでその間の差分を表示させることも出来る.

  リビジョン番号をクリックするとそのリビジョンの内容の表示へ, 更新日時, 更新作業者をクリックするとそれらの値でリビジョン検索をした結果を表示する.

- 差分表示部

  あるファイルの 2 つのリビジョン間の内容の差分を表示する. 前のリビジョンからどの行が削除され, どの行が加えられ, そしてどの行がどのように変更されたかを判別し

やすいようにそれぞれの場合について別の色で背景を塗り, わかりやすいように表示する.

- コード表示部

  あるファイルのあるリビジョンの内容を表示する.

  画面の上部にはファイルパス, リビジョン番号, 更新日時, 更新作業者, 所属ブランチ名, ログメッセージといったリビジョン情報を表示し, その下部にリビジョンの内容を表示する.

  そのファイルが C 言語のソースファイルであった場合は, ここにリビジョン関係を考慮した関数クロスリファレンス機能が加わる. 関数を参照している箇所にはリンクが張ってあり, そこをクリックするとそのリビジョンのチェックイン時点での関数定義元へとジャンプする.

- リビジョン検索部

  指定した条件に合致するリビジョンを検索する. 指定する条件の種類には更新日時, 更新作業者, キーワードがある. また検索する範囲をディレクトリ名やブランチ名を指定して絞ることも出来る. 検索結果はファイルパス, 更新日時, 更新作業者, リビジョン番号, ログメッセージ, キーワードでソートすることが出来, またそれぞれ逆順でのソートも出来る.

- プロジェクトデータ閲覧部

  ファイル, もしくはディレクトリ下のファイルの総行数, 編集量, 累計 commiter 数の折れ線グラフを生成して表示する. 複数のファイル, ディレクトリのグラフを同時に表示してそれぞれを比較することも出来る. またブランチ名を指定することでグラフ化の対象をそのブランチのみに絞ることも出来る.

- cvs history 表示部

  過去にリポジトリに起きたイベントを検索, 表示する. 検索時にはイベントの種類, イベントを起こしたユーザ名, 期間, そしてディレクトリを指定することが出来, その条件に合致するイベント情報群が時系列で表示される. このとき同時に起きたと考えられる複数のイベント情報群は 1 つにまとめて表示される.

4 CREBASS の実装

本節では, これまで述べた開発履歴理解支援システム CREBASS の実装を行なった. 開発に用いた言語は Java で, コードサイズは全体で約 9100 行となった. 開発環境は以下のとおりである.

- CPU:Penitum4 1.5GHz
- RAM:512MB
- OS:FreeBSD 4.9-STABLE
- データベース:GNU GDBM 1.8.3
- WEB サーバ:Jakarta Tomcat 5.0.16
- JDK 1.4.2

4.1 データベース

必要な情報にアクセスしやすいように, あらかじめ CVS リポジトリを解析してデータベースを作成しておく.

データベース作成部のメインルーチンでは CVS リポジトリの中を巡回し, 各サブデータベース作成部に RCS ファイルを渡してデータベースを作成してもらうという処理を行う.

基本的に関数定義情報, リビジョン情報, プロジェクト情報,cvs history 情報は 1 度のリポジトリの巡回でまとめてデータベース化することが出来るが, 関数参照情報は関数定義情報が全て揃ってからでないと作成することが出来ない. よって前者の 4 つのデータベースを作成した後, 再度リポジトリ内を巡回して関数参照情報データベースを作成することになる.

4.1.1 関数定義・参照情報データベース

受けとった RCS ファイルからリビジョンツリーを作り, その中の全てのリビジョンについて構文解析を行ない, データベースを作成する.

C 言語のソースファイルの構文解析には, クロスリファレンサである GNU GLOBAL[9] のサブツール,gctags を用いる.

関数定義情報データベースの各エントリは以下の情報を持つ.

- ファイルパス

  当該ソースファイルが一意に特定できるように, リポジトリ内のソースファイル名をフルパスで記述する.

- 各リビジョンにおける関数定義情報

  ファイル内の全リビジョンそれぞれにおいての関数定義情報を持つ.

  - リビジョン番号

    ファイル内のリビジョンの特定のために使う.

  - 所属ブランチ名

    関数の参照先を探す際, 出来る限り参照元と同じブランチ上にあるリビジョンの定義を探すようにする. この時にリビジョンがどのブランチに属しているかをすぐに判断できるようにするためのものである.

  - 更新日時

    このリビジョンがチェックインされた日時を表す. 関数の参照元のリビジョンから参照先のリビジョンを探す際にこの日時を基準に対応するものを探す.

  - 各関数の定義情報

    リビジョン内のどの関数がどこで定義されているかを表す.

    * 関数名
    * 定義されている行番号

関数参照情報データベースの各エントリは以下の情報を持つ.

- ファイルパス

  当該ソースファイルが一意に特定できるように, リポジトリ内のソースファイル名をフルパスで記述する.

- リビジョン番号

  リビジョンを特定するのにファイルパスとリビジョン番号の 2 つを使う.

- 所属ブランチ名

  関数の参照先のリビジョンを特定する際, 出来る限り参照元と同じブランチ上で探すために用いられる. この情報と関数定義データベースのエントリ内の所属ブランチ名を照らし合わせて同一ブランチ上のリビジョンかどうかを判定する.

- 更新日時

  関数の参照先のリビジョンを探す時に用いられる. この時刻と同時刻, もしくはそれ以前にチェックインされたリビジョンが, 参照先リビジョンの候補となる.

- リビジョン内での関数参照データ

  リビジョン内で参照している関数の情報を持つ.

  - 関数名

    どの関数を参照しているかを特定するために用いられる.
  - 参照元の行番号

    コードを表示する際, どこにこの関数の参照があるかを把握するために用いられる.
  - 参照先ファイル名

    参照している関数と同じ名前の関数定義がどのファイルで行なわれているかを表す. コードの表示時には, ここで示されたファイル内で, 所属ブランチ名と更新日時から該当する参照先リビジョンを特定することになる. この値の取得には, 関数定義データベース作成時に同時に作っておいた, 関数名とその定義ファイルのハッシュデータベースを用いる.

具体的な処理の流れは以下のとおりである.

- 関数定義情報データベース作成

  1. RCS ファイルが C 言語のソースのものでないならば処理を中断する
  2. RCS ファイルからリビジョンツリーを作成する
  3. リビジョンツリー中の全リビジョンについて以下の処理を行う
     (a) そのリビジョンのコードをチェックアウトする
     (b) チェックアウトしたコードを gctags で構文解析し, どの関数が何行目に定義されているかを取り出す
     (c) 後の関数参照情報計算のために, 関数名をキー, その定義ファイルを値としたハッシュデータベースに, ここで定義されていた関数を登録する
     (d) 取り出した情報をそのリビジョンの関数定義情報としてまとめる
  4. 各リビジョンで作成した関数定義情報をまとめてそのファイルの関数定義情報とし, データベースに格納する

- 関数参照情報データベース作成

  1. RCS ファイルが C 言語のソースのものでないならば処理を中断する
  2. RCS ファイルからリビジョンツリーを作成する

3. リビジョンツリー中の全リビジョンについて以下の処理を行う
   (a) そのリビジョンのコードをチェックアウトする
   (b) チェックアウトしたコードを-r オプション付きの gctags で構文解析し, 何行目でどの関数を参照しているかを取り出す
   (c) 関数定義情報の抽出の際に作っておいたハッシュデータベースを参照して, 参照している関数がどのファイルで定義されているかを取り出す
   (d) 参照している関数名, 参照元の行番号, 定義元のファイル名をまとめて参照情報とする
   (e) リビジョン内の参照情報をまとめてそのリビジョンの関数参照情報とし, データベースに” 参照元ファイルパス:リビジョン番号” をキーとして格納する

4.1.2 リビジョン情報データベース

受けとった RCS ファイルからリビジョンツリーを作り, その中の全てのリビジョンについてリビジョン情報を抽出し, データベースを作成する.

リビジョン情報データベースの各エントリは以下の情報を持つ.

- ファイルパス

  ファイルを一意に特定することが出来るように, リポジトリ内のソースファイルをフルパスで記述したものを登録する.

- リビジョン番号

  ファイルのどの段階の生成物かを特定するために必要な情報. リビジョン情報の特定するには, ファイルパスとリビジョン番号の 2 つを用いる.

- 更新日時

  利用者が各リビジョン間の前後関係を直観的に調べたり, 同時刻に変更されたファイルを検索するために用いる情報.

- 更新作業者

  同一更新者に更新されたファイルを検索するために用いる情報. 更新日時と組み合わせて用いることでファイルの関連性が増すと考えられる.

- 所属ブランチ名

このリビジョンがどのブランチに所属しているかを特定するための情報. ブランチによって開発の志向が異なると考えられるので, 開発情報の検索の際の有用な指針になると思われる.

- ログメッセージ

  ログメッセージはチェックインの際に付けられるコメントであり, そのチェックインの目的を端的に表すものである.

- キーワード

  リビジョン情報に含まれているログメッセージを解析することで, 出現頻度が高い単語, ファイルパスをキーワードとして取得する.

- 内部識別番号

  データベース作成時に各レコード毎に個別の識別番号をふり当てる. 検索効率の向上のために用いる.

このとき, リビジョン情報のデータベースだけではなく, 更新日時, 更新作業者, キーワードをキーとし, それに対応するリビジョン情報の ID を値とするハッシュデータベースも作成する. これによってリビジョン情報の検索時にデータベース全体を巡回する必要が無くなり, 検索効率が向上する.

次に各リビジョンのキーワードの抽出について説明する. キーワードはチェックイン時のログメッセージから抽出し, 基本的に複数回登場する単語のうち上位 3 個程度をそのリビジョンのキーワードとする. ただし, アルファベットの大文字のみからなる単語や, ファイルパスなどは重要である可能性が高いので, これらの単語には重みを付け, キーワードになりやすいようにしている.

4.1.3 プロジェクトデータベース

受けとった RCS ファイルからリビジョンツリーを作り, その中の全てのリビジョンについて所属ブランチ, 更新日時, 更新作業者, 行数, そして編集量のデータを取り出してそのリビジョンのプロジェクトデータとしてまとめ, 最後に全リビジョンのデータをそのファイルのプロジェクトデータとしてまとめてデータベースに格納する.

プロジェクトデータベースの各エントリは以下の情報を持つ.

- ファイルパス

  ファイルを一意に特定することが出来るように, リポジトリ内のファイルをフルパスで記述したものを登録する.

- 各リビジョンのデータ

  ファイル内の各リビジョンの行数, 編集量などのデータを持つ.

  - リビジョン番号

    ファイル内のリビジョンの特定に用いられる.

  - 所属ブランチ

    このリビジョンがどのブランチに属しているかの特定に用いられる. グラフ作成時にブランチ名を指定した時は, その値をこのエントリの値を比較し, 一致しないリビジョンはグラフ作成には用いない.

  - 更新日時

    このリビジョンの更新日時を表す. グラフ作成時にはこの値は横軸の値として用いられる.

  - 更新作業者

    このリビジョンの更新作業者を表す. グラフ作成時にはこの値から累計 commiter 数を計算し, それを縦軸の値として用いる.

  - 行数

    このリビジョンの行数を表す. グラフ作成時にはこの値は縦軸の値として用いられる.

  - 編集量

    このリビジョンが前のリビジョンから何行足され, また何行引かれたかを表す. グラフ作成時にはこの値は縦軸の値として用いられる.

4.1.4 cvs history 情報データベース

cvs history コマンドの出力を解析してヒストリイベント情報を生成し, データベース化する. cvs history データベースの各エントリは以下の情報を持つ.

- 日時
- ユーザ名
- 同時に起こったイベント群このイベント群の中のここのエントリは以下の情報を持つ.

  - イベントの種類
  - イベントの種類に特有のデータ

イベントの種類	イベントに特有のデータ
タグ付け	タグ名, タグを付けるモジュール, タグを付けるリビジョン番号, タグ
チェックアウト	チェックアウトするモジュール, タグ
エクスポート	エクスポートするモジュール, タグ
リリース	リリースするディレクトリ
古くなったファイルを作業用コピーから削除	ファイル名, リビジョン番号
ファイルをチェックアウトし, ユーザファイルを上書き	
競合発生	
マージ	
チェックイン	
ファイル追加	
ファイル削除	

表 1: イベントの種類とそれに特有のデータ

イベントの種類に特有なデータとは, 例えばタグ付けイベントならばタグを付けたファイル・モジュール, 及びタグ名, またファイルのチェックインイベントならばチェックインしたファイルパス, リビジョン番号などである. 表 1 にイベントの種類とそれに特有なデータをまとめたので参照されたい.

データベース構築の際, 同時に起こったと考えられる複数のイベントを一まとまりのイベントとしてまとめてデータベース化する必要がある. ここで同時に起こったと考えられるイベントの条件として以下の 2 つを挙げる.

- 同一ユーザによって起こされたイベントである
- イベントの起こった時刻が前後 1 分以内の範囲に収まっている

同時に起こったイベントとは, あるユーザが複数のファイルをコミットしただとか複数のファイルに同時にタグを付けたなどという状況を想定しているのでたとえ同時刻であろうとも別のユーザによって起こされたイベントは同時に起こったイベントであるとは考えないことにする.

また時間的に前後 1 分の猶予を持たせているのは,CVS が複数のファイルを処理する時は

1 つずつ別々に処理をする, という特性のため, 同時に行なったイベントでも多少の時間のずれが生じる可能性があるためである.

次に実際にどのようにして cvs history データを取得するかを説明する. cvs history の出力は一連の行である. それぞれの行が 1 つのヒストリイベントを表している. そしてこの行はまずユーザ毎にソートされた後に日時についてソートされた並びになっている. そこで cvs history データベースの流れは以下のようになる.

1. cvs history の出力を 1 行読み込む
2. 行を解析してヒストリイベントデータを生成する
3. 一つ前に生成したヒストリイベントのユーザ名と日時を確認する
   - 一つ前とユーザ名が同じで日時の差が 1 分以内ならば
     - 一つ前のイベントと同時に起こったイベントと判断し, まとめる
   - そうでなければ
     - まとめていたデータを一つの同時イベントとして確定させる
4. 全ての出力を処理したら生成した同時イベントを時系列でソートしてデータベースに格納する

4.2 ユーザインターフェース

ユーザインターフェース部はユーザからの要求を受けて, リポジトリやデータベースから情報を取り出し, 表示する. 以下ではそれぞれの部分がどのようにして情報を検索し, 表示しているかについて説明する.

4.2.1 ディレクトリ構造表示部

指定されたディレクトリのパスを受けとり, そのディレクトリ中のサブディレクトリ, ファイルを表示する. ファイルについては最終チェックインからの経過時間, 最終更新者, リビジョン番号, ログメッセージが表示される. またパスだけでなく, タグも受けとった場合はそのタグを持つリビジョンについての情報を表示する. ディレクトリ構造表示画面を図 5 に示す.

処理の流れは以下のようになる.

1. CVS リポジトリ中の指定されたディレクトリに含まれるファイルのリストを取得する
2. リスト中の全ファイルについて以下の処理を行う

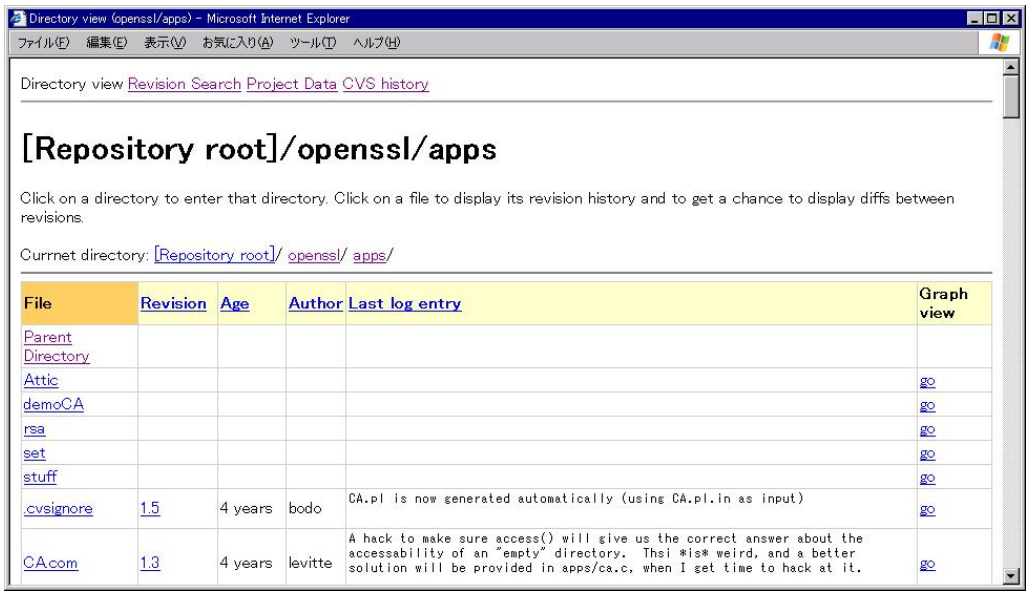

図 5: ディレクトリ構造表示画面

(a) ディレクトリならばそのまま出力する

(b) ディレクトリでなく, かつ RCS ファイルでもないファイルならば何もしない

(c) RCS ファイルならば, そのファイルからリビジョンツリーを作成する

(d) タグが指定されているならそのリビジョン, そうでなければ最新リビジョンを取り出す

(e) そのリビジョンからリビジョン番号, 経過時間, 更新者, ログメッセージを取り出し出力する

4.2.2 ファイルログ表示部

指定されたファイルのログメッセージの履歴を他の更新情報とともに表示する. また, タグを指定した場合はそのタグを持つリビジョンのログのみを表示する. ファイルログ表示画面を図 6 に示す.

処理の流れとしては, 指定されたファイルに対応する RCS ファイルをリポジトリから取り出し, それからリビジョンツリーを作成, タグが指定されているならばツリーから必要なリビジョンを取り出し, 更新日時が新しいものから順に必要な情報を取り出し, 表示していく, というようになる.

ただし各情報には必要に応じて適切なリンクを張る必要がある. 例えばリビジョン番号ならばそのリビジョンのコード表示部へ, 更新者, 更新日時ならばその値で検索したリビジョン

図 6: ファイルログ表示画面

検索の結果へ, タグならばそのタグを指定したファイルログ表示部へ, そして差分表示部へのリンクを張る. このとき必要な画面へ必要な情報を含めたリンクを計算して張らなければならない.

4.2.3 差分表示部

指定されたファイル, 及び 2 つのリビジョンから差分を計算し, どこが追加, 削除, 変更された行なのかわかりやすいよう色つきで表示する. 差分表示画面を図 7 に示す.

まず指定された 2 つのリビジョンの内容をチェックアウトし,diff コマンドを用いて unified 形式の差分を取得する.unified 形式の diff の出力では行頭に'+' か'-' が付くことがあり,'+' が付いている行は前のリビジョンからそこに行が足されたことを表しており,'-' が付いている行は前のリビジョンのその行が削除されたことを意味している. また行の内容が変更されたことは行の削除と追加を組み合わせることで表現している.

そこで unified 形式の diff の出力の行頭の文字を確認しながら削除された行, 追加された行, 変更された行を判別し, 色分けして表示を行なっている.

4.2.4 コード表示部

指定されたファイル, リビジョンの内容を, そのリビジョンのチェックイン情報とともに表示する. さらにそのファイルが C 言語のソースコードであった場合は, 関数の参照元のクリッ

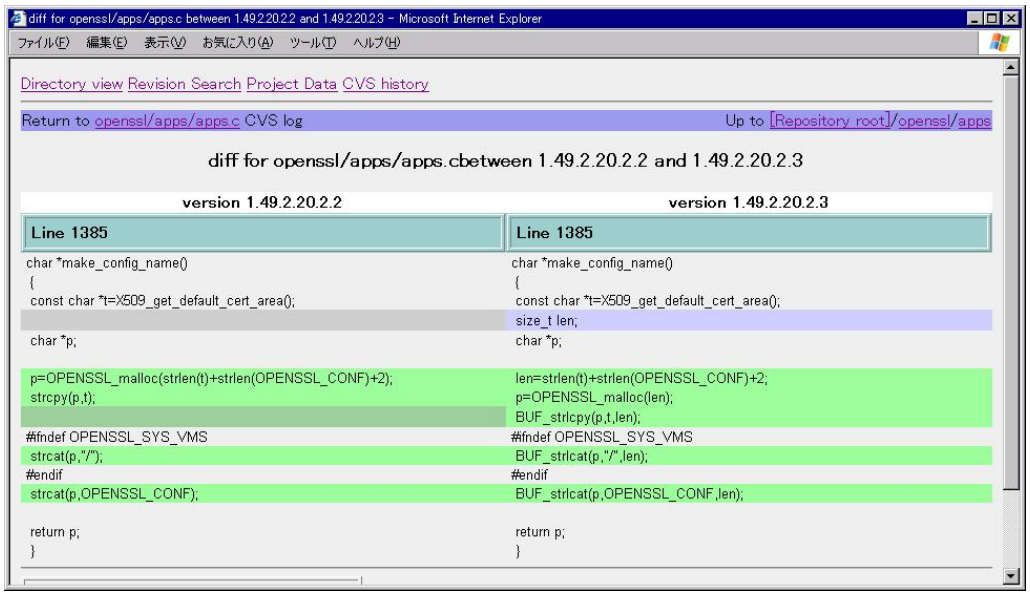

図 7: 差分表示画面

ク時にその時点での関数の定義元へとジャンプする為のリンクも張る. コード表示画面を図 8 と図 9 に示す.

リビジョンの内容の取得はリポジトリからのチェックアウト, またチェックインの情報は RCS ファイルから読み込んで表示するだけなので説明は割愛する.

関数の参照先へのジャンプについてはデータベースを参照して適切なジャンプ先を取得し, そこへのリンクを張るという形で実現する.

参照先の導出は以下の手順で行う.

1. 関数参照情報のエントリを参照し, その関数の定義ファイルがいくつあるかを確認する
2. 定義ファイルが 1 つならば
   (a) そのファイルの関数定義情報をデータベースから取り出す
   (b) 参照元のリビジョンと同じブランチ上で, かつその更新日時以前にチェックインされたリビジョンのなかで最新のリビジョンを捜し出す
      - 参照元と同じブランチ上で見つからなかった場合はメインブランチ上で再度該当するリビジョンを捜し出す
      - それでも該当するリビジョンが見つからない場合は参照先がないものとする
   (c) 該当するリビジョンが見つかった場合は, その中で参照している関数が本当に定義されているか確認する

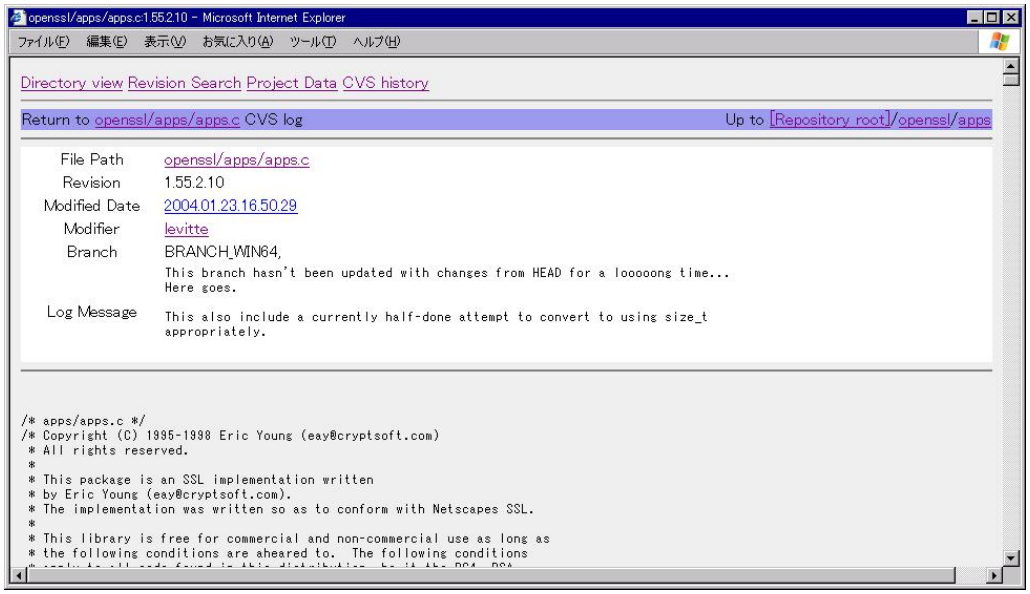

図 8: コード表示画面

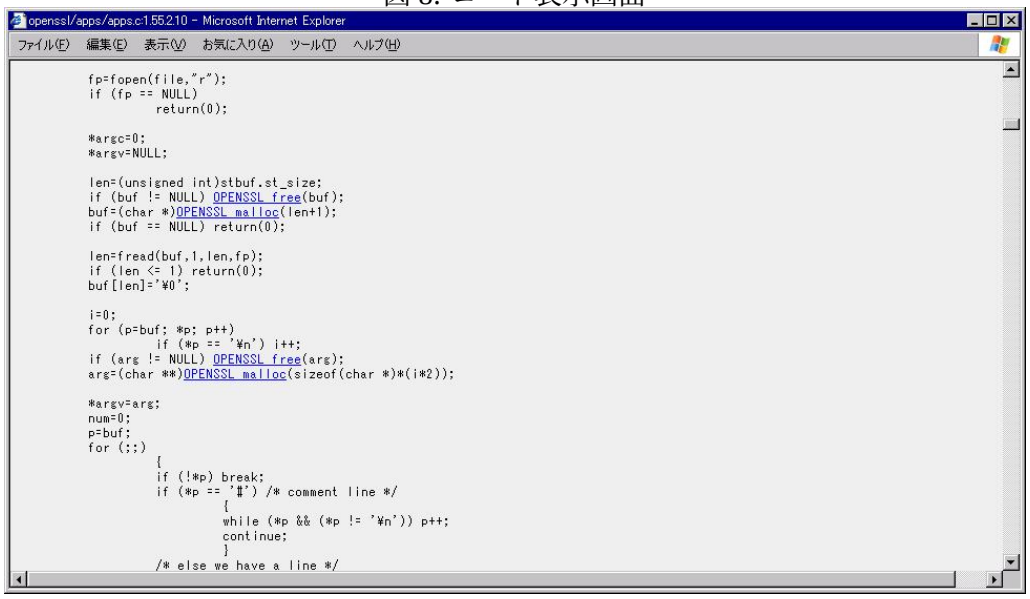

図 9: 参照先へのリンクがついた関数

- 該当する関数定義が見つからなかった場合は参照先がないものとする

(d) 該当する関数定義が見つかった場合はそこを参照先とし, そのファイル, リビジョン, 行番号へのリンクを張る

3. 定義ファイルが 2 つ以上あるならば

   (a) どちらのファイルの定義を見るか選ばせる為の画面へのリンクを張る

4.2.5 リビジョン検索部

ユーザから検索条件を受けとり, その条件に合致するリビジョン情報を検索結果として表示する. リビジョン検索画面とその結果を図 10 と図 11 に示す.

検索の流れは以下のようになる.

1. 更新日時データベースを参照し, 指定された日時, 及びその前後 1 分をキーとするデータを取り出す. このデータにはその日時に更新されたリビジョン情報の ID が格納されている.
2. 更新作業者データベースを参照し, 指定された更新作業者をキーとする, リビジョン情報の ID を取り出す
3. キーワードデータベースを参照し, 指定されたキーワードをキーとする, リビジョン情報の ID を取り出す
4. 得られた ID をマージし, 全ての条件に合致する ID を導く
5. 検索結果の ID を用いてリビジョン情報データベースからリビジョン情報を取り出す
6. 検索結果を, 指定されたブランチ名, ディレクトリ名でフィルタリングする
7. 最終的に残った結果を, 指定された条件でソートし, 表示する

4.2.6 プロジェクトデータ閲覧部

指定されたファイル, もしくは指定されたディレクトリ以下の全ファイルの総行数, 編集量, 累計 commiter 数の変遷を折れ線グラフとして表示する. またブランチ名を指定した時は, そのブランチ上のリビジョンのみで折れ線グラフを構築する. 各データの折れ線グラフを表示した画面を図 12, 図 13, そして図 14 に示す.

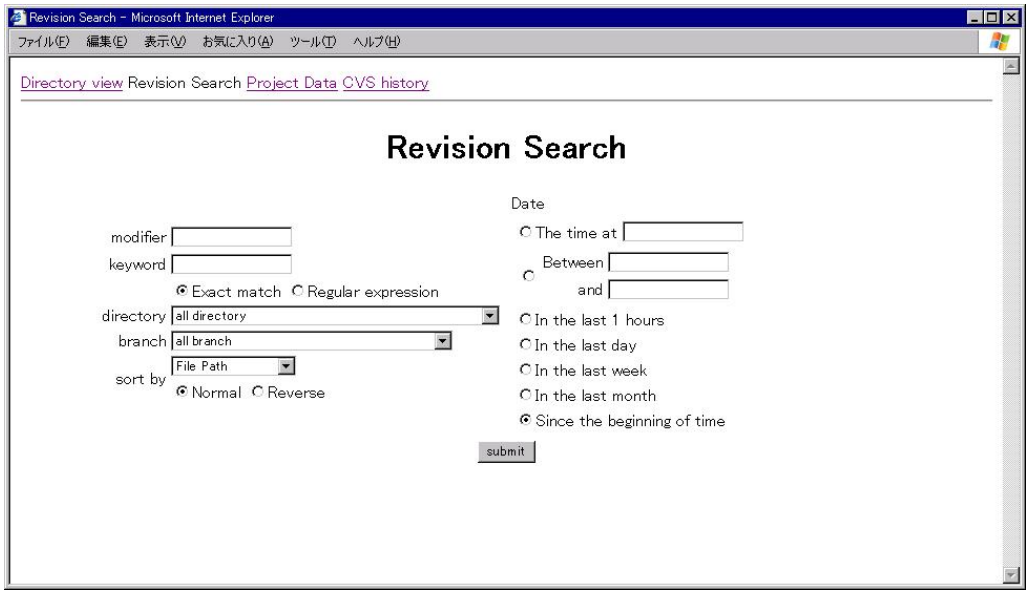

図 10: リビジョン検索画面

図 11: リビジョン検索の結果

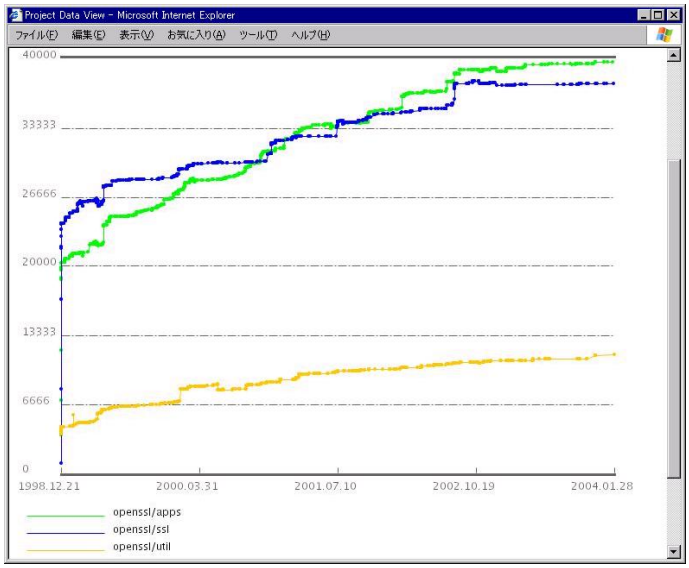

図 12: 総行数の折れ線グラフ

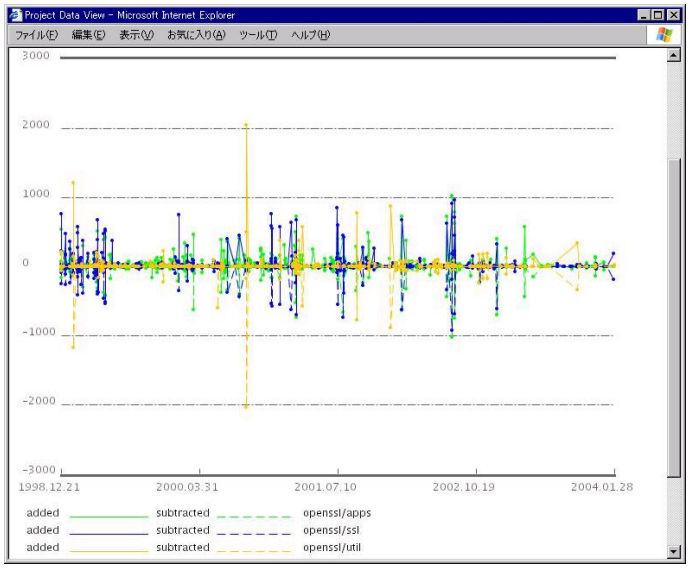

図 13: 編集量の折れ線グラフ

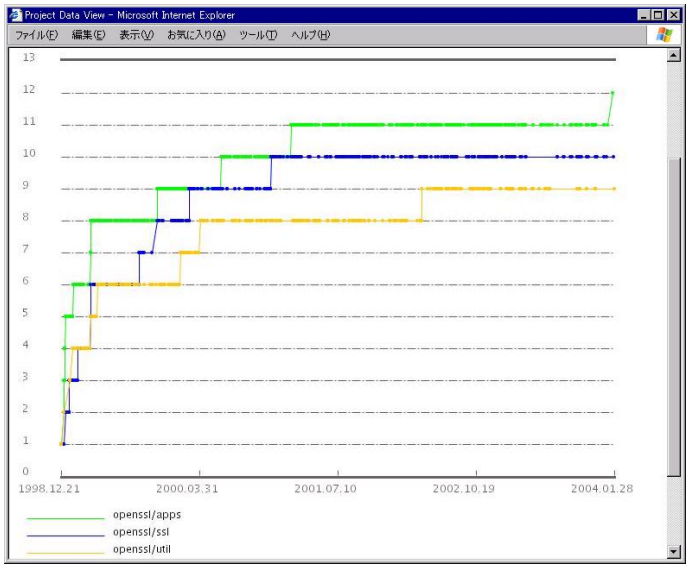

図 14: 累計 commiter 数の折れ線グラフ

プロジェクトデータベースには既に各ファイル毎の行数などの変遷が格納されている. よって 1 つのファイルのグラフを書くときはデータベースから取り出したデータからそのままグラフを書けば良い.

ディレクトリが指定された場合はそのディレクトリ下の全てのファイルのデータをデータベースから取り出し, それらをマージしたデータを用いてグラフを作成する.

4.2.7 cvs history 表示部

ユーザからヒストリイベントの種類, ユーザ名, 期間, そしてディレクトリといった条件を受け取り, それらの条件に一致するヒストリイベントを表示する.cvs history 表示画面を図 15 に示す.

ヒストリデータベースからは時系列にそってデータを取り出すことが出来るので, 処理の流れとしては順にデータを取り出し, 条件に一致するデータならば表示する, といった流れになる.

ただし同時に起こったイベントの中に指定された種類のイベントとそうでないイベントの両方が入っていることもある. そのときは指定されたイベントのみを抜き出して表示する必要がある.

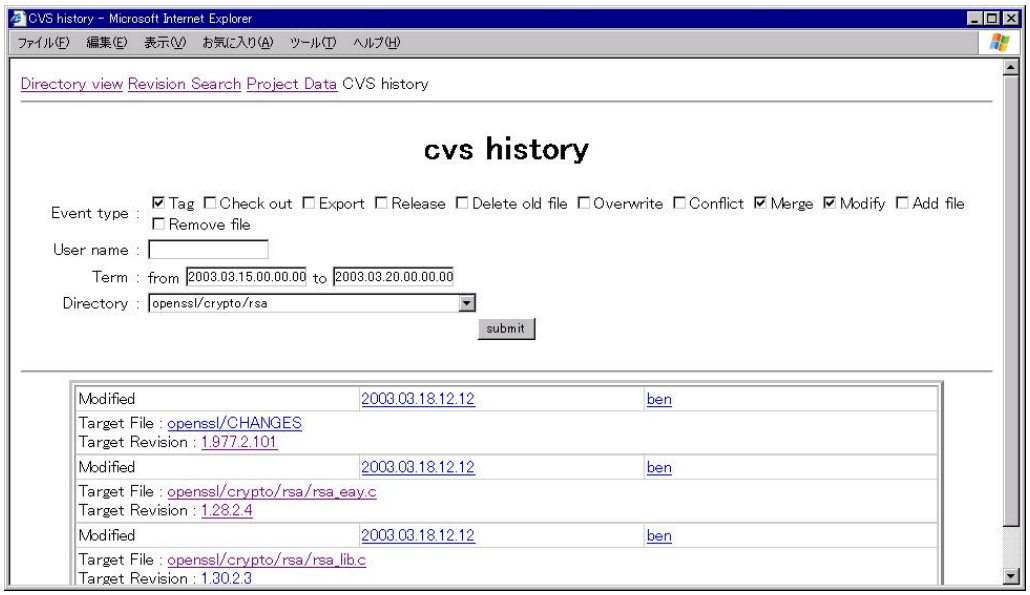

図 15: cvs history 表示画面

5 評価

本節では実際にシステムを動作し, 得られた結果について考察する.

5.1 実験対象

本システムの適用実験を行う為に, 実際のソフトウェア開発の中から OpenSSL[16] の開発をモデルとして仮想的な開発環境を用意する. つまり OpenSSL の CVS リポジトリをミラーリングして, そこから必要な情報を取得し, データベース化する.

なお, 実験環境は以下のとおりである.

- CPU:Pentium4 1.5GHz
- RAM:512MB
- OS:FreeBSD 4.9-STABLE
- データベース:GNU GDBM 1.8.3
- WEB サーバ:Jakarta Tomcat 5.0.16
- JDK 1.4.2

5.2 実験の内容

本システムの評価実験として以下の実験を行なった.

2003 年 3 月 17 日に OpenSSL のリリースが行なわれた. このリリースは, ホストの秘密鍵が流出する可能性があるという,RSA blinding に関する脆弱性を修正する為のものである. 今, 利用者はこのリリースに関するプロジェクトへの変更点の詳細を確認したいものとする. そこで本システムを用いて変更箇所の検索, 及び変更内容の確認を行うというものである.

まず利用者はこのリリースに関する変更箇所を確認する為に, リビジョン検索機能を用いて,2003 年 3 月 15 日から同 20 日までにチェックインされ, かつキーワード RSA を持つリビジョンを検索した.

検索を行なった結果,7 つのリビジョンが条件に一致するものとして表示された. 表示された結果を見ると, そのうち 3 件が ben というユーザによって同時にチェックインされたリビジョン, 残り 4 件が bodo というユーザによって同時にチェックインされたリビジョンであることがわかった.

チェックインの内容を確認する為にログメッセージを確認すると, ben のチェックインには”Turn on RSA blinding by default.” と書かれていた. また ben がチェックインを行なった

openssl/CHANGES ファイルを読むと, タイミング攻撃を回避する為に,RSA blinding をデフォルトでオンにするという記述が確認できた. 以上のことから ben のチェックインは RSA blinding に関する脆弱性を修正する為のものであることが確認できた.

次に RSA blinding をデフォルトでオンにする為の具体的な修正内容を確認する為に, チェックインされた他の 2 つのファイル, openssl/crypto/rsa/rsa_eay.c と, openssl/crypto/rsa/rsa_lib.c の前リビジョンからの差分を確認することにした. すると openssl/crypto/rsa/rsa_eay.c において, 新たに rsa_eay_blinding という関数と, BLINDING_HELPER というマクロが定義されており, それらを用いて blinding のデフォルトでの機能使用を行なっていることが確認できた.

しかし rsa_eay_blinding 関数内で使用している関数 CRYPTO_w_lock, CRYPTO_w_unlock の実装についてはこのファイルからはわからず, また同時にコミットされたファイルからもその記述は発見できなかった. そこでチェックインされたソースコードをコード表示部で表示したところ, 両関数の定義へのリンクが張ってあったので, それをクリックすることでその関数の定義を確認できた.

本システムを用いて実例に基づいた適用実験を行なった結果, リビジョン検索を行うことで必要な更新履歴を特定することが可能なことがわかった. しかし, その情報だけでは更新部分の実装方法について完全に理解することが出来なかった. そこでリビジョン関係を考慮した関数クロスリファレンスシステムを用いることで, そのリビジョンが参照している関数の実装の詳細を取得することが出来た.

5.3 考察

今回の適用実験から, 利用者は日時やキーワードを用いてリビジョン検索を行うことで自分の知りたい開発履歴情報を検索できることがわかった. またリビジョン関係を考慮した関数クロスリファレンスシステムを用いることで, あるリビジョンのソースコード内で参照している関数の定義を調べることが出来, ソフトウェアの実装の理解を深めることが出来ることがわかる.

ここで既存のシステムを用いて今回と実験を行うことを考える. CVSWeb ではリビジョンの検索が出来ない為, プロジェクトの詳細を理解している人でない限り, 該当する変更箇所がどこであるかを瞬時に捜し出すことは困難である. 一方,ViewCVS や Bonsai[2] などはリビジョン検索機能を持っているが, ログメッセージに関する検索を行うことが出来ないので, プロジェクトに精通していないユーザは知りたいトピックに関するリビジョンを絞り込むのに困難を要すると思われる. また, 該当するリビジョンを検索できたとしても, そのリビジョンがその時点で参照している関数の定義を取得する方法が無い為, 対応する関数定義を得る為には時間を指定してプロジェクトをチェックアウトした上で,grep などを用いて検索する必

要がある. 以上のように, 既存のシステムではプロジェクトに精通していないユーザは知りたい情報を検索することが出来なかったり, 多大な手順が必要であったりすると考えられる.

しかし一方では解決に至っていない部分も存在する. 例えばリビジョン検索のときに用いるキーワードの抽出は必ずしも完全とはいえず, ログメッセージの内容を正しく抽象化した単語を抽出できないこともある. また, ログメッセージ自体がコミットの内容を正しく記述できていないこともあり, キーワードだけで目的のリビジョンを正しく検索できない恐れがある. また, 関数クロスリファレンサにも問題点がある. 本システムでは関数の定義, 参照関係の導出を, 関数名の一致のみから判断している. この結果, 複数のファイルで同名の関数が定義されていたときに, 関数の参照先を一意に特定することが出来なくってしまう. 現在はこういった状況にあったときはユーザにどの定義を閲覧するか選択させるようにしているが, この選択で必ずしも正しい定義元を選択できるとは限らず, 間違った関数定義情報を取得してしまう恐れがある.

キーワードの問題に関しては今後データ取得の方法を改善したり, キーワード取得の対象をログメッセージだけでなく, ソースコードのコメントなどからも取得するなどして改善できると考える. また, 関数の定義元の特定手法も関数名の一致だけではなく,static 関数であるかや, 関数のシグネチャなどの属性も用いて特定することで, かなり改良できるのではないかと考えられる.

6　まとめ

本研究では CVS リポジトリに蓄積されたリビジョン情報を解析して, 開発履歴情報の閲覧・検索を行ない, ユーザの開発履歴理解を支援するシステムを作成した. 本システムは各ファイルの開発履歴や任意のリビジョン間の差分表示の他に, リビジョン関係を考慮した関数クロスリファレンス, 更新日時・更新作業者・キーワードによるリビジョン検索, 任意のファイル, もしくはディレクトリ下のファイル群の開発情報の遷移を折れ線グラフで視覚的に確認する機能, 過去にリポジトリに対して起こったヒストリイベントを検索・表示する機能などがある. そして実際のソフトウェア開発で蓄積された開発履歴情報を用いて適用実験を行ない, ユーザが必要とする開発履歴情報の閲覧・検索が行なえることを確認した.

今後の課題としては, リビジョン検索時に用いるキーワードの抽出手法の改善, 関数クロスリファレンサによる関数の参照先特定手法の改善, より大規模なリポジトリによる適用実験, そして実行速度, 及びスケーラビリティの向上が挙げられる.

謝辞

本論文を作成するにあたり，常に適切な御指導および御助言を賜わ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致します．

本論文の作成にあたり，逐次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 楠本 真二 助教授に心から深く感謝致します．

本論文の作成にあたり，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 助手に心から深く感謝致します．

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致します．

参考文献

[1] The Apache Software Foundation, Apache Projects, http://www.apache.org/.

[2] Bonsai, http://www.mozilla.org/bonsai.html.

[3] Brian Berliner, “CVS II:Parallelizing Software Development”, In USENIX Association, editor, Proceedings of the Winter 1990 USENIX Conference, pages 341–352, Berkeley, CA, USA, 1990.

[4] Collab. Net, Inc., SourceCast, http://www.collab.net/products/sourcecast/.

[5] CvsGraph, http://www.akhphd.au.dk/˜bertho/cvsgraph/.

[6] CVSWeb, http://www.freebsd.org/projects/cvsweb.html/.

[7] Eric S. Raymond, “The Cathedral & the Bazaar”, O’REILLY, 1999.

[8] Free Software Foundation, Inc., The GNU Project, http://www.gnu.org/.

[9] GNU GLOBAL, http://www.gnu.org/software/global/.

[10] Jacky Estublier, “Software Configuration Management: A Roadmap”. The Future of Software Engineering in 22nd ICSE, pp.281–289, 2000.

[11] Karl Fogel, “Open Source Development with CVS”, The Coriolis Group, 2000.

[12] Linux Online Inc., The Linux Home Page, http://www.linux.org/.

[13] Merant, Inc., PVCS Home Page, http://www.merant.com/pvcs/.

[14] Microsoft Corporation, Microsoft Visual SourceSafe, http://msdn.microsoft.com/ssafe/.

[15] Open Source Development Lab, Inc., Open Source Development Lab, http://www.osdlab.org/.

[16] OpenSSL, http://www.openssl.org/.

[17] Peter Fröhlich and Woflgang Nejdl, “WebRC Configuration Management for a Cooperation Tool”, SCM-7, LNCS 1235, pp.175–185, 1997.

[18] Peter H. Feiler, “Configuration Management Models in Commercial Environments”, CMU/SEI-91-TR-7 ESD-9-TR-7, March, 1991.

[19] Rational Software Corporation, Software configuration management and effective team development with Rational ClearCase, http://www.rational.com/products/clearcase/.

[20] Reidar Conradi and Berbard Westfechtel, “Version models for software configuration management”, ACM Computing Surveys, Vol. 30, No.2, pp.232–280, June 1998.

[21] The FreeBSD Project, The FreeBSD Project, http://www.freebsd.org/.

[22] Ulf Asklund, Lars Bendix, Henrik B Christensen, and Boris Magnusson, “The Unified Extensional Versioning Model”, 9th International Symposium, SCM-9, LNCS1675, pp.100–122, 1999.

[23] VA Linux Systems, Inc., SourceForge, http://sourceforge.net/.

[24] ViewCVS, http://viewcvs.sourceforge.net/.

[25] Walter F. Tichy, “RCS - A System for Version Control”, SOFTWARE - PRACTICE AND EXPERIENCE, VOL.15(7), pp.637–654, 1985.

[26] 落水浩一郎, “分散共同ソフトウェア開発に対するソフトウェアプロセスモデルに関する基礎考察”, 電子情報通信学会技術研究報告, SS2000-48(2001-01), pp.49–56, 2001.

[27] 鯉江英隆, 西本卓也, 馬場肇, “バージョン管理システム (CVS) の導入と活用”, SOFT BANK, December, 2000