特別研究報告

題目

オープンソース開発支援のための
リビジョン情報と電子メールの検索システム

指導教官
井上 克郎 教授

報告者
佐々木 啓

平成 15 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

オープンソース開発支援のための
リビジョン情報と電子メールの検索システム

佐々木 啓

内容梗概

オープンソース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よく管理するためのリビジョン管理システムや開発者相互の連絡を行うための電子メールを用いたメーリングリストシステムが用いられる．これらのシステムでは，開発者が行ったプロダクト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に保存されており，その履歴情報には将来の開発に活用することのできる情報が多く含まれている．ソフトウェア開発の際に，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，開発の手助けとなることが期待されている．しかし，開発が進むにつれてそれらの情報は膨大なものになるため，開発者が必要とする情報を的確に取得する事は容易ではない．

そこで，本研究では，開発蓄積情報を取得するためのソフトウェア開発支援環境の構築を行う．具体的には，先に私の所属する研究グループにおいて作成された，ソースコード修正支援システム (CoDS) とオープンソース開発支援のためのソースコード及びメールの履歴対応表示システム (SPxR) を統合する事により，開発者が必要とする情報の抽出とそれに関連するデータの表示を行うシステムの試作を行った．さらに，実際のオープンソース開発で用いられた版管理システムの開発履歴情報とメーリングリストアーカイブを用いて，本システムの評価を行った．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．本システムを用いたオープンソース開発支援環境を構築することにより，効率よく開発作業を行うことが可能となり，開発者の負担を軽減し，より質の高いソフトウェアを開発するための基礎とすることができると考えられる．

主な用語

オープンソースソフトウェア開発 (Open Source Software Development)
リビジョン管理 (Revision Control)
ソースコード検索 (Source Code Searching)

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ともない，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ため，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，開発者相互の意志疎通や進捗状況の報告などが行われる．これらのシステムでは開発者が行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できる情報が多く蓄積されている．ソフトウェア再利用の際に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 [9]．このように，オープンソースソフトウェア開発を行う上で，過去に開発されたコードを参照することや再利用することは効率的な手法であると言える．

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やリポジトリに分かれて蓄積されている可能性もある．このため，開発者は参照すべきリポジトリや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ことができるソフトウェア開発支援環境の構築を行う．具体的には，先に私の所属する研究グループにおいて作成されたソースコード修正支援システム (CoDS)[29] とオープンソース開発支援のためのソースコード及びメールの履歴対応表示システム (SPxR)[15] を統合することにより，開発者が必要とする情報の抽出とそれに関連するデータの表示を行うシステムの試作を行った．本システムは特定の情報を用いることによって，該当するファイルや電子メール情報とそれに関連する情報を検索することができるシステムであり，それらの情報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．

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ビジョン管理システムとメーリングリストアーカイブを用いて，本システムの評価を行った．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．

以上のことから本システムを用いたオープンソースソフトウェア開発支援環境を構築する

ことにより，効率よく開発作業を行うことが可能となり，開発者の負担を軽減し，より質の高いソフトウェアを開発するための基礎とすることができると考えられる,.

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について説明し，その問題点について述べる．3 節では，ソフトウェア開発の支援環境について説明し，その問題点似ついて述べる．4 節では，システムの設計と実装について述べ，5 節では，その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とその開発環境

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 [25][27]，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[31] や Linux[18]，GNU[11]，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　オープンソースソフトウェア環境

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 [8] の一つである CVS(Concurrent Versions System)[3][10][17][23] 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．

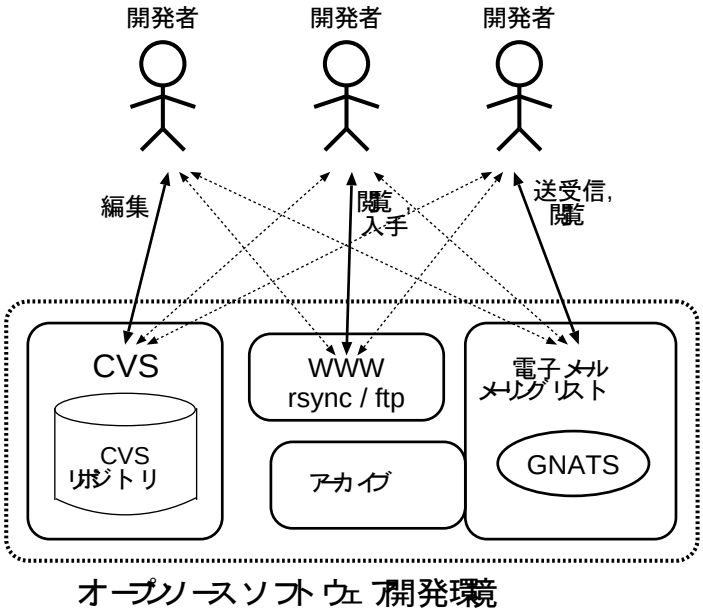

図 1: オープンソースソフトウェア環境の構成例

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 [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．

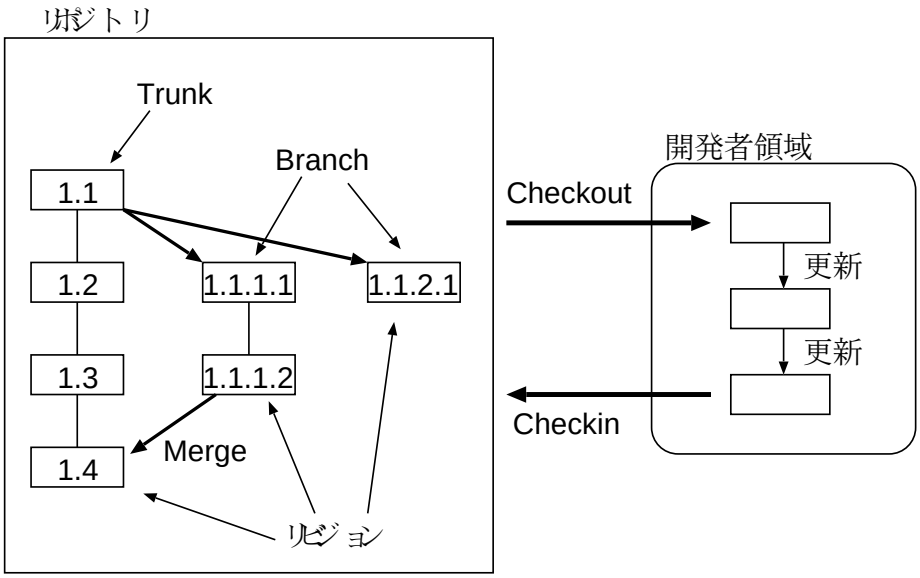

図 2: Checkout/Checkin Model

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[32] や CVS[3][10] といったシステムが標準で利用可能となっている．ClearCase[26] 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[21]

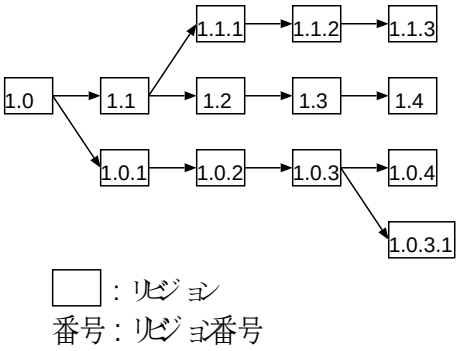

図 3: リビジョンツリーの例

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 [12] も存在する．

近年ハードウェアの進化と共に GUI 化が進んだ為，リポジトリ内部情報の視覚的な把握が可能となった．具体例として，以下のものが挙げられる．

- ネットスケープ等，ウェブブラウザを通して閲覧する．GUI の為の新たなシステム導入が不必要である．
- 版管理システムそのものが GUI 化されている．コマンドラインからの操作に比べ，間違いが少ない．

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[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][10][17] 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，同じ電子メールを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 オープンソースソフトウェア開発の問題点

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[33] や SourceCast[5]，OSDL(Open Source Development Lab.)[24] 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

```
% 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

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．

3 関連研究

3.1 版管理システムを用いた開発支援環境

Moraine と MAME

我々の研究グループにおいて，自動的にすべてのファイルの変更履歴を保存する版管理ファイルシステム Moraine を開発している．また，Moraine を利用した，容易にメトリクスデータを収集可能なメトリクス環境 MAME(Moraine As a Metrics) を構築している [35]. Moraine の利用者は，リポジトリへの check-in や check-out といった版管理における基本的な作業を意識せず開発を進めることができる．また，既存のファイルシステムを用いた堆積型 (stackable) ファイルシステムとして実装しており，既存の環境に容易に導入可能である．

MAME は，Moraine で収集したファイルの変更履歴を用い，さまざまなメトリクスデータを提供する．さらに，任意のファイルの詳細な変更履歴を Web ブラウザを用いて参照できるツールも提供する．Moraine 上に開発環境を構築し，開発環境中のファイルに対する全ての操作は Moraine によって記録される．開発者はデータの表現や分析ツールなどで，記録された履歴を Moraine から取得することが出来る．開発環境自身は何も変更する必要はない．Moraine 上で開発するだけである．また，開発者は MAME の存在を気にする必要はない．

MAME は，版管理システムの履歴から情報を取得するという意味では，本研究と同様の目的を達成するものであるが，利用者が閲覧したいソースコードそのものを取り出す機能は備えていない．

CVSSearch

版管理システムを利用したソースコードの検索システムとして，CVSSearch [4] がある．CVSSearch では，コミットログをリポジトリ内のコードに埋め込むことによって，目的のソースコードを検索するものである．コミットログは，新しいリビジョンをリポジトリに格納するときに付与するコメントのことである．CVSSearch では，コミットログはソースコード中に書かれているコメントに比べ，機能追加やバグの修正等の内容が的確に記述されているとしている．そこで，CVS リポジトリに格納されているソースコードの各行に，変更が加えられたリビジョンのコミットログを”CVS comment” として埋め込み，この”CVS comment” を用いてソースコードの検索を行い，開発者の支援を行っている．

しかしこの手法は，コミットログを検索の対象にしているため，有効性はコミットログの質に大きく依存する．ソースコードに比べ，コミットログは的確に記述されていなくてもプログラムの動作には影響がない．従って，開発者がコミットログを的確に記述していない可能性がある．このような場合には，CVSSearch を用いて必要な情報を検索するのは困難に

なる．

3.2 履歴閲覧ツール

リポジトリ内に保存されているファイルの履歴情報を Web ブラウザ経由で視覚的に閲覧できる CVSWeb [7] や ViewCVS [34] といったツールが開発・利用されている．これらのツールは，簡単なナビゲーション機能を持ち，リポジトリ内にあるファイルの各リビジョンの内容を表示したり，リビジョン間の差分を色付で表示することができるなどの機能を持つ．

しかし，各リビジョンの内容を検索する機能に乏しいため，あるプロジェクトの開発・保守作業において，それとは別のプロジェクトで用いられたリポジトリの中にあるソースコードの中から，利用者が目的としたソースコードを探し出すのは非常に難しい．

3.3 プログラム変更履歴検索システム (CoDS)

ソースコード変更履歴検索システム (CoDS) は，CVS リポジトリに蓄積されたソースコードの変更履歴をデータベース化する．開発者がソースコード片を入力することにより，それらの情報の中から類似したソースコードの変更履歴を検索することが可能となる．開発者は，手持ちのソースコード片を用いて検索を行うことで，開発者のソースコード修正作業を支援する．

しかし，修正するファイルに対する関連ファイルを検索することはできないため，バグの修正が複数のファイルにまたがる場合には修正作業を十分に支援することは難しい．

3.4 ソースコード・メールの履歴対応表示システム (SPxR)

ソースコード・メールの履歴対応表示システム (SPxR) は，CVS のリビジョン情報と電子メールから情報を取得し管理する．また，それらの情報から統合情報を生成し，管理する．これらの情報を開発者に提供することで，開発者は必要なファイルの情報とその関連情報の習得を可能にし，開発者の作業を支援する事ができる．これらのツールは，利用者自身が従事しているプロジェクトの履歴の閲覧には有用である．

しかし，データを検索する機能を持ち合わせていないため，開発に関わっていないプロジェクトの情報を検索する場合，表示された膨大なデータの中から特定のデータを検出することが難しく利用者が求める情報を的確に検索する事が難しい．

4　リビジョン情報と電子メールの検索システム

本研究では版管理システム CVS を用いた分散ソフトウェア開発の一例であるオープンソース開発を対象として，版管理内の履歴情報とメーリングリストから開発者が必要とする情報を容易に取得するための開発支援環境の構築を目指している．本節では、試作したリビジョン情報と電子メールの検索システムシステムについて説明する．

4.1　検索対象

本ソースコード検索システムでは，検索の結果得られたファイルと関連のあるファイルを検索する事で開発者が必要な情報を取得する事ができる．以下に示す 3 種類の方法で必要な開発情報とその関連情報を検索する．

- ソースコードを用いた検索
  それらの情報にソースコード片を与える事で，類似するソースコードを検索し，関連する CVS 情報を表示する．
- CVS 情報を用いた検索
  CVS 情報をキーとして CVS と電子メールの履歴情報の中から一致するデータを持つデータを表示する．
- 電子メール情報を用いた検索
  電子メール情報をキーとして CVS と電子メールの履歴情報の中から一致するデータを持つデータを表示する．

表示された情報は，そのままキーとして次の検索を行なう事ができる．そのようにして利用者は対象となるデータを絞り込む事が可能である．

対象となるデータの説明は次節で行なう．

4.2　データベース

このシステムは 4 つのデータベースを用いることにより情報の検索を行なう．

- 類似コード検索データベース
- CVS 情報データベース
- 電子メール情報データベース

- 統合用データベース

これらのデータを，あらかじめデータベースにしておくことで CVS ツールとして以下のような利点がある．

- リビジョン情報を取得するコストが大幅な削減が可能

  既存のシステムは，毎回必要なリビジョン情報を CVS リポジトリから取得し，リビジョン情報を提供するため，これらのシステムを利用すると過去に取得した情報と同じ情報を再度取得することが多く，そのコストは無視できない．

  それに対して，CVS 情報をデータベース化しておけば，一度取得した情報を CVS リポジトリから再度取得することなく，有効に再利用できる．また，新たに追加されたリビジョンがある場合には，その差分情報だけを取得すればよいため，リビジョン情報を取得するコストが大幅に削減できる．

- リビジョン情報に必要な情報を付加して管理することが可能

  既存のシステムは，必要なリビジョン情報を CVS リポジトリから取得するため，リビジョン情報に必要な情報を付加して管理することは困難である．

  それに対して，リビジョン情報をデータベース化することにより，リビジョン情報に必要な情報を付加して管理することが可能である．

- リビジョン情報の検索能力が向上する

  既存のシステムでは，毎回必要なリビジョン情報を CVS リポジトリから取得するため，CVS リポジトリ内のリビジョン情報に対する検索を行うことは可能であるが，そのコストは非常に大きくなる．

  それに対して，リビジョン情報をデータベース化することにより，その検索能力は飛躍的に向上し，利用者の要求に応じた情報を提供することが可能となる．

類似コード検索データベース

データベースの作成では，$D(F, p, q)$ を，利用者が指定したリポジトリ（またはリポジトリ内のディレクトリ）内にあるすべての C 言語のソースファイルから収集する．

1. **ファイル名** F

   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．

2. **リビジョン番号の対 p,q**

   1 のどのリビジョン間における変更の情報であるかを特定するための情報である．

3. **q のコミット日時**

   各リビジョンの前後関係を直観的に判断できるように，リビジョン q のコミット日時の情報を付与する．

4. **q のログメッセージ**

   コミットログは，直前のリビジョンからの差分を自然語で記述したものであり，直前のリビジョンとの間でどのような変更を行ったのかを理解する助けとなると考える．コミットログを情報として利用者に提示することにより，検索結果が目的としているものであるかどうかを利用者が判断することができる．

5. **リビジョン p において，次のリビジョン q までに変更された部分（前後の数行を含む）） $c(F, p, q)$ に関する差分情報**

   ソースコード片を検索する際，入力されたソースコード片 I とデータベースのこの部分をトークン単位で比較する．その結果，I と類似したトークン列であると判定されたコードを持つものを検索結果として出力する．検索の効率化のため，c を字句解析（4.2.1 節参照）し，トークン列に変換したものを格納する．また，ある程度まとまった単位で意味のある比較を行うことができるように，UNIX の context 形式の diff 出力を用いて $c(F, p, q)$ を取得する．context 形式の diff では，2 つのテキストの間で相違する行を，その前後数行を含めて出力する．相違する行が離れて存在する場合は，いくつかのブロックに分けて出力される．context 形式の diff 出力の一部を図 5 に示す．

CVS 情報データベース

CVS リポジトリからリビジョン情報を取得し，CVS 情報データベースを作成する． CVS の固有情報は RCS 形式のリビジョン情報を解析することにより取得する．

1. **内部識別番号**

   データベース作成時に各レコードごとに個別の識別番号をふり当てる．CVS 情報の処理を簡単化するために用いる．

2. **ファイルパス**

   ファイルを一意に特定する事ができるように，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．

```
*** 3236,3259 ****
  if (String_table[cs__]) /* scrolling region */
  {
-     list[1] = 0;
!     list[0] = LINES;
      String_Out(String_table[cs__], list, 2);
      Curr_y = Curr_x = -1;
! }

  top_of_win = curscr->first_line;
--- 3384,3407 ----
  if (String_table[cs__]) /* scrolling region */
  {
!     list[0] = LINES - 1;
      String_Out(String_table[cs__], list, 2);
      Curr_y = Curr_x = -1;
!     }

  top_of_win = curscr->first_line;
+ curr = top_of_win;
```

図 5: context diff の例

3. **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．CVS 情報の中から特定の CVS 情報を指定する場合にはファイルパスとリビジョン番号を用いる．

4. **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間に変更されたファイルを検索するために用いる情報．

5. **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．

6. **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，出現頻度が高いファイルパスをキーワードとして取得する．

7. **関連情報識別子**
   CVS 情報の中で「同一時間に更新された」「同一更新者に更新された」「キーワードが同じ」などの共通点がある場合，それらの内部識別番号を関連識別番号として登録する．利用者が関連した情報を必要とする場合はこの番号をもとに情報を抽出する．

電子メール情報データベース

電子メールアーカイブに蓄積された電子メールを解析し，必要な情報を抽出していく．ここで対象になる電子メールは以下に示す「コミットメール」と「議論メール」に分けられる．

- コミットメール
  CVS リポジトリのリビジョンが更新された際に，CVS から送信されるメッセージである．コミットメールは更新されたリビジョンの更新情報が記述されていて更新されたリビジョンに関する情報を開発者が容易に入手することができる．

- 議論メール
  コミットメール以外の全ての電子メールである．議論メールには，開発作業に関連する一般的な話題が記述されていて，開発作業の内容理解に役立つ．

データベース内のそれぞれのデータ構造は以下の通り．

1. **内部識別番号**
   データベース作成時に各レコードごとに個別の識別番号をふり当てる．mail 情報の処理を簡単化するために用いる．

2. **送信者**
   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．

3. **送信日時**
   ファイルが送信された時刻．直観的にメールの時間的順序を認識することができ、受信したメールの中から必要なメールを大まかに検索するのに用いられる．

4. **サブジェクト**
   ファイルの概要を表す情報．開発者が最も用意にメールの内容をチェックできると思われる情報.

5. **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，出現頻度が高いファイルパスをキーワードとして取得する．

6. **関連情報識別子**
   コミットメールに対する返信メールのような，ある電子メールと関連のある電子メールに対してそれらの内部情報識別子をお互いに関連情報識別子として登録する．

7. **更新作業者**（コミットメールのみ）
   実際に CVS に登録している名前が登録されているため CVS 情報との統合の際に有効となる情報.

8. **更新日時**（コミットメールのみ）
   実際に CVS に登録した時刻が登録されているため CVS 情報との統合の際に有効となる情報.

9. **更新ファイルパス・リビジョン番号**（コミットメールのみ）
   実際に CVS に登録したファイルパスとリビジョン番号が登録されているため CVS 情報との統合の際に有効となる情報.

統合情報データベース

上記の CVS 情報生成部と電子メール情報生成部の情報の中から互いに関連性を持つ情報をそれぞれのデータベースから取り出して比較し，関連性があればそれらの情報を統合し，統合用データベースに格納する．統合情報生成部で登録の対象となる情報は以下の二つ.

- **内部識別番号**
  電子メール情報生成部の内部識別番号と同じで，関連する CVS 情報があれば登録する.

- **関連識別番号**
  CVS 情報生成部の内部識別番号と同じで，関連する電子メール情報があれば登録する.

電子メール情報と CVS 情報の関連性は以下の判断基準をもとに判定するようにした.

- 「更新日時」が一致する
  コミットメールから取得された電子メール情報の場合は,「更新日時」のデータを持っているので，CVS 情報の「更新日時」と比較することが可能である．同一か時差が±5 分の場合は関連性をもつと判断する.

- 「ファイルパス」と「リビジョン番号」が一致する
  コミットメールから取得された電子メール情報の場合は,「ファイルパス」「リビジョン番号」のデータを持っているので，CVS 情報の「ファイルパス」「リビジョン番号」と比較することが可能である．両者が一致すれば関連性を持つと判断する.

- 「キーワード」が一致する
  電子メール情報と CVS 情報は互いに「キーワード」のデータを持っており，これらを比較する．両者が一致すれば関連性を持つと判断する.

ここでは，CVS と電子メールの関連が分かれば良く，さらに詳しい情報はそれぞれのデータベースから抽出すれば良いので，ハッシュデータベースでキーを値それぞれに内部番号識別子と関連情報識別子を格納する．

4.3 構成概要

本システムは 3 つの部分から構成される．

- 類似コード検索部 (CoDS 部)
- 開発履歴データベース管理部 (SPxR 部)
- データ表示部

4.4 類似コード検索部 (CoDS 部)

類似コード検索部は，類似コード検索 DB（前述），データベース作成ツール，字句解析ツール，トークン比較ツールの 4 部分からなる．利用者はあらかじめデータベース作成ツールから類似コード検索 DB に版管理システムのリビジョン情報を登録する．その後，ソースコード片を入力するとトークン比較ツールで入力コードをトークンに分解し，類似コード検索ツールで類似検索データベース中の差分情報との類似度を計測し，一定以上の類似度を持つソースコードのデータをデータ表示部へ渡す処理を行う．

字句解析

入力されたソースコード片とデータベース内にあるソースコード片との比較をトークン単位で行うために，比較の前にソースコード片を字句解析し，トークン列に変換する．

字句解析では，ソースコード片をプログラミング言語の文法に従って，整数で表されたトークン列に変換する．このとき，ソースコード中の空白とコメントは無視する．さらに，実用的に意味のある比較を行うために，ユーザ定義の変数，関数，型等は名前が異なっていても等価なトークンとみなす．言語仕様やライブラリ等で定義されている手続き・関数等の名前についてはそれぞれを別のトークンに変換する．これらのトークンと予約語を合わせて，本論文では「キートークン」と呼ぶ．出力の際には，整数で表されたトークンの番号に加え，そのトークンが出現する行番号も付加する．

さらに，データベースの差分情報には，検索効率を向上させるための工夫として，当該ソースコード片に出現するキートークンを列挙したものをこの部分に付加する．このフィールドの使用法については後述する．

トークン比較

ソースコードの比較部分では，上記の字句解析によって変換されたトークンを 1 つの文字とみなし，文字列の一致問題を解くことにより，類似コードの検索を行う．

ここでは CoDS[29] 同様，類似部分を抽出する系列比較アルゴリズムとして，” 局所アラインメント [14]” を用いている．局所アラインメントは，与えられた 2 つの系列の中のそれぞれの部分系列のうち等価で最長のものを 1 つ求める問題である．ここでいう等価な部分系列は，完全に一致している必要はなく，異なる要素が存在してもよい．このアルゴリズムを用いることによって，2 つのソースコード片の間で，互いに類似している部分を含んでいるかどうかを調べることができる

局所アラインメント

局所アラインメントは，2 つの文字列の任意の部分文字列間で，スコア最大のアラインメントを求めるアルゴリズムである．例えば，

```
S = abcbccbbb
T = cdcbcdccba
```

上のような S と T とを一致のスコアを上の定義により求めると，以下のようになる．

```
S′ = bcbccb
T′ = bcdccb
スコア = 5 - 1 - 0 = 4
```

文字列 S と T の間の局所アラインメントは，次の式で表される最大のスコア $MaxSlocal$ を持つアラインメントを求める問題として定式化できる [28].

$$MaxSlocal = \max\{Sc[S[i..n], T[j..m]] : 1 \leq i \leq n \leq |S|, 1 \leq j \leq m \leq m \leq |T|\}.$$

局所アラインメントを求めるアルゴリズムを以下に示す.

- 入力は，2 つの文字列 $S[i], T[j]$（$1 \leq i \leq |S|, 1 \leq j \leq |T|$）
- 出力は MaxSlocal と，$S_{local}[i, j]$（$1 \leq i \leq |S|, 1 \leq j \leq |T|$）
- $S_{local}[0, 0] \leftarrow 0$
- i を 1 から $|S|$ まで変化させながら
  - $S_{local}[i, 0] \leftarrow 0$

- j を 1 から $|T|$ まで変化させながら
  - $S_{local}[0, j] \leftarrow 0$
- $MaxSlocal \leftarrow 0$
- j を 1 から $|T|$ まで変化させながら
  - i を 1 から $|S|$ まで変化させながら
    * $S_{local} \leftarrow \max\{0, S_{local}[i-1, j]-\delta, S_{local}[i-1, j-1]+\sigma(S[i], T[j]), S_{local}[i, j-1]+\delta\}$
    * $MaxSlocal < Slocal[i, j]$ ならば $MaxSlocal \leftarrow Slocal[i, j]$

このアルゴリズムの時間計算量は $O(|S| \cdot |T|)$ である．このアルゴリズムを用いてトークン列の比較を行い，スコアを出力する．

検索効率の向上

局所アラインメントを用いると，時間計算量が 2 文字列の長さの積のオーダーと大きく，膨大な回数の比較を行う場合，実用的な時間で検索が行われない．そこで，トークンの比較を時間効率よく行うため，利用者は，入力としたソースコードの中で重要と思われるキートークンを「特性キートークン」として指定する．指定がない場合は，利用者から入力されたソースコード片の中の全てのキートークンを特性キートークンとする．なお，キートークンが含まれていないソースコード片については，アラインメントを求める対象としない．

なお，類似度の判定は CoDS と同様の基準を採用している．

4.5 開発履歴データベース管理部 (SPxR 部)

この部分はさらに，CVS 情報管理部，電子メール情報管理部，統合情報管理部の 3 つの部分に分けることができる．それぞれの部分はデータベース (前述) とデータベース作成ツールからなる．あらかじめ版管理システム CVS と，電子メールとそのアーカイブから情報を取得しデータベース化する．また，それらの統合情報を生成して統合用データベースを作成し管理する．データ表示部から検索する際にそれらの情報を開発者に提供することで，開発者の作業を支援する．

4.6 データ表示部

この部分は開発者と各履歴データベースを結ぶインターフェイスの役割を果たしている．開発者から検索要求を受けると履歴データベースと連結し，要求されたデータを表示する．

類似コード検索結果表示

提示するのに必要な情報は，以下の通り．

1. ファイル名とリビジョン番号の組 (F, p, q)
2. q のコミット日時・更新者・ログメッセージ
3. I と類似しているとみなされた，リビジョン p における部分トークン列

検索結果はファイル毎にまとめてファイル名 F を一覧で表示する．F を 1 つ選択すると，検索結果の (1) および (2) のうち，該当するファイルに対するものを一覧表示する．その中の各リビジョンの対 (p, q) を選択することにより，p から q への差分を表示するようにする．このとき，リビジョン p において，I と類似しているとみなされた部分を強調表示する．

CVS 情報・電子メール情報検索結果表示

- CVS 情報，電子メール情報の検索と表示

  利用者から CVS 情報・電子メール情報の一部をキーとして与えられた際に，SPxR 部に一致する情報あるかを検索する．一致する情報があればその CVS 情報・電子メール情報の一覧を表示する．検索された CVS 情報，電子メール情報をキーとして検索を行なう事により，必要とする情報を絞り込む事も可能である．

- 統合情報の表示と検索

  CVS 情報，電子メール情報を表示した上で，統合情報を利用して，それらの間で相互に存在する関連を表示することにより，開発者に統合情報を提供する．また，統合情報の検索を行い，開発者が必要とする情報を提供する．

本システムを利用する事により，開発者は特定のソースコードや開発履歴情報からそれらに関連する開発情報を入手することで開発のコストを軽くすることができる．

5　システムの実装

本節ではこれまで述べた手法に基づくリビジョン情報と電子メールの検索システムの実装を行なった．実験環境は以下の通りであり，図 6 に本システムの構成を示す．

- CPU:PentiumIV 1.5GHz
- RAM:512MB(SDRAM)
- OS:GNU/Linux
- データベース:GNU GDBM, PostgreSQL7.2, BerkelyDB
- WEB サーバ:Apache 1.3.27

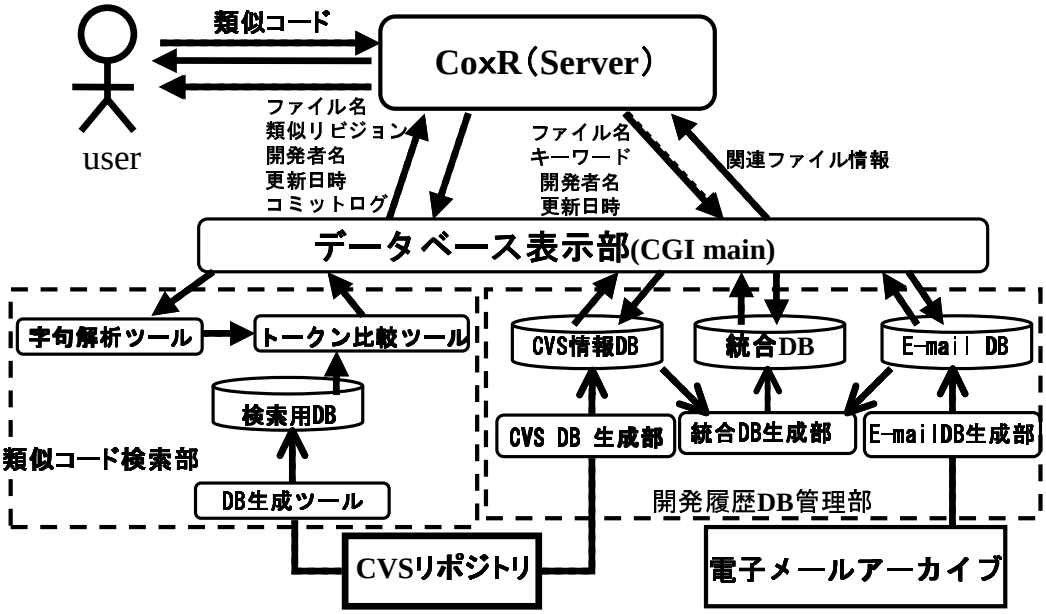

図 6: システム構成図

5.1　類似コード検索部

類似コード検索部は，C 言語で書かれたソースコードを対象とし，版管理システムとして CVS を用いていることを前提としている．また，内部で RCS のコマンドを呼び出しているため，RCS がインストールされている必要がある．

データベース作成ツール (getdb.pl)

Perl 言語を用いて CVS リポジトリから必要なデータを取り出し，データベースとして保存するツール作成した．本データベース作成ツールでは，指定されたディレクトリ（サブディレクトリも含む）にある C 言語のソースファイルの履歴ファイル（"*.c,v"ファイルおよび"*.h,v"ファイル）に関してデータベース作成の処理を行っている．また，Perl でデータベースを扱うルーチンは，ハッシュとファイルを結びつけることができるように GNU GDBM 1.8.0 を用いる．全体で 900 行程度のプログラムで、処理の流れは以下の通りになっている．

- 指定されたディレクトリ以下にある（サブディレクトリ内も含む）C 言語のソースファイルのリストを取得する
- リストの各ファイル f について，以下の処理を行う
  1. f のリビジョンツリーを取得する．f が 1 つしかリビジョンを持たない場合は，リストの次のファイルの処理に移る
  2. 1 で得られた f のリビジョンツリーをたどりながら，次のリビジョンとの差分を context 形式で取る．コメント部分は除去しておく．
  3. 2 で得られた f の各隣接リビジョン間 (p, q) の差分出力から，リビジョン p のコードを抜き出し，それを字句解析ツールを用いてトークン列に変換する．
  4. 3 のトークン列から，キートークンを別に抽出する．
  5. f のリビジョン q のコミット時のログメッセージを取得する．
  6. f のリビジョン間別に 1 つのハッシュの要素として書き込むことにより，データベースのレコードを追加する．この時，f の 1 つのリビジョン間に 1 つのハッシュの要素が対応するようにする．

字句解析ツール (scanner.c)

このツールは，C 言語で記述されたソースコードを入力とし，上で述べた仕様に基づいて，トークンのリストを出力する．ソースコードの比較の際に，マッチした部分が分かるように，トークンのリストには，入力されたソースコード中における行番号を付加して出力する．また，キートークンとして，C 言語のキーワード，標準ライブラリ関数名を用いる．本字句解析ツールは全体で 1000 行程度のプログラムで C 言語を用いて実装を行った．

トークン比較ツール (mathcing.c,alignment.c)

このツールは入力された 2 つのトークン列 S, T の間でアラインメントを行い，最大のスコアと，マッチしている部分の（トークン列 T の元のソースコード片における）行番号の範囲を出力する．全体で 500 行程度で C 言語で実装している．

5.2 開発履歴データベース管理部

先に述べたように開発履歴データベース管理部は CVS 情報管理部，電子メール情報管理部，統合情報管理部の三つの部分に分かれている．なお，これら三つの部分はそれぞれ設定ファイル (.commsysconf) で設定を管理する．

ただ，”.commsysconf”で検索対象となるリポジトリを一意に定めてしまうと，関連情報が膨大になり過ぎて有効な情報が得られなくなる可能性があるので，利用者が一定の大きさのリポジトリ内を検索対象として指定することにより”.commsyconf”の設定を変えることができるようにした．

CVD 情報管理部

この部分は CVS 情報生成ツールと CVS 情報データベースから生成される．

- CVS 情報生成ツール (cvsinfogen3.pl)
  CVS 情報生成ツールの処理は以下の通り．全体は 600 行程度で perl 言語で実装した．
  - 設定ファイル (`.commsysconf`) に指定された CVS リポジトリのルートディレクトリ等の設定情報を読み込む．
  - ルートディレクトリから，CVS リポジトリ内のディレクトリ構造を再帰的に辿って，各 RCS ファイルの解析を行い，リビジョン情報を取得する．
  - 取得したリビジョン情報から CVS 情報を生成して，CVS 情報データベースに登録する．

  CVS 情報の取得方法として，引数に新規作成 `create` と更新 `update` を指定することができる．`create` が指定された場合には，CVS リポジトリから全てのリビジョン情報を取得して CVS 情報を生成し，CVS 情報データベースに登録する．

  `update` が指定された場合には，前回の情報取得時から後に更新作業が行われたリビジョン情報だけを取得して CVS 情報を生成し，CVS 情報データベースに登録する．

- CVS 情報データベース
  CVS 情報データベースには，PostgreSQL と，perl5 の BerkeleyDB を利用した．”.commsysconf”から取り出した検索対象となるリポジトリに応じてそれぞれ別々のデータベースを作成する．

  開発者からデータ要求を受けると，まず開発者からハッシュデータベースより受け取ったデータを満たす内部識別番号の集合を探す．その後，その内部識別番号をもとに PostgreSQL から各レコードが持つ情報を表示する．

  - PostgreSQL
    ・主キーを内部識別番号とし，前述の 7 つの情報をレコードとして格納.
  - BerkeleyDB
    ・キー…更新日時・更新作業者・キーワード・関連情報識別番号
    ・値…内部識別番号

電子メール情報管理部

電子メール情報管理部は，電子メール情報生成ツールと電子メール情報データベースから構成される．

- 電子メール情報生成ツール (mailinfogen3.pl)
  電子メール情報生成ツールの処理概要は以下の通り．全体は 800 行程度で perl 言語で実装した.

  - 設定ファイル (`.commsysconf`) に指定された電子メールアーカイブのルートディレクトリ等の設定情報を読み込む.
  - ルートディレクトリから，電子メールアーカイブのディレクトリ構造を再帰的に辿って，各電子メールの解析を行う.
  - 電子メール情報を生成して，電子メール情報データベースに登録する.

  電子メール情報の取得方法として，CVS 情報の場合と同様に，引数に新規作成 `create` と更新 `update` を指定する.

  `create` が指定された場合には，電子メールアーカイブから全ての電子メールに関する情報を取得して，最初から電子メール情報を生成し，電子メール情報データベースに登録する.

`update` が指定された場合には，前回の情報取得時から後に送信された電子メールに関する情報だけを取得して，電子メール情報を生成し，電子メール情報データベースに登録する．

- 電子メール情報データベース
  電子メール情報データベースにも，PostgreSQL と，perl5 の BerkeleyDB を利用した．

  コミットメールの場合,「更新ファイルパス・リビジョン番号」は複数行含まれる場合が多い．統合情報を作成する際には全てのファイルパスを必要とするが，データ表示の際はそのデータは必要ない．そのままメールを解析するとメールを解析する際に内部情報識別子は主キーとして設定することができないので「更新ファイルパス・リビジョン番号」のデータを用いる統合用データベースでは主キーを設定せず作成し,「更新ファイルパス・リビジョン番号」の情報を用いない表示用データベースは内部識別番号を主キーとしてを用意しデータ表示の高速化を図る。

  以上を踏まえてこれらのデータは検索・表示の際に効率良く処理できるように以下の三つのデータベースを用いる．データ統合の際とデータ表示の際に扱う情報は効率化のため分けて設計した．

  – ハッシュデータベース
    ・キー…更新日時・更新作業者・キーワード・関連情報識別番号
    ・値…内部識別番号
  – PostgreSQL を用いたデータベース
    ・主キーを内部識別番号とし，全ての電子メール情報を格納する．情報をレコードとして格納（データ表示用）．
    ・主キーを設定せず，コミットメールの情報をレコードとして格納（データ統合用）．

統合情報管理部

統合情報管理部は，統合情報生成ツールと統合情報データベースから構成される．

- 統合情報生成ツール (integinfogen3.pl)
  統合情報生成ツールの処理概要は以下の通り．全体は 300 行程度で perl 言語で実装した．

  – 設定ファイル (`.commsysconf`) から，統合情報に関する設定情報を読み込む．

– CVS 情報データベースから CVS 情報を，電子メール情報データベースから電子メール情報を取得して，両者のデータを比較する．
– 相互に関連を持つ情報である場合には，両者を結合して統合情報を生成し，統合情報データベースに登録する．

統合情報の生成方法は，CVS 情報・電子メール情報の場合と同様に，新規作成 `create` と更新 `update` を指定する．

`create` が指定された場合には，全ての CVS 情報・電子メール情報が統合情報を生成する対象となり，最初からデータベースを作り直す．

`update` が指定された場合には，前回の統合情報生成時から後に追加された CVS 情報・電子メール情報が，統合情報を生成する対象となり，データベースに情報が追加される．

- 統合情報データベース
  電子メール情報データベースには，perl5 の BerkeleyDB を利用した．
  CVS 情報データベースとと電子メール情報データベースを検索する際に，関連する一方の内部識別番号を検索することができればその内部識別番号をもとに CVS 情報データベース・電子メール情報の情報を登録する．

5.3 データ検索・表示部

データ検索・表示部では開発者からの検索要求を受けて，類似コード検索部・開発履歴データベース管理部と連係し開発者が必要とする情報を参照する．以下ではそれぞれの検索方法に対する処理について説明する．このシステムの実装は全て Perl 言語で行なった．トップ画面を図 7 に示す．

類似コード検索

入力コードと特性キートークン・検索対象となるリポジトリを与えることで検索を開始する．詳しい処理は以下の通り．

1. 利用者から入力されたソースコード片 I を字句解析ツールを用いてトークン列 T_I に変換する．
2. あらかじめキートークンが指定されていなければ，T_I において，現れるキートークン K_I を全て抽出する．キートークンがなければエラーとして終了する．

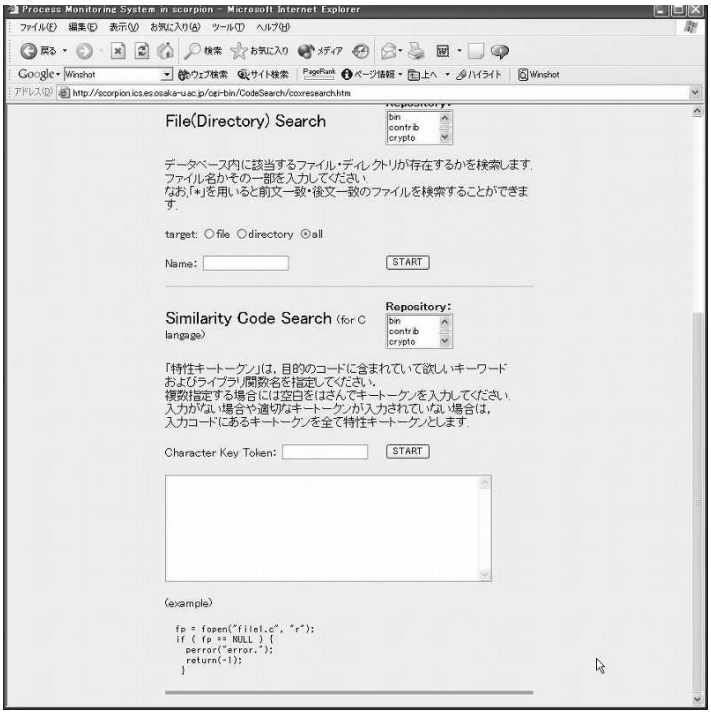

図 7: トップ画面

3. T_I のトークン数に応じて，アラインメントスコアの閾値 α を計算する．

4. データベースの各レコードについて，

   - レコードからトークン列部分を取り出す．このときの各トークン列を t とする．
   - 各 t について，
     - K_I が t に存在しなければ，何もせず次の t について処理.
     - K_I が t に存在する場合，T_I と t を入力としてトークン比較ツールを実行する．このときのアラインメントのスコアを Sc とする．
     - $Sc \geq \alpha$ ならば，マッチした行番号の範囲の情報を出力部に送り，次のレコードについて処理.

検索結果はまず，ファイル一覧を表示する (図 8 参照)． 利用者はファイルパスや，類似したリビジョンの数を参考にファイルを選択する (図 9 参照)．ファイル情報には類似コード該当部分の他に (図 10)，更新者，更新日時，そのファイルのリビジョン情報全体の情報が含まれるため，連動した検索が可能である．詳しくは以下の項目を参照のこと．(検索対象のリポジトリは最初に選択したものを引き継ぐ)

また，出力の際に一時的なデータベースを作成し，まず，そのデータベースに検索結果を登録して後からでも参照できるようにした．この検索結果を利用して上記の方法で再び類似コードを探索することも可能である．

前述の通り，実装は Perl 言語で行ない，Web インターフェースで利用できるよう，CGI を用いている．全体は 150 行程度のプログラムである．

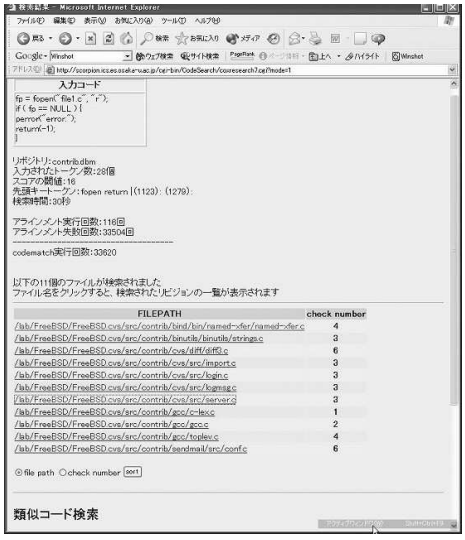

図 8: 類似コードファイル一覧

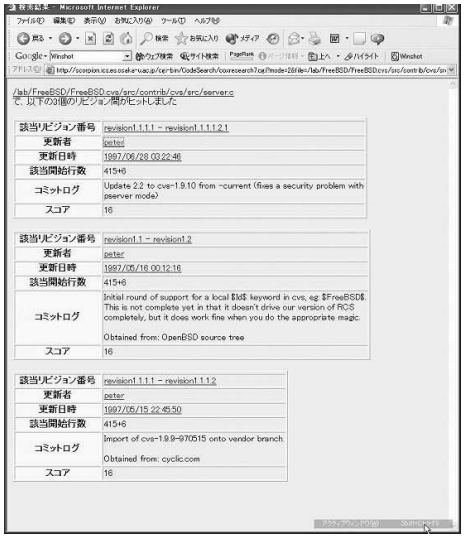

図 9: 該当リビジョン情報

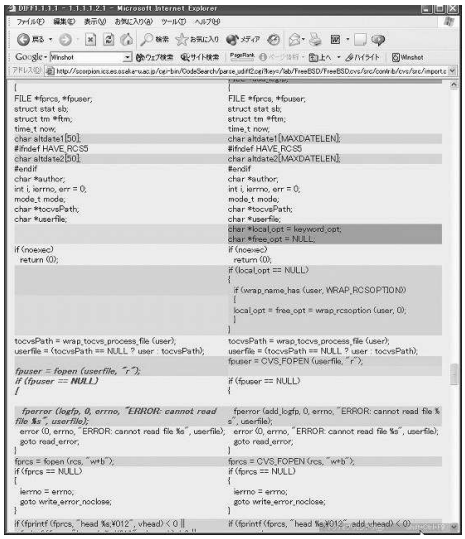

図 10: 類似部分表示

CVS 情報・電子メール情報検索

- ファイル・ディレクトリ名検索
  検索対象となるリポジトリを選択した後で，UNIX コマンドの”find”を用いて指定されたディレクトリ以下で該当するファイル・ディレクトリを検索する．CVS にコミットされたファイルには”,v”という拡張子がつくが，実際のファイル名を検索する際にはそれを付けて検索することは有用ではないのでファイル名を検索する際にはシステム側が”,v”を付加して検索するようにした．(図 11 参照) 出力される filename を抽出してそのデータを元に類似コード検索も可能である．その場合検索対象となるリポジトリは最初に選択したものをそのまま引き継ぐ．

  Web インターフェースで利用できるよう，CGI を用いている．全体は 200 行程度のプログラムである．

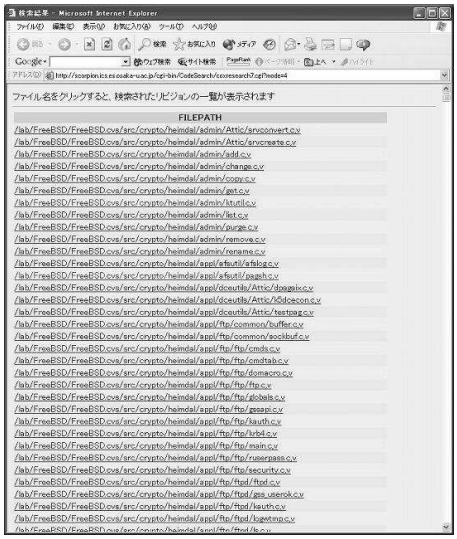

図 11: ファイル名検索一覧

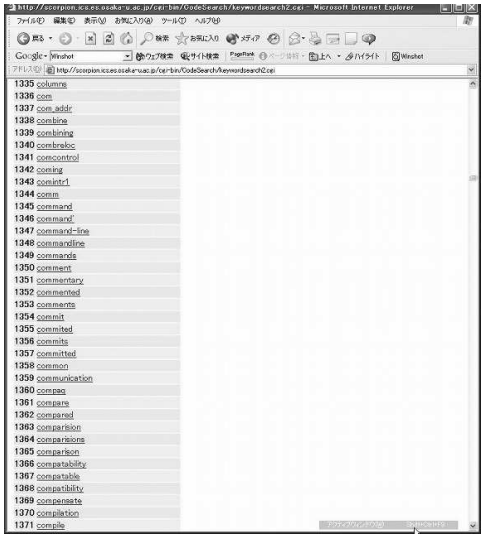

図 12: キーワード一覧

- キーワード検索
  利用者が要求したキーワードを CVS 情報データベース・電子メール情報データベースの BerkeleyDB から該当するキーを持つデータを探索する．キーを持つデータがあればその値を PostgerSQL に与えレコードを全て表示する．(図 13 参照)

  なお，システムのトップからキーワード検索を行なう場合，図 12 のように CVS 情報データベース・電子メール情報データベースに登録されているキーワードの一覧を表示する．

- ファイルパス・電子メールを用いた検索

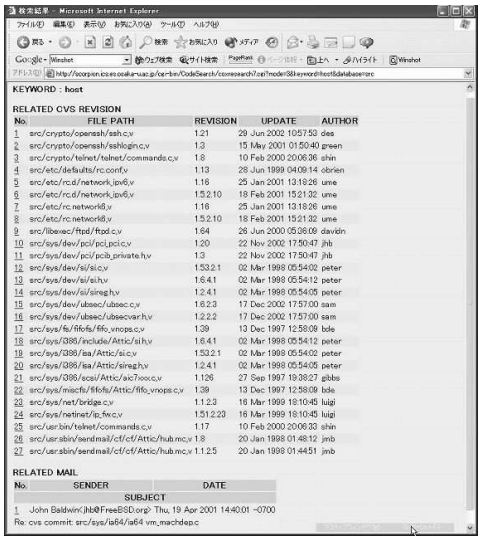

図 13: キーワード関連情報一覧

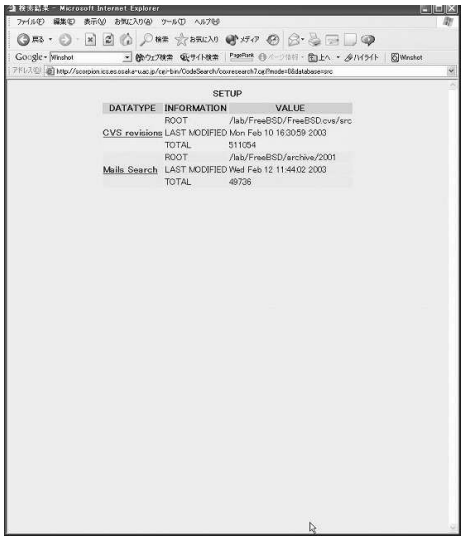

図 14: ファイルパス検索

利用者がおおよその見当をつけてファイルや電子メール情報を検索する場合，直接ファイルパスやメールリストを探すことも可能である.（図 14 参照）

CVS 情報を検索する場合，利用者はディレクトリ構造（図 15）の中から特定のファイル情報 (リポジトリ情報) を取り出すことができる (図 17 参照)．リビジョン情報では開発者・開発日時・ログ・差分の情報を得ることができる．さらにリポジトリ情報の中からさらにリビジョンを選択することでファイルの全文を閲覧することが可能である (図 16 参照).

電子メール情報は受信時間順に表示される (図 18)．一つのメールを選択すると関連するキーワードや CVS リポジトリが表示される (図 19 参照)．利用者が表示するメールリストの範囲を指定することができる.

また，範囲を絞り込むことによる類似コード検索とファイル名・ディレクトリ検索も可能である．それらの情報を用いて関連情報を検索できる．それらの検索方法は以下に示す通り.

以下の二つの検索部分とともに実装した．合わせて 1500 行程度のプログラムである.

- 関連情報検索

  キーワード検索と同様で利用者が要求した関連情報（開発者・更新日時）を CVS 情報データベース・電子メール情報データベースの BerkeleyDB から該当するキーを持つデータを探索する．キーを持つデータがあればその値を PostgerSQL に与えレコード

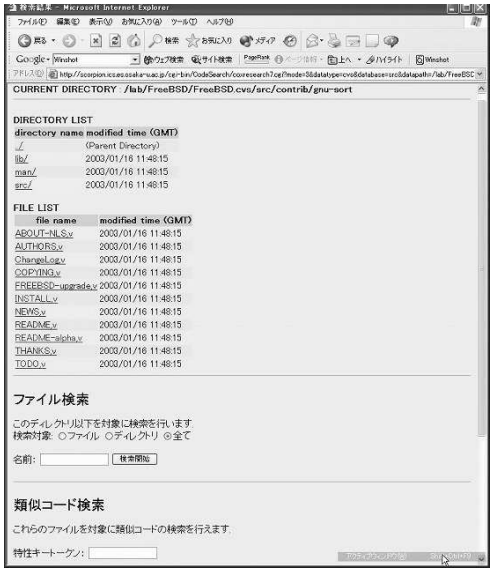

図 15: ディレクトリ情報一覧

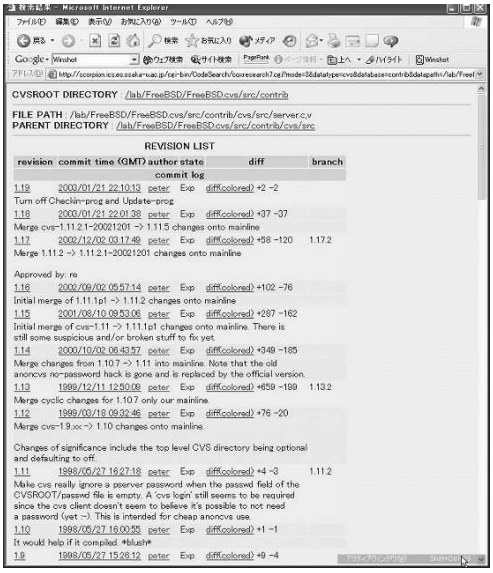

図 16: リポジトリ情報一覧

を全て表示する.（図 20，図 21，図 22 参照）

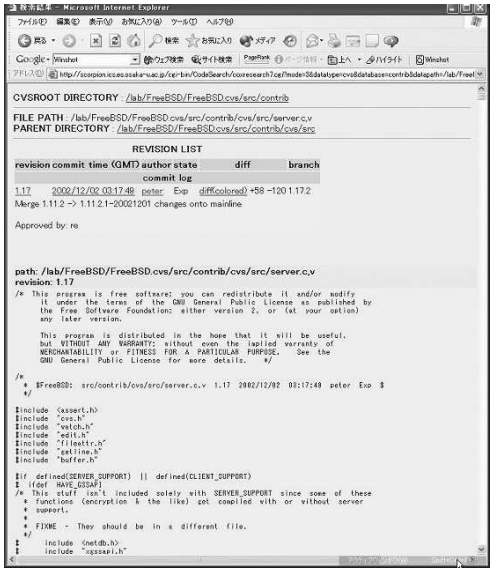

図 17: リビジョン情報一覧

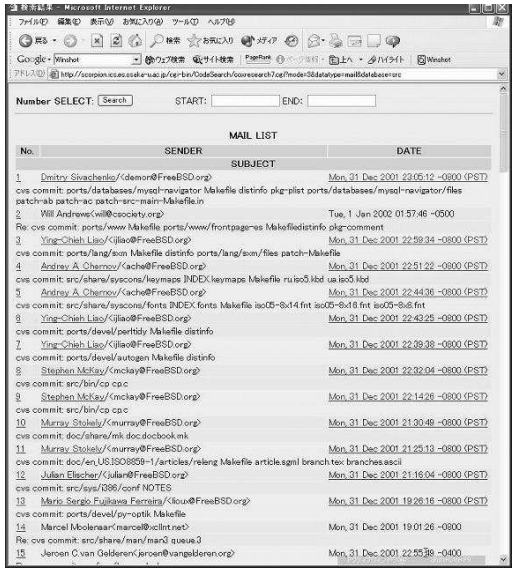

図 18: メール一覧

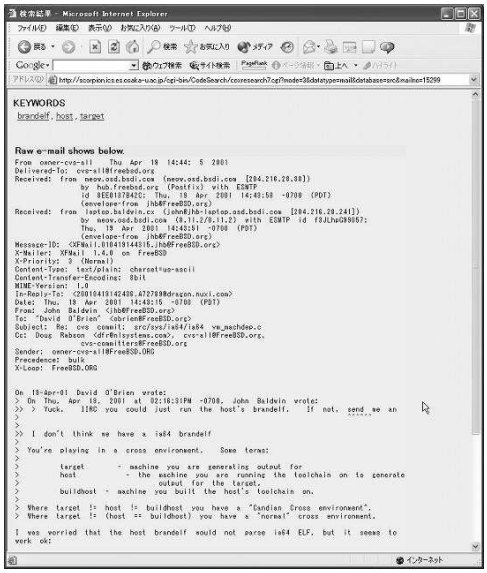

図 19: メール情報一覧

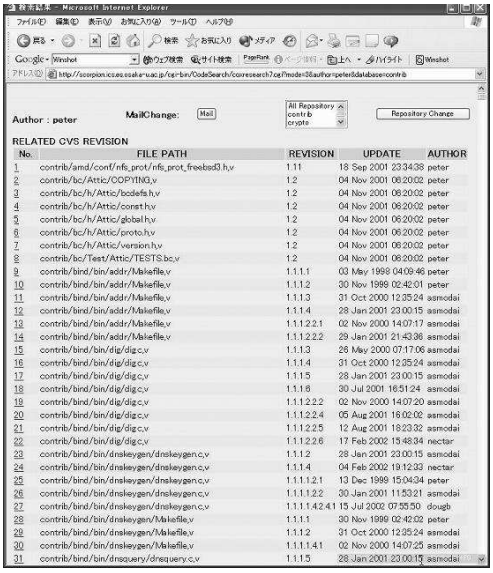

図 20: author 情報 (CVS) 一覧

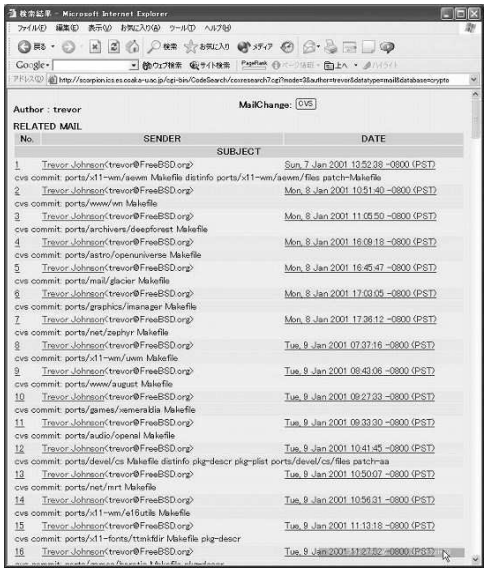

図 21: author 情報 (mail) 一覧

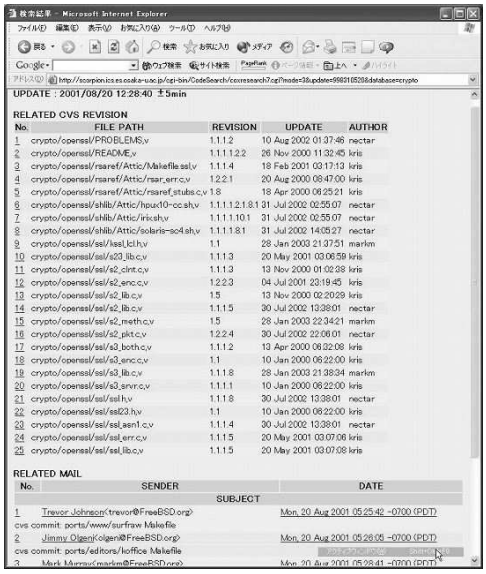

図 22: 更新日時一覧

6 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．

6.1 実験対象

本システムの適用実験を行なうために，実際のオープンソース開発の中からオープンソフトウェアの代表例である FreeBSD[31] の開発をモデルとして仮想的なオープン開発環境を用意する．

- CVS リポジトリ
  FreeBSD の CVS リポジトリをミラーリングして，そこから必要なリビジョン情報を取得する．今回の実験では CVS リポジトリの ROOT を/lab/FreeBSD/FreeBSD.cvs/src（ファイル総数 51379, リビジョン総数 511054）と設定し実験の対象とする．
- 電子メール FreeBSD のメーリングリストアーカイブをミラーリングして，そこから必要な電子メール情報を取得する．今回の実験では/lab/FreeBSD/archive/2001 以下の 49736 件の電子メールを対象とした．

なお，実験環境は以下の通りである．

- CPU:PentiumⅣ 1.5GHz
- RAM:512MB (SDRAM)
- OS:Debian GNU/Linux
- データベース:GNU GDBM, PostgreSQL7.2, BerkelyDB
- WEB サーバ:Apache 1.3.27

6.2 実験の概要

本システムの評価実験として以下の 2 つの実験を行なった．

- 検索実例に基づいたデータ検索
  利用者が FreeBSD 開発の例として OpenSSH を開発していると想定して，パスワードのセキュリティー問題に関する修正事例をもとに類似コードの検索と関連情報を検索を行なう．

- 検索速度評価
  CoDS 単体では検索の際に一つのキートークンしか用いることができなかったが，本システムは複数のキートークンを用いて類似度を測定する範囲をあらかじめ絞り込むことができる．この部分について評価する．

6.3 検索実例に基づいたデータ検索

今，利用者は OpenSSH のパスワードの送信を行なうソースコードの開発を行なっているとする．このコードはパスワードをソケットで送信する際にパスワードの長さもデータとして送信してしまい．これを用いてパスワード攻撃が行なわれる可能性が指摘されたものとする．本システムの利用者はその問題点を解消するためにパケット送信の際に長さを隠蔽子て送信できる者がないかを検索している．
なお，入力として与えるソースコードは以下の通り．

```
if (i != 0)
   error("Permission denied, please try again.");
password = read_passphrase(prompt, 0);
packet_start(SSH_CMSG_AUTH_PASSWORD);
packet_put_string(password, strlen(password));
memset(password, 0, strlen(password));
xfree(password);
packet_send();
packet_write_wait();
```

上のソースコードを用いて類似コードの特定を行ない．その関連情報を参照する事でバグ修正に役立てるものかどうかを検証する．類似したソースコードは複数存在し，図 23 のようになった．

その中で複数の類似コードが検索された検索結果の一つとして表 1 のような類似コードが検索された．

コミットログから「パスワード送信の際に長さが分からないように送る」と記述してあることから，利用者が必要である情報だといえる．しかし，中身を見ると関数の変更 `packet_put_string(response_stren(response)→ssh_put_password(response)` は記載してあるが，関数 `ssh_put_password` についてはこれ以上記載されていない．

そこで，次に関連情報の検索を行なう．ここで考えられる検索方法としては類似コード検索により出力された結果を用いる．検索方法として以下の 4 つを考える．

表 1: 検索結果の例

ファイル名	/lab/FreeBSD/FreeBSD.cvs/src/crypto/openssh/sshconection1.c
リビジョン間	1.7-1.8
日時	2001/03/20 02:06:40
開発者	green
コミットログ	Make password attacks based on traffic analysis harder by requiring that"non-echoed" characters are still echoed back in a null packet, as well as pad passwords sent to not give hints to the length otherwise. Obtained from: OpenBSD

類似コードの例	if (i != 0)
	error("Permission denied, please try again.");
	password = read_passphrase(prompt, 0);
	packet_start(SSH_CMSG_AUTH_PASSWORD);
	ssh_put_password(response);
	* memset(password, 0, strlen(password));
	xfree(password);
	packet_send();
	packet_write_wait();

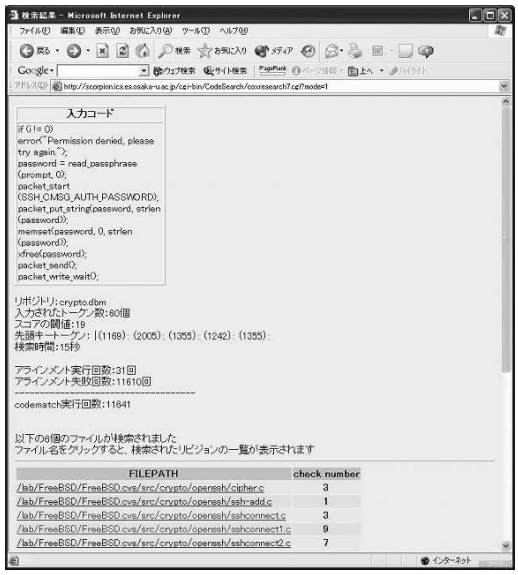

図 23: 検索結果の例

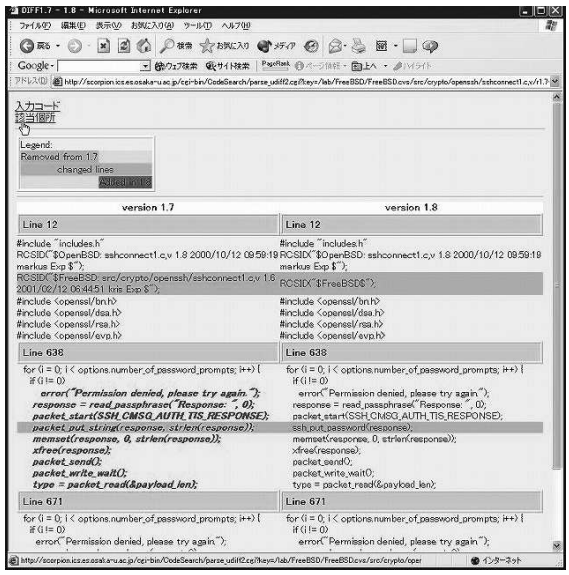

図 24: 差分の表示

1. 開発者 green による検索
2. 更新日時 2001/03/20 02:06:40 前後にコミットされたファイルによる検索
3. それに伴う込とメール情報の検索
4. キーワード「openssh」による検索

その結果，同時にコミットされたファイルの中に（図 25 参照）sshconnection.c（ファイルパス：/lab/FreeBSD/FreeBSD.cvs/src/crypto/openssh/sshconection.c）内に関数 ssh_put_password の内容が記載されている事を検索できた.(図 26 参照)

```
void
  ssh_put_password(char *password)
  {
    int size;
    char *padded;

    size = roundup(strlen(password) + 1, 32);
    padded = xmalloc(size);
    memset(padded, 0, size);
    strlcpy(padded, password, size);
    packet_put_string(padded, size);
```

```
    memset(padded, 0, size);
    xfree(padded);
}
```

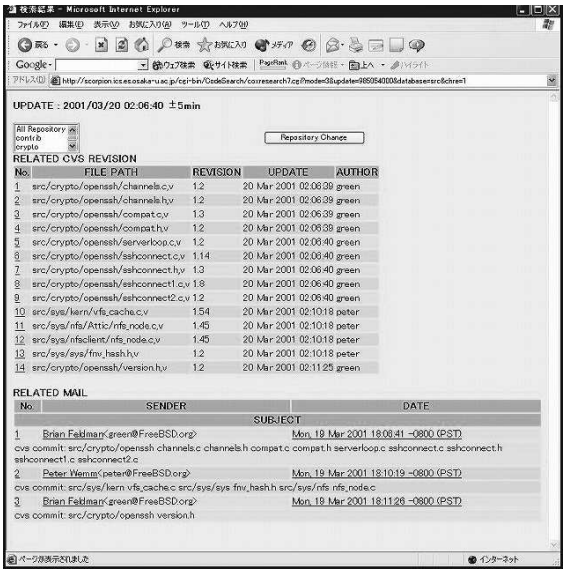

図 25: 更新日時間帯による検索

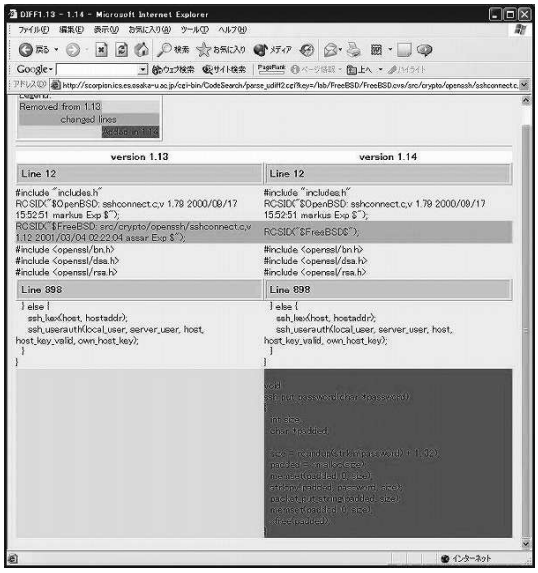

図 26: 関数 ssh_put_password の表示

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，類似コードによる検索を行なうことで修正部分の特定が可能となることが分かる．しかし，その情報だけでは完全に利用者の要求を満たすことはできなかった．そのため関連情報の中から必要となる情報を取得することができた．

6.4 検索速度評価

ここでは 3 つの類似検索データベースに対し，3 つのソースコードを与えて検索時間の計測を行なった．用意したデータベースは以下のものである．これらは FreeBSD の CVS リポジトリの src 以下のソースコードについての情報を取得したものである．これらのデータは表 2 に示す．

また，入力として与えたソースコード片 X・Y・Z に関するデータは表 5 に示す．特性キートークンを変えて計測する．

以上のデータを用いて類似コード検索の速度の検証を行なった. 単位時間は全て秒である．

CoDS では，入力として与える特性キートークンを出現頻度が少ないものを対象として用いると速くなることが分かっている [29]．本システムでは，特性キートークンを複数用いて類似比較をする範囲を絞り込んだ結果，さらに検索速度が上昇することが分かる．

	データ習得ファイル数	差分数	総ファイル数
src/crypto/openssh	67	6382	251
src/usr.bin	763	18403	827
src/contrib	318	31542	5395

表 2: データベースに用いたリポジトリに関するデータ

	行数	トークン数	特性キートークン
X	5	28	(if),(if,return),(if,return,fopen)
Y	25	134	(if),(if,return),(if,return,fopen)
Z	38	206	(if),(if,return),(if,return,fopen)

表 3: 与えた入力に関するデータ

6.5 考察

適用実験における考察

今回の適用実験から，利用者は手持ちのコードを用いることで類似した部分の情報を検索することができる．そのため，利用者は必要な情報を特定することが可能である．またその情報の関連情報を検索することにより，処理の意図が明確となり，大きなプロジェクトに対してその全体に対する理解を深めることができることが分かる．

情報検索の前提としてまず, 基本となる情報の特定が必要である. そのためには，上記の例のようにソースコードを用いたり複数のキーワードを用いることによって利用者自身が情報を絞り込む作業が必要である．しかし，適用実験や開発途中の利用例から情報を絞り込むことが可能になればそれに伴う関連情報を検索することは容易になる．また，コミットメールや同一更新者・同一時間帯に保存された開発履歴は関連性が強いことも確認できた．一方で解決に至っていない部分も存在する．たとえば，キーワード検索はデータ作成の際にコミットログやメールの内容に大きく影響を受けるので必ずしも有用な検索ができるとは限らない．また，表示された情報が全て利用者の求める情報であるとは限らないため，その情報に捕らわれて検索を繰り返していくうちに本来必要な情報を見落としてしまう恐れもある．

キーワードの問題は今後データ取得の方法を改善したり，あるキーワードを持つファイルに対して他の情報で関連性を持つと思われるファイルも同時に表示できるようにすることで改善できると考える．また，キーとして与える情報を複数にすることにより一度に表示する

	X	Y	Z
openssh	32	32	32
usr.bin	55	74	80
contrib	111	113	191

表 4: 検索の所要時間 (if)

	X	Y	Z
openssh	10	18	20
usr.bin	20	27	30
contrib	47	59	67

表 5: 検索の所要時間 (if,return)

	X	Y	Z
openssh	2	4	5
usr.bin	7	9	10
contrib	9	14	15

表 6: 検索の所要時間 (if,return,fopen)

情報の精度を上げることにより情報の見落としを防ぐことができるようになる．これらの点を改善できればさらなる開発支援が期待できる．

検索速度評価における考察

CoDS に対してさらに情報を絞り込むことにより，本システムでは検索速度が上がることを確認できた．しかし，この方法は始めに検索対象となる範囲を絞り込んでいるため，入力として与えるキートークンが間違っていれば，利用者が本当に必要としている情報を切り落としてしまう必要がある．今後，文字列一致の計算方法を改良することにより，情報全体の検索時間を速くすることによりさらなる改善が見込まれる．

7 まとめ

本研究ではオープンソースソフトウェア開発を研究対象として，現状のオープンソースソフトウェア開発における問題点を二つ指摘した．各システムが固有の形式で開発履歴を蓄積し，一定の方法でしか開発履歴を閲覧できない「システム固有の情報蓄積」の問題と，各システム間で連係を持たず，開発情報が分散して存在する「システム間の連係不足」の問題である．これらの問題は，開発者が過去に蓄積された有用なソフトウェアプロダクトを参照・利用することを困難にし，開発者に大きな不利益を与えるものだと考えられる．

そこでこれらの問題点を解決するために，数年間のオープンソースソフトウェア開発で蓄積された膨大なソフトウェアプロダクトの履歴を開発者が有効に利用できることを目的とした，オープンソースソフトウェア開発向けの開発支援環境を提案した．さらに本研究では，先に私の所属する研究グループでこれらの問題点を解決するために開発されたソースコード修正支援システム（CoDS）と，オープンソースソフトウェア開発支援のためのソースコード及びメールの履歴対応表示システム（SPxR）の統合を図り，新システムの実装を行なった．

本システムは「ソースコード修正支援部」と「開発履歴データベース生成部」と「データ検索・表示プログラム」の 3 つのサブシステムから構成されている．開発者はファイルを検索するためのキーワードを入力することにより該当するファイル情報を取得でき，さらに関連情報を検索することができる．本システムを用いることにより，従来では直接検索することが困難だった情報を提供することが可能となり，オープンソースソフトウェア開発における開発作業を支援することが可能になる．

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るデータを用いて本システムの実験を行なった．その結果，本システムは大規模なオープンソースソフトウェア開発に適用した場合でも実時間で動作し，実用的であることを確認した．さらに，本システムを用いることにより，開発者が必要とする情報を提供することが可能になることを確認した．

本システムの改善点としては実行速度の向上や関連情報の充実などが挙げられる．実行速度の向上については，検索手法のさらなる改善やデータベースを効率的に構築することが挙げられる．関連情報の充実については，関連性の有無だけではなく，情報のスコア付けを行なう手法を用いることや検索方法を増やすことなどが考えられる．これらの改善点を踏まえてオープンソースソフトウェア開発支援用の検索システムとしての充実を目指し，オープンソースソフトウェア開発環境支援に貢献することを目指す．

謝辞

本論文を作成するにあたり，常に適切な御指導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，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 助手に深く感謝致します.

本論文の作成において，適切な御助言を頂きました富士通株式会社 石川 武志 氏に深く感謝致します.

本論文の作成において，適切な御助言を頂きました松下電器産業株式会社 田原 靖太 氏に深く感謝致します.

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] 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.

[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] Annie Chen, Eric Chou, Joshua Wong, Andrew Y Yao, Qing Zhang, Shao Zhang, and Amir Michail, “CVSSearch:Searching through source code using CVS comments”, To appear in International Conference on Software Maintenance, 2001.

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

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

[7] CVSWeb, http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/.

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

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

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

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

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

[13] 藤原晃, “類似度を用いたプログラムの再利用性尺度の提案と実現”, 大阪大学大学院基礎工学研究科修士学位論文, 2002.

[14] Dan Gusfield, “Algorithms on Strings, Trees, and Sequences”, Cambridge University Press, 1997.

[15] 石川武志, 山本哲男, 松下誠, 井上克郎, “ソフトウェア開発時における版管理システムを利用したコミュニケーション支援システムの提案”, 情報処理学会研究報告, 2001-SE-133, Vol.2001, No.92, pp.23–30, 2001.

[16] 石川武志, “オープンソース開発支援のためのソースコードおよびメールの履歴対応表示システム”, 大阪大学基礎工学部情報科学科修士学位論文, 2002.

[17] 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

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

[19] 松下誠, 井上克郎, “自由な開発形態を支援するソフトウェア開発環境”, ソフトウェアシンポジウム 2000 論文集, pp.236–242, 2000.

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

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

[22] Namazu Project, 全文検索システム Namazu, http://www.namazu.org

[23] 大月美佳, “入門 CVS Concurrent Versions System”, SHUWA SYSTEM Co., Ltd, 2001

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

[25] 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.

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

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

[28] Temple Smith and Michael Waterman, “Identification of Common Molecular Subsequences”, J.Molecular Biology, 147, pp.195-197, 1981.

[29] 田原靖太, “既存ソフトウェアの変更履歴を利用したソースコード修正支援システム”, 大阪大学大学院基礎工学研究科修士学位論文, 2002.

[30] 寺口正義, 松下誠, 井上克郎, “バージョン間の差分を利用したデバグ手法の提案”, 電子情報通信学会技術研究報告, SS99-52, pp.17–24, 2000.

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

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

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

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

[35] 山本 哲男, 松下 誠, 井上 克郎, “バージョン管理ファイルシステムを用いた保守支援ツールの提案”, 電子情報通信学会技術研究報告 Vol.99, No.164, SS98-23, pp. 65–72, 1999.7.9.