修士学位論文

題目

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

指導教官

井上 克郎 教授

報告者

田原 靖太

平成 14 年 2 月 13 日

大阪大学 大学院基礎工学研究科

情報数理系専攻 ソフトウェア科学分野

平成 13 年度 修士学位論文

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

田原 靖太

内容梗概

近年のソフトウェア開発では，開発されたプロダクトを効率よく管理するために，版管理システムを利用することが多くなってきている．版管理システムは，プロダクトに対する追加，削除，修正等の作業履歴を蓄積・管理する．版管理システムに記録されている過去のプロダクトの開発履歴を閲覧することにより，過去の開発に関してより深く理解することができ，将来の開発作業に有用な情報を得ることができると考えられる．

しかし，蓄積された情報の中から，開発者が必要とする情報を選び出すことは容易ではない．例えば，ソフトウェア保守の段階において，プログラムに欠陥が発見されたとする．このとき，同様の欠陥に対する修正の履歴を既存のリポジトリから取得することができれば，現在の修正作業が行いやすくなる．しかし，実際にはどのリポジトリの，どのファイルの履歴を見ればよいかを特定することは困難である．また，リポジトリ内を検索する既存のシステムは，実際のソースコードに即した検索が行えないため，このような状況で修正作業に適用することは容易ではない．

本研究では，上記の問題点を解決するために，版管理システムに蓄積された履歴情報をソースコードの修正支援に役立てる手法 DIM の提案を行う．DIM は，蓄積された既存ソフトウェアのソースコード変更履歴を，ソースコード片を用いて検索し，結果を利用者に提示することにより，ソースコード修正作業の支援を行う．また，DIM に基づくソースコード修正支援システム CoDS の試作を行う．また，試作したシステム CoDS を用いた適用実験を行い，上で述べた問題点が解決されることを確認した．

主な用語

版管理 (Version Control)
ソースコード修正支援 (Supporting Source Code Modification)
ソースコード検索 (Source Code Searching)

目 次

1 はじめに **4**

2 版管理システム **6**

3 関連研究 **9**
3.1 版管理システムを用いた開発支援環境 9
3.2 履歴閲覧ツール 10

4 ソースコード修正支援手法 DIM **11**
4.1 データベースの構築 11
4.1.1 変更部分の抽出 12
4.1.2 字句解析 14
4.2 データベースの検索 15
4.2.1 2 つのソースコードの比較 16
4.2.2 アラインメント 16
4.2.3 類似しているかどうかの判定 20
4.3 検索結果の表示 21
4.3.1 結果の一覧表示 21
4.3.2 差分の表示 21

5 ソースコード修正支援システム CoDS **22**
5.1 データベース作成ツール 22
5.2 字句解析ツール 24
5.3 トークン比較ツール 24
5.4 CoDS-Main 24

6 適用実験 **30**
6.1 実験対象 30
6.2 検索の実例 30
6.3 結果の考察 32
6.4 検索速度 33
6.4.1 検索速度に影響を与えるもの 33
6.4.2 検索速度計測実験 34

7 あとがき **38**

謝辞 **39**

参考文献 **40**

1 はじめに

年々大規模化，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，近年のソフトウェア開発では，版管理システムと呼ばれるシステムを用いることが多くなっている．版管理システムは，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．また，リポジトリに格納された履歴を後に参照する機構を備えている．リポジトリに格納された履歴の中には，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情報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る [6]．

しかし，蓄積された情報の中から，開発者が必要とする情報を選び出すことは容易ではない．

例えば，ソフトウェア保守の段階において，プログラムに欠陥が発見されたとする．このとき，同様の欠陥に対する修正の履歴を既存のリポジトリから取得することができれば，現在の修正作業が行いやすくなる．しかし，実際にどのリポジトリの，どのファイルの履歴を見ればよいかを特定することは難しい．

一方，版管理システムに保存されているソースコードの中から，必要なものを検索するシステムとして，CVSSearch[4] が開発されている．CVSSearch は，キーワードによってリポジトリ格納時のコメントを検索することにより，関連するソースコードを取り出す手法を用いている．そのため，コメントを的確に記述していない履歴の中からは，目的のコードを取り出すことが困難となる．

本研究では，上記の問題点を解決するために，版管理システムに蓄積された履歴情報から目的の情報を検索することによるソースコード修正支援手法 DIM の提案を行う．また，それに基づくソースコード修正支援システム CoDS の試作を行う．DIM は，蓄積された既存ソフトウェアのソースコード変更履歴を，ソースコード片を用いて検索することにより，ソフトウェア保守における修正作業の支援を行う．具体的には，まず版管理システムに蓄積されているソースコードの変更履歴を検索用にデータベース化して保存しておく．次に，開発者の手元にあるソースコードからその一部分を入力として与え上記のデータベースを検索する．検索では，入力として与えたコードと類似した部分を含むソースコードをデータベースから取り出す．検索された変更履歴とそれに付随する情報を利用者に提示することにより，ソースコードの修正を支援する．

今回試作した CoDS は，CGI を使用しており，Web インタフェースにより利用することができる．CoDS は，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と,CGI 部である CoDS-Main から構成されている．字句解析ツール，トークン比較ツールは C 言語で，データベース作成ツール，CGI 部は Perl 言語で記述されており，全体で約 4000 行程度

となった．

また，試作したシステム CoDS を用いた適用実験を行った．その結果，リポジトリから同様の箇所の修正事例を取得することができ，上で述べた問題点が解決されていることを確認した．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 では準備として本研究に関連する分野について述べ，3 では従来の版管理システムを用いた開発支援環境について述べる．4 ではソースコード修正支援手法 DIM について述べ，5 ではソースコード修正支援システム CoDS について述べる．6 において本システムの適用実験について述べ，最後に 7 で本研究のまとめと今後の課題を述べる．

2　版管理システム

本節では，本論文を理解するために重要となる版管理システムについて説明する．
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 [1][18][13]．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．

The Checkout/Checkin Model

このモデルは，ファイルを単位としたリビジョン制御に関して定義されている (図 1 参照)．
リビジョン管理下にあるコンポーネントはシステムに依存したフォーマット形式のファイルとしてリポジトリに格納されている．開発者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

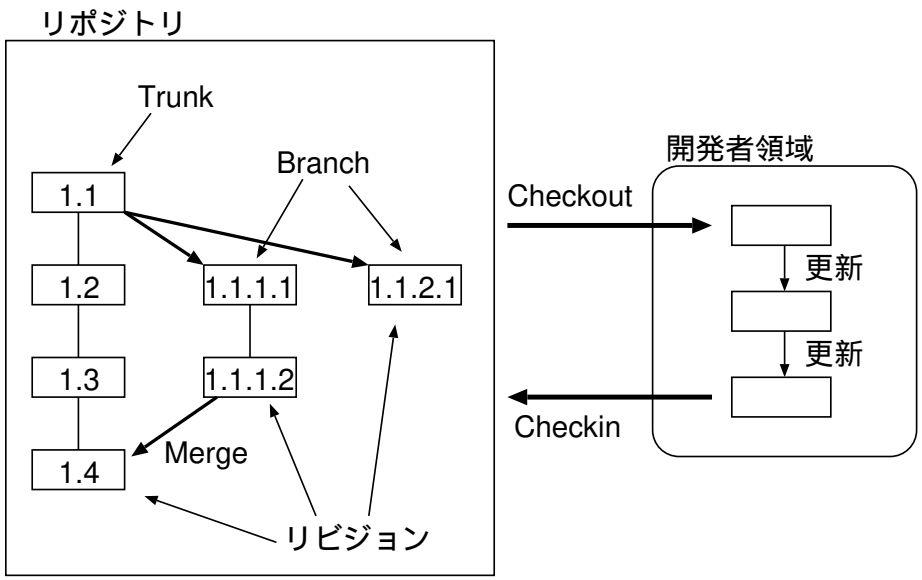


図 1: Checkout/Checkin Model

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[21] や CVS[2][8] といったシステムが標準で利用可能となっている．ClearCase[17] 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[16] や PVCS[15] 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 [9] 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

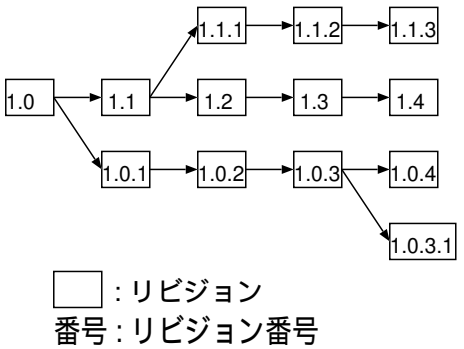


図 2: リビジョンツリーの例

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[21] 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[2][8][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．

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 [23]．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 [5] や ViewCVS [22] 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．

4　ソースコード修正支援手法 DIM

本研究では，版管理システムに蓄積されているソースコードの変更に関する情報を，ソースコード片を用いて検索して取り出すことを考える．そして，取り出された情報を開発者に提示することにより，現在作業中のソースコードの修正を支援する．

提案する手法 DIM(Delta Inexact Match) は，すでにリポジトリに蓄積されているデータを利用するため，DIM を用いるための特別なデータを蓄積する必要がないまた，検索のためのキーとして，ソースコード片をそのまま与えることができるため，検索キーワードを考える手間を省くことができる．

DIM におけるソースコード修正支援方法は以下のとおりである．

1. データベースの構築
   版管理システムのリポジトリから，必要な情報を自動的に抽出し，得られた情報をリポジトリとは別の検索用データベースとして保存する．検索用データベースには，直後のリビジョンで変更を受けたソースコード断片が格納されており，この部分が検索の対象となる．

2. データベースの検索
   上記のデータベースを，ソースコード片を利用して検索できるようにする．利用者は，同様の箇所の修正情報を参照したいソースコード片を入力として与える．与えられたソースコード片と，データベースの各レコードのソースコード片との比較は，トークン単位で行い，与えられたソースコード片に対応するトークン列と類似したトークン列を持ったレコードを検索結果として与える．

3. 検索結果の表示
   検索結果として得られたソースコード片を持つリビジョンと，直後のリビジョンとの差分を利用者に提示する．この時，入力と類似しているとみなされた部分が強調表示される．利用者は提示された差分とそれに関するコメントを参照することにより，現在のプログラムに対して行うべき修正を理解する．

以降の節では，上記の (1) ~ (3) について，順を追って説明する．

4.1　データベースの構築

本節では，版管理システムのリポジトリに格納された情報をもとに作成されるデータベースについて述べる．

リポジトリ内のあるソースファイル F において，隣接する 2 つのリビジョンの番号を p, q (ただし，q がより新しいリビジョン) とする．DIM では以下のデータを，"F の p から q までの変更情報 $D(F,p,q)$" とし，データベースの 1 レコードとして保存する．

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) $c(F,p,q)$ に関する情報
   検索時に，入力されたソースコード片との比較の対象となる部分である．この部分の詳細については後述する．

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) 内にあるすべてのソースファイルから収集する．

4.1.1 変更部分の抽出

ここでは，上の (5) について説明する．

ソースコード片を検索する際，利用者によって与えられた入力ソースコード片 I とデータベースのこの部分 c と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 (次節参照) し，トークン列に変換したもの t 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，変更があった行の前後の数行を含めた形で格納する．

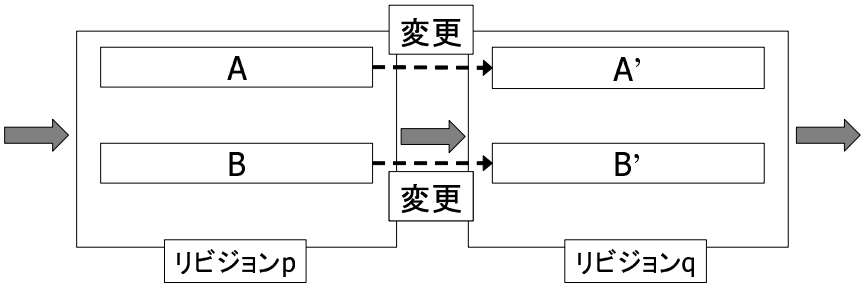


図 3: 変更部分コード

$c(F, p, q)$ の内容について，図 3 に模式的に示す．図 3 では，リビジョン p における A という部分がリビジョン q において A' に変更され，リビジョン p における B という部分がリビジョン q において B' に変更されたことを表している．この例では，

$$c(F, p, q) = \{A, B\}$$

となる．

$c(F, p, q)$ は，UNIX の context 形式の diff 出力を用いて取得する．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 4 に示す．この出力を解析することにより，リビジョン p に対して，次のリビジョン q で変更された部分のコードを，前後数行を含めて取得することができる．また，図のように context 形式の diff 出力では，各ブロックに対して，元のファイルにおける行番号が振られているので，各ブロックの開始行番号も情報として加え，後に参照しやすいようにする．

図のように，context diff 出力では，リビジョン p の内容とリビジョン q の内容が異なっている部分を，変更があった行だけではなく，その前後数行も含めてそれぞれ出力する．`***` で始まるブロックがリビジョン p，`---`で始まるブロックがリビジョン q の内容を表している．`***`，`---`の後に書かれている数値が，元のファイル内における開始行番号と終了行番号を示している．リビジョン p のブロックにおいて，行頭に'-' がついている行は，q において削除された行を表す．また，リビジョン q のブロックにおいて，行頭に'+' がついている行は，リビジョン q において追加された行を表す．'!' は，リビジョン p からリビジョン q で変更が加えられた行を表す．ここで，

1. リビジョン p からリビジョン q において，行の削除のみが行われた場合
2. リビジョン p からリビジョン q において，行の追加のみが行われた場合

```
*** 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 4: context diff の例 1

は，どちらか片方のコード部分は空白になる．特に，上の 2 の場合 (図 5) は，リビジョン p のブロックが空白になるので，リビジョン q のブロックの出力をもとにしてリビジョン p を生成しなければならない．それ以外の場合は，context diff 出力のリビジョン p のブロックを行頭の記号を取り除くだけで，$c(F, p, q)$ を得ることができる．

また，p と q の間で，コメントのみが変更されている場合は差分を検出しないようにするため，ソースコードからコメント部分を除去した状態で差分をとるようにする．このとき，元のソースコードと，コメントを取り除いたソースコードとの間で行の対応がとれるように，同じ行の行番号が変わらないようにする．

4.1.2 字句解析

利用者によって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，比較の前にあらかじめソースコード片を字句解析し，トークン列に変換しておく必要がある．

```
*** 51,56 ****
--- 51,57 ----
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
+ #include <unistd.h>
#ifdef SHELL #define main printfcmd
```

図 5: context diff の例 2

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生成されるプログラムの機能に影響しないので無視する．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（入力の先頭行を 1 とする）も付加する．

さらに，(5) 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．

4.2 データベースの検索

ここでは，入力として受け取ったソースコード片を用いて，データベース内のソースコード片を検索する手法について述べる．

すでに述べたように，DIM ではソースコードの比較はトークン単位で行う．比較をトークン単位とすることで，空白，改行やコメントを無視して比較を行うことができる．また，識別子や定数等の特定の種類のトークンを 1 つのトークンに固定することで，変数名や関数名の変更されたソースコード片も等価なものとして比較することができる．

4.2.1 2 つのソースコードの比較

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．

従来のソースコード比較では，一定数連続して一致する部分トークン列のみを一致トークンとして検出する，完全一致のアルゴリズムを採用している．このアルゴリズムは文字列の比較を最大で $O(n)$ で行うことができ，高速ではあるが，途中に異なるトークンが 1 つでも混入していれば，一致したとはみなされない．

DIM では，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 [10]" 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は，完全一致系列を見つける場合に比べて多くの計算量を要する．しかし，ソースコードの検索という用途の性質上，アラインメントを行う回数をある程度絞ることができると考え，比較のアルゴリズムとして採用した．

4.2.2 アラインメント

ここでは，文字列比較アルゴリズムのひとつであるアラインメントについて説明する．

文字列への文字の挿入，または文字列からの文字の削除のあった位置に - （ギャップと呼ぶ）を入れて，文字列の対応する位置を合わせる操作をアラインメント (alignment) と呼ぶ (アラインメントを行った結果得られたギャップ付の文字列の対をアラインメントと呼ぶこともある)．

アラインメントの例を以下に示す．次の例は文字列 S と T をアラインメントした結果が S', T' であることを示す．

```
S = abdc → S′ = abdc-
T = acca → T′ = ac-ca
```

一般に，2 つの文字列から構成可能なアラインメントの数は膨大なものとなる．そこで，各々のアラインメントごとに，それがどの程度よい物なのかをスコアで表す．アラインメントのスコア S_{align} は，以下の式を用いる．

$$S_{align} = f_1(n_{match}) - f_2(n_{mismatch}) - g(n_{gap})$$

ここで，n_{match},$n_{mismatch}$,n_{gap} はそれぞれアラインメントにより得られる，文字の一致および不一致 (置換) の個数，挿入されたギャップの個数である．上記の式では，文字の一致の

数が多いほどスコアが高くなり，文字の置換やギャップの個数が多いほど，スコアは低くなる．本稿では，以後，スコアが最大となるアラインメントの結果得られた文字列の対をアラインメントと呼ぶことにし，そのときのスコアを単にスコアと呼ぶ．

トークン列の比較においては，上記の 1 つの文字を 1 つのトークンにあてはめて考える．2 つのトークン列のアラインメントを求め，そのスコアによって，2 つのトークン列が類似しているかどうかを判定する．

DIM では，f_1, f_2 および g を，

$$f_1(x) = x$$

$$f_2(x) = x$$

$$g(x) = x$$

と定義する．

スコア最大のアラインメント

文字列 S の i 番目の文字を $S[i]$ ($1 \leq i \leq |S|$ $|S|$ は S の文字数) で表すことにする．$S[i]$ ($1 \leq i \leq |S|$) と $T[j]$ ($1 \leq j \leq |T|$) とのアラインメントのうち，最大のスコアをとるものを S_c とすると，S_c は次の式で表される．

$$S_c = \max \sum_{i=1}^{L} \sigma(S'[i], T'[i])$$

ここで S' , T' は , それぞれ S および T についてスコアを最大にするようなアラインメントをした結果であり , L はそのアラインメントの長さである . (すなわち , $|S'| = |T'| = L$) . また , $\sigma(a, b)$ は , 文字 a と b を照合したときのスコアであり , $\sigma(a, a)$ は文字の一致 , $\sigma(a, b)(a \neq b)$ は文字の置換が起こっている , $\sigma(a, -)$ または $\sigma(-, a)$ はギャップに対応する．

このようなスコア S_c を求めるには，次の漸化式を解けばよい．

$$\begin{aligned}
S_c[0,0] &\leftarrow 0, \\
S_c[i,0] &\leftarrow \textstyle\sum_{k=1}^{i} \sigma(S[k], -), (1 \leq i \leq |S|) \\
S_c[0,j] &\leftarrow \textstyle\sum_{k=1}^{j} \sigma(-, T[k]), (1 \leq j \leq |T|) \\
S_c[i,j] &\leftarrow \max\{S_c[i-1,j] + \sigma(S[i], -), \\
&\qquad S_c[i-1,j-1] + \sigma(S[i], T[j]), \\
&\qquad S_c[i,j-1] + \sigma(-, T[j])\}. \\
&\qquad (1 \leq i \leq |S|, 1 \leq j \leq |T|)
\end{aligned}$$

以下では，一致，置換およびギャップのスコアを定数としたときのアルゴリズムを示す．

- 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 $S_c[i, j]$
- $S_c[0, 0] \leftarrow 0$
- i を 1 から $|S|$ まで変化させながら
  - $S_c[i, 0] \leftarrow i \cdot (-\delta)$
- j を 1 から $|T|$ まで変化させながら
  - $S_c[0, j] \leftarrow j \cdot (-\delta)$
- i を 1 から $|S|$ まで変化させながら
  - j を 1 から $|T|$ まで変化させながら
    - $S_c \leftarrow \max\{S_c[i-1, j] - \delta, S_c[i-1, j-1] + \sigma(S[i], T[j]), S_c[i, j-1] + \delta\}$

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 [19]．

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

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，入力したトークン列の一部分と，データベース内にあるトークン列の一部分が，類似部分として抽出されることになる．しかし，すでに述べたように，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，以下のような制約を設ける．

- 利用者は，入力としたソースコードの中で特に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 1 つ指定することができる．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で最初に現れるキートークンを特性キートークンとする．
- データベース内のコードのうち，特性キートークンが含まれていないものは，入力コード片と類似しているとみなさない（キートークンの存在が前提となっているため，入力にキートークンを含んでいないものはエラーとする）．

このような制約を加えることで，アラインメントを求める回数を減らすことができ，検索効率の向上が見込まれる．すなわち，特性キートークンが，検索対象となるデータベース内のソースコード片に含まれているかどうかを調べ，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．

C 言語におけるキーワードを表 1 に，標準ライブラリを表 2 示す．

表 1: C 言語のキーワード

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

表 2: C 言語の標準ライブラリ

asser.h	float.h	math.h	stdarg.h	stdlib.h
ctype.h	limits.h	setjmp.h	stddef.h	string.h
errno.h	locale.h	signal.h	stdio.h	time.h

4.2.3 類似しているかどうかの判定

2 つのトークン列のアラインメントのスコアが一定値 α を超えるものを類似しているとみなす．スコアの定義により，アラインメントのスコアが入力されたトークン列の長さを超えることはなく，データベース内のトークン列が入力したトークン列と全く同一のトークン列を持つ場合スコアが最大となり，完全一致部分を求める操作に対応する．DIM では，完全一致は求めないので，入力したトークン列の長さに応じて α を決める必要がある．また，入力の全体と類似しているものだけでなく，入力の一部分と類似しているコードも検索したい．そこで入力トークンの長さを L_{input} として，$L \geq 30$ のとき $\alpha = 18$, $L < 30$ のとき $\alpha = 0.6 \cdot L$（整数部）と定義した．

4.3 検索結果の表示

データベースから，入力したソースコード片 I と類似した部分を持つレコードを検索した後，その結果を利用者に提示する．提示するのに必要な情報は，以下の 3 つである．

1. ファイル名とリビジョン番号の組 (F, p, q)
2. q のコミット日時とログメッセージ
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

検索によって上記のデータが得られたら，まず結果を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行単位で強調表示する．

4.3.1 結果の一覧表示

一般に，1 つのファイルに対して複数のリビジョンに関する変更情報が検索結果として返される．そこで，検索結果をまずファイル毎に分けて表示するようにする．その中のファイルを 1 つ選択することで，そのファイルに対する変更情報が一覧表示されるようにする．一覧表示では，ファイル名とリビジョン番号の組 (F, p, q)，q のコミット日時および q のコミットログを一覧で表示する．利用者は，表示された一覧の中からコミットログを参考にして，目的に合ったソースファイルを選択する．

4.3.2 差分の表示

ファイル名とリビジョン番号の組 (F, p, q) を選択することにより，F の p から q への差分を表示する．利用者が目的の部分を探しやすいようにするため，入力されたソースコード片と類似している部分を行単位で強調表示する．(3) では，類似部分のトークン列が，元のソースコード片の何行目にあったかが記述されているので，この情報をもとに行う．利用者が差分を参照することで，現在のソースコードにどのような修正を行えばよいかを知ることができる．

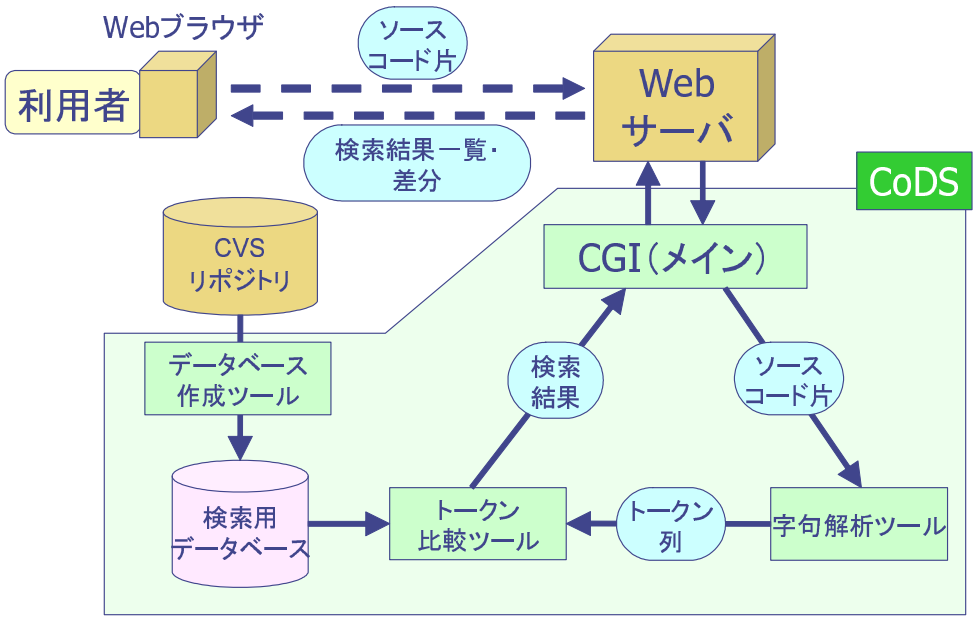


図 6: システム構成図

5　ソースコード修正支援システム CoDS

本節では，これまで述べた手法に基づくソースコード修正支援システム CoDS について述べる．CoDS 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．

CoDS は，以下のツールから構成される．

- CoDS-Main(CGI)
- 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

各ツール間の関係，データの流れを図 6 に示す．

5.1　データベース作成ツール

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を Perl 言語を用いて作成した．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として , GNU GDBM 1.8.0 を用いている . GDBM を用いると , Perl においてハッシュ (連想配列) とファイルを結びつけることができる . また , キーと値の長さに制限がない . ここでは , 本データベース作成ツールの詳細について説明する .

大まかな処理の流れは以下のようになっている .

- 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 , 以下の処理は行わず , 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 形式で次々ととっていく . 差分をとる前に , 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 .

以下 , 上の 1 ~ 6 各項目について説明する .

リビジョンツリーの取得

各リビジョンについて , そのリビジョンと隣接しているリビジョンの番号を取得しておき , 後のコード抽出の際に利用する . CVS リポジトリ内の履歴ファイル (元のファイル名に",v" を付け加えたもの) の記述をもとにして , あるファイル f がどのようなリビジョンツリーを持つのかを調べることができる . 履歴ファイルは , リポジトリに登録されているファイル毎に存在する . 履歴ファイルからは ,

- 現在の最新版のリビジョン番号
- 各リビジョンにおいて , 分岐しているブランチの最初のリビジョン番号と , 次のリビジョン番号

の情報を参照することができる．この情報を取り込むことにより，各リビジョンに関して，隣接するリビジョンを知ることができる．

コード部分の取得

ここでは，あるファイル f の，隣接する 2 つのリビジョン p から q において変更された部分と，その前後数行を context 形式の diff 出力を用いて取得する．この時，変更された部分としてコメントのみが抽出されるのを防ぐため，あらかじめコメント部分を取り除いた状態で比較を行う．このため，f のリビジョン p，q はそれぞれ先にチェックアウトしておき，コメント除去の処理を行った後，2 つのリビジョンで diff を実行する．コメント部分は削除されるが，改行文字は削除しないので，コメント除去処理の前後で，行番号は保存される．また，diff コマンド実行時は，-B オプションを付け，空行が入っているもしくは足りない等の違いは無視するようにする．

隣接リビジョン間について差分がとれたら，その出力からリビジョン p の部分のみを抜き出す．そして，抜き出した部分を次節で述べる字句解析ツールによってトークン列に変換する．

5.2 字句解析ツール

ソースコードをトークン列に変換するための字句解析ツールは，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，その中に現れる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，どの部分がマッチしているかを知ることができるように，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 C 言語を用いて実装を行った．

5.3 トークン比較ツール

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．トークン比較ツールは C 言語を用いて実装を行った．

5.4 CoDS-Main

利用者からソースコード片の入力を受け付け，入力されたソースコード片と類似する部分を含むコードをデータベース内から探索し，見つかったデータを一覧表示する．CoDS-Main での処理の概要を以下に示す．

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. T_I において，最初に現れるキートークン K_I を抽出する．キートークンがなければエラーとして終了する．
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．

CoDS-Main は，Perl 言語を用い，Web インターフェースで利用できるよう，CGI を用いている．CoDS-Main は，Web インターフェースで利用できるよう，CGI を用いている．本ツールを用いて，ソースコード片を検索した結果の一例を図 7 ~ 図 9 に示す．

図 7 は，検索結果をファイル毎にまとめたものの表示の画面，図 8 は，各ファイル別の変更情報表示の画面である．また，図 8 においてファイル名とリビジョンの組の部分を選択すると，そのファイルの当該リビジョン間の差分を図 9 のように表示する．その際，入力コードも参照することができる．図 10 のように，マッチ部分が強調表示されている．

図 7: 検索ファイル名一覧

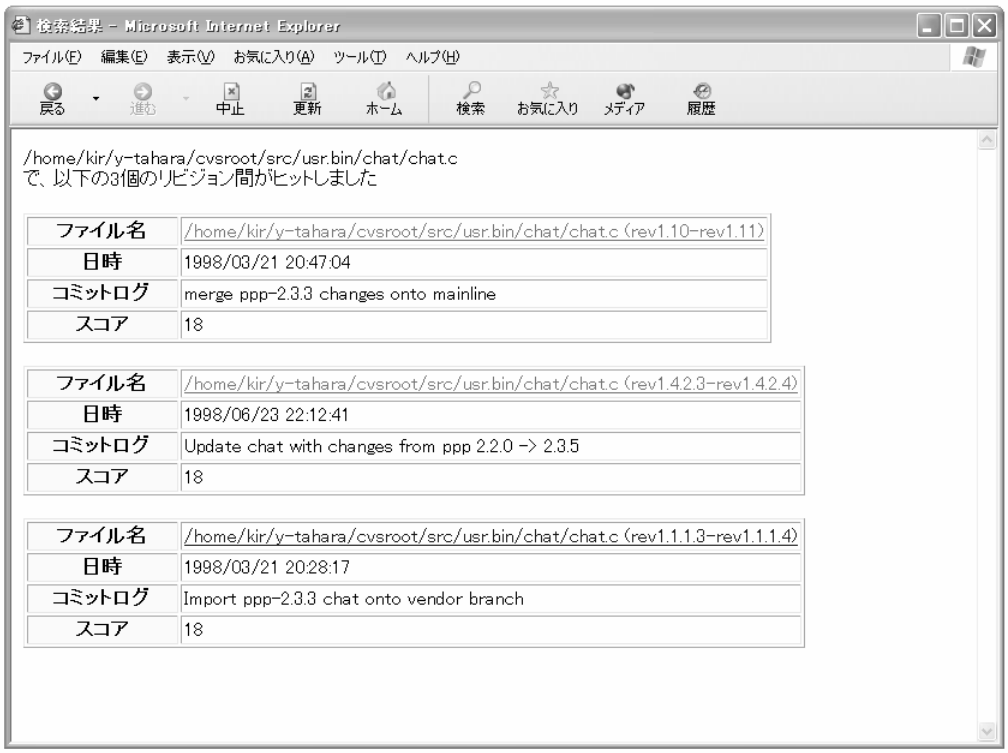


図 8: 検索結果のファイル別一覧

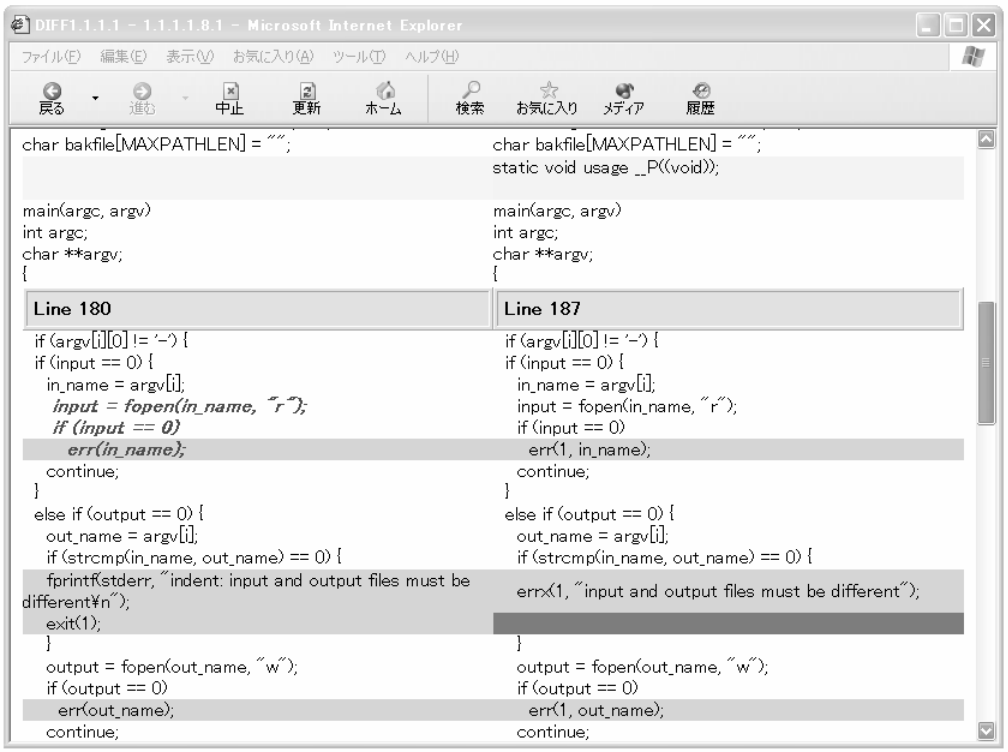

DIFF1.1.1.1 - 1.1.1.1.8.1 - Microsoft Internet Explorer
ファイル(F) 編集(E) 表示(V) お気に入り(A) ツール(T) ヘルプ(H)
戻る 進む 中止 更新 ホーム 検索 お気に入り メディア 履歴
char bakfile[MAXPATHLEN] = "";
main(argc, argv)
int argc;
char **argv;
{
Line 180
if (argv[i][0] != '-') {
if (input == 0) {
in_name = argv[i];
input = fopen(in_name, "r");
if (input == 0)
err(in_name);
continue;
}
else if (output == 0) {
out_name = argv[i];
if (strcmp(in_name, out_name) == 0) {
fprintf(stderr, "indent: input and output files must be different¥n");
exit(1);
}
output = fopen(out_name, "w");
if (output == 0)
err(out_name);
continue;
char bakfile[MAXPATHLEN] = "";
static void usage __P((void));
main(argc, argv)
int argc;
char **argv;
{
Line 187
if (argv[i][0] != '-') {
if (input == 0) {
in_name = argv[i];
input = fopen(in_name, "r");
if (input == 0)
err(1, in_name);
continue;
}
else if (output == 0) {
out_name = argv[i];
if (strcmp(in_name, out_name) == 0) {
errx(1, "input and output files must be different");
}
output = fopen(out_name, "w");
if (output == 0)
err(1, out_name);
continue;

図 9: 差分の表示

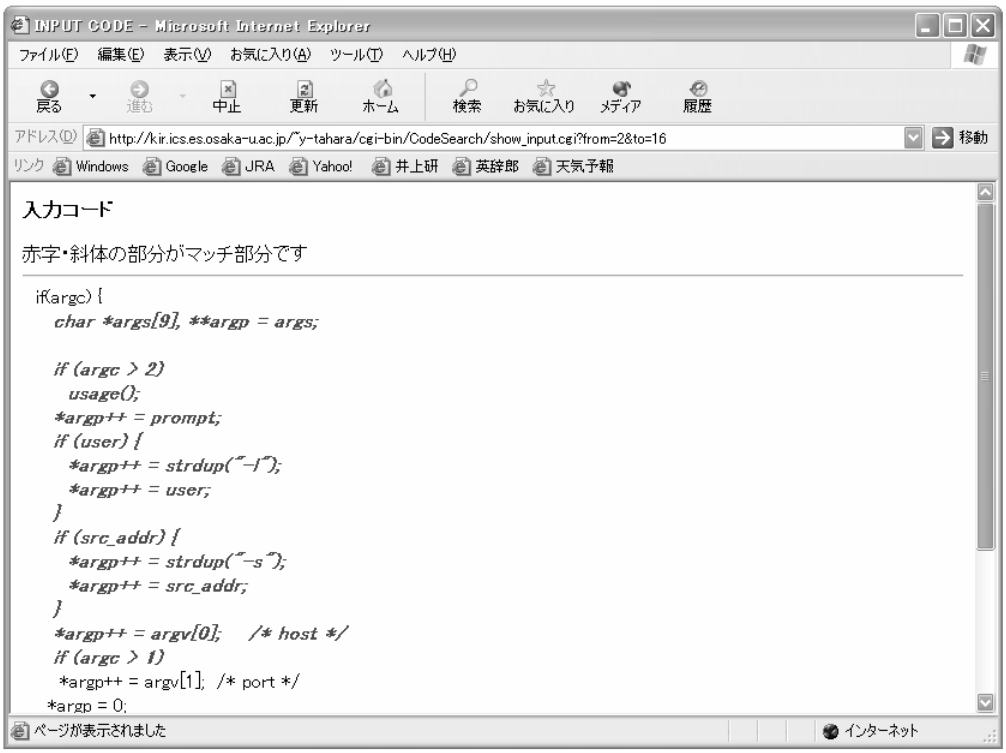


図 10: 入力コードの表示

6 適用実験

本節では，実際に入力を CoDS に与え，得られた検索結果について考察する．

6.1 実験対象

オープンソースソフトウェアの代表例である FreeBSD[20] のオープンソース開発環境から FreeBSD の CVS リポジトリをミラーリングして，そこから必要なソースファイルのリビジョン情報を取得する．

なお、実験環境は，以下の通りである．

- CPU : Pentium4 2.0GHz
- RAM : 1.0GB
- OS : FreeBSD 4.5-STABLE
- データベース : GNU GDBM
- WEB サーバ : Apache 1.3.22

6.2 検索の実例

CoDS に以下のコード片を入力として与えた時の出力結果の一つを示す．

入力コード：

```
*ptr = malloc(SIZE);
*if (ptr== NULL) {
*fprintf(stderr, "cannot allocate memory.");
*exit(1);
}
```

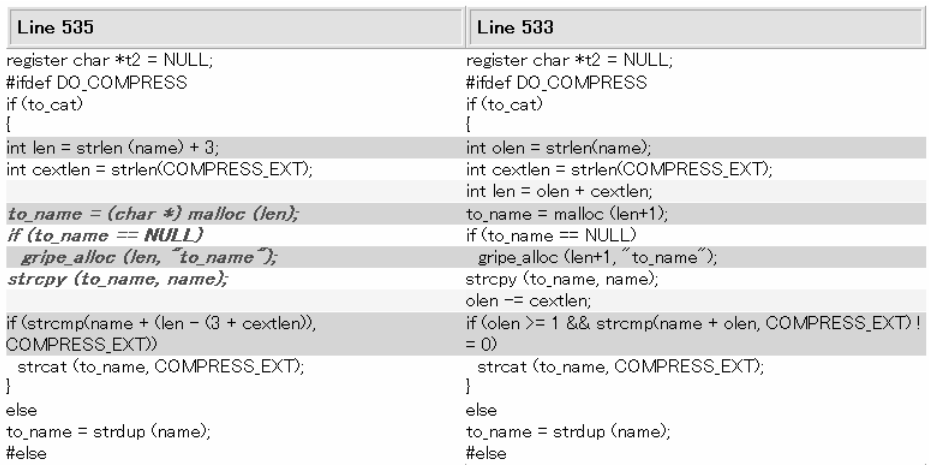
この入力コードは，メモリを確保し，確保に失敗したときはエラーメッセージを出力してプログラムを終了する機能を実装したものである．

この入力を与えたところ，検索結果の一つとして表 3 のような類似ソースコード片が検出された．さらに，検索されたリビジョンと，次のリビジョンとの間の差分は図 11 のように表示される．

表 3: 検索結果の例

ファイル名	/home/kir/y-tahara/cvsroot/src/gnu/usr.bin/man/man/man.c
リビジョン間	1.24-1.25
日時	1996/12/19 10:45:16
コミットログ	Even more buffer overflow fixes Change CATMODE to 0644, because group man not used Add immutable sbit to man binary, so if user even got man uid, he can't replace man binary with fake one Should go to 2.2 Submitted by: Marc Slemko <marcs@znep.com> with small editing by me

類似コードの例	if (to_cat)
	  {
	    int len = strlen (name) + 3;
	    int cextlen = strlen(COMPRESS_EXT);
	
	* to_name = (char *) malloc (len);
	* if (to_name == NULL)
	* gripe_alloc (len, "to_name");
	* strcpy (to_name, name);
	
	    if (strcmp(name + (len - (3 + cextlen)), COMPRESS_EXT))
	     strcat (to_name, COMPRESS_EXT);
	  }
	else



Line 535	Line 533
register char *t2 = NULL;	register char *t2 = NULL;
#ifdef DO_COMPRESS	#ifdef DO_COMPRESS
if (to_cat)	if (to_cat)
{	{
int len = strlen (name) + 3;	int olen = strlen(name);
int cextlen = strlen(COMPRESS_EXT);	int cextlen = strlen(COMPRESS_EXT);
	int len = olen + cextlen;
*to_name = (char *) malloc (len);*	to_name = malloc (len+1);
if (to_name == NULL)	if (to_name == NULL)
* gripe_alloc (len, "to_name");*	gripe_alloc (len+1, "to_name");
strcpy (to_name, name);	strcpy (to_name, name);
	olen -= cextlen;
if (strcmp(name + (len - (3 + cextlen)), COMPRESS_EXT))	if (olen >= 1 && strcmp(name + olen, COMPRESS_EXT) != 0)
strcat (to_name, COMPRESS_EXT);	strcat (to_name, COMPRESS_EXT);
}	}
else	else
to_name = strdup (name);	to_name = strdup (name);
#else	#else

図 11: 差分の表示

6.3 結果の考察

表 3 のコード部分において，行頭に*を付けた部分が，入力と類似している部分である．検索されたコードも，メモリの確保を行い，それに失敗した場合，`gripe_alloc(len, "to_name")` という関数を呼び出している．したがって，この検索結果は入力として与えたコードと同様の処理を行っている部分であるといえる．

図 11 を見ると，入力とマッチしたコード片は，次のような修正が行われていることがわかる．

変更前）`to_name = (char *) malloc (len  );`
変更後）`to_name =          malloc (len+1);`

変更前）`gripe_alloc (len  , "to_name");`
変更後）`gripe_alloc (len+1, "to_name");`

また，コミットログを参照すれば，この部分は，バッファのオーバフローの修正の一部であることがわかる．以上のように，ある入力を CoDS に与えた際，その入力と類似した部分に対する過去の修正事例を検索し，そこから有用な情報が得ることができることを示した．利用者は，この結果を見て，自分が入力としたコード片を持つプログラムに対して，同様のエラーが起こっていないかを確認し，提示された修正事例が適用できないかを判断する．

ところが，まだ解決に至っていない部分も存在する．修正の際には，入力と一致した部分だけでなく，同時に変更された部分に関しても参照する必要がある．現在のところ，入力と一致した部分の修正に関連する修正のみを抽出する機能は備えていない．これは，依存関係

解析等をこの段階で行うことにより，将来的に可能になると考える．また，該当する修正が，ほかのファイルにも及んでいる場合，それを同時に表示する機能も実装していないため，利用者はこの情報をもとにして，リポジトリ内部から同時にコミットされたであろうファイルを探索しなければならない．これらの点が改善されれば，さらなる修正支援が期待できる．

6.4 検索速度

検索の際には，利用者から与えられたソースコード片が，データベース内にあるすべてのソースコード片と比較されることになる．実際の運用においては，データベースにはできる限り多くのコードが蓄積されていたほうがよいが，データベースのサイズが大きくなればなるほど，検索にかかる時間は大きくなることは明らかである．そこで，実際にシステムを稼動させたときに，検索にどの程度時間がかかるのかを調べた．

6.4.1 検索速度に影響を与えるもの

検索時間に影響を与えると考えられる要素として，主に以下のものがあげられる．

1. 入力コード片のトークンの総数
2. データベース内にあるソースコード片のトークンの総数
3. 特性キートークン
4. データベースにおいて各コード片に出現するキートークン

1, 2 は，2 つの系列の長さの積に比例する時間計算量を要するアラインメントの性質から自明である．ここでは，3 および 4 について具体的な考察を加える．

すでに述べたように，DIM では，特性キートークンによってデータベース内のコードをふるいにかける．データベース内にある各コード片 (トークン列) には，そこに現れるキートークンのリストが情報として付随している．比較を行う際に，アラインメントをとる前段階として，このリストの中に特性キートークンが存在するかどうかを探索する．したがって，特性キートークンが，データベース内の多くのコード片に含まれているものである場合には，アラインメントを行う回数が増加し，検索時間がかかることになる．C 言語において，比較的よく現れるキートークンとしては，if がある．表 4 に，上の例で用いたデータベース内の各コード片が持つキートークンを集計し，キートークン別に出現コード片数が最も多いものから順に並べたものである．

表 4 のように，”if” を含んでいるコード片が最も多く，次いで”else”, ”return”, ”while”, ”for” と続く．”if” と”else” の多さが目立つが，それ以下の順位のトークンについては，全体の 30%程度であり，アラインメント回数を相当減らすことができている．

表 4: キートークン別出現コード片数

if	15512
else	7633
return	6387
while	5541
for	5127
break	5023
全コード片数	21453

6.4.2 検索速度計測実験

ここでは，3 個の入力コード片と 3 個のデータベースを用意し，検索時間の計測を行った．用意したデータベース A, B, C に関するデータを表 5 に示す．

入力として用いたソースコード片は以下のものである．このソースコード片のトークン数は 27 である．また，データベースは FreeBSD の CVS リポジトリの src/usr.bin 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．これについてのデータを表 5 に示す．

表 5: データベースに用いたリポジトリに関するデータ

	データ取得ファイル数	レコード数	格納トークン総数
A	151	1478	約 66 万 9000
B	733	7466	約 158 万 2000
C	371	3220	約 321 万 1000

また，入力として用いたソースコード片 X, Y, Z に関するデータを表 6 に示す．

表 6: 与えた入力に関するデータ

	行数	トークン数	特性キートークン
X	3	22	if
Y	14	78	if
Z	53	290	if

ここでは，X ~ Z の特性キートークンはすべて”if” に統一し，検索時間に影響するトーク

ン長以外の要素を排除している．以上のような入力とデータベースに対して，CoDS が入力をコードを受け取ってから，結果を送り出すまでの所要時間を計測した．その結果を表 7 に示す．また，この表をグラフで表したものを図 12 に示す．

表 7: 検索の所要時間 (単位：秒)

	X	Y	Z
A	9	13	23
B	29	36	59
C	52	66	105

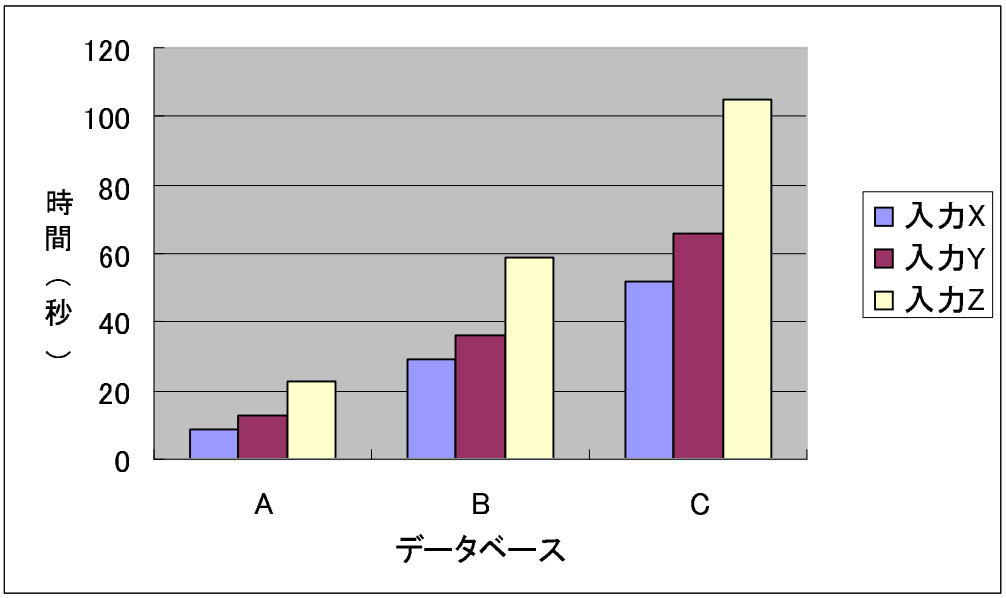


図 12: 検索の所要時間

入力の特性キートークンが”if” である場合，検索時間がかなり大きくなると考えられるが，この場合でも 2 分以内に結果が出力されている．表 4 を見ても，特性キートークンが”if” や”else” でなければ，検索時間はさらに小さくなることが期待できる．

そこで，今度は特性キートークンが”for” であるような入力 P, Q, R を用意して，検索時間の測定を行った．トークン数は入力 X ~ Z と等しい (表 8)．

このような入力に対して，表 9，図 13 のような結果が得られた．同じトークン数でも，特性キートークンが”if” であるときに比べて検索時間が小さくなっていることがわかる．

ただし，入力のサイズがさらに大きくなった場合はさらに検索時間が大きくなることが予

表 8: 特性キートークンが for の場合

	行数	トークン数	特性キートークン
P	3	22	for
Q	14	78	for
R	53	290	for

表 9: 特性キートークンが for の場合の検索の所要時間 (単位：秒)

	P	Q	R
A	5	6	13
B	14	15	24
C	24	26	44

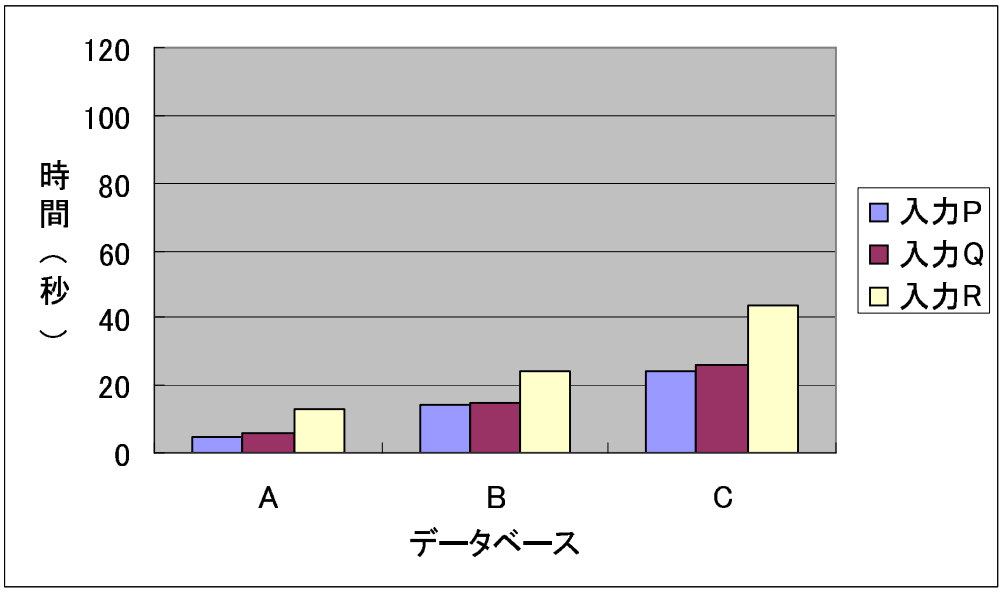


図 13: 特性キートークンが for の場合の検索の所要時間

想されるので，さらに検索速度を向上させるための手法が必要であると考えている．

7　あとがき

本研究では，過去のソフトウェア開発履歴から，必要なソースコード修正の情報を取り出すことにより，現在作業中のソースコード修正を支援する手法 DIM を提案した．DIM は，版管理システムのリポジトリからソースコードの変更履歴を取り出してデータベース化する．そのデータベースをソースコード片を入力として検索し，その結果を利用者に提示する．また，DIM に基づくソースコード修正支援システム CoDS の試作を行い，適用実験を行った．その結果，利用者が目的としているソースコード修正情報が検索され，かつ実用的なサイズのデータベースに対して実用時間で検索を行えることがわかった．今後の課題としては，より妥当な検索結果を得るために，スコアの計算方法を改良することや，実際の保守作業に適用することにより，DIM の有効性を評価することなどがあげられる．

謝辞

本論文を作成するにあたり，常に適切な御指導を賜わりました大阪大学大学院基礎工学研究科情報数理系専攻 井上 克郎 教授に心より深く感謝致します．

本論文の作成において，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基礎工学研究科情報数理系専攻 楠本 真二 助教授に深く感謝致します．

本論文の作成において，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基礎工学研究科情報数理系専攻 松下 誠 助手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基礎工学研究科情報数理系専攻井上研究室の皆様に深く感謝致します．

参考文献

[1] 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.

[2] Brian Berliner, “CVS II:Parallelizing Software Development”, In USENIX, Washinton D.C., 1990.

[3] CASE 1988-89, “Sentry Market Research”, Westborough, MA, pp.13-14, 1989.

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

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

[7] FIPS Publication 106, “Guideline on Software Maintenance”, Federal Information Processing Standards, U.S.Department of Commerce/National Bureau of Standards, June 1984.

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

[9] Peter Fröhlich and Woflgang Nejdl, “WebRC Configuration Management for a Cooperation Tool”, SCM-7, LNCS 1235, pp175–185, 1997.

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

[11] 古賀健太郎，飯田元，松本健一，井上克郎,“ソフトウェアコンポーネント利用情報の収集と共有”，電子情報通信学会技術研究報告, ソフトウェアサイエンス研究会, Vol. 100, No. 472, SS2000-27, pp.1-8, Nov. 28, 2000.

[12] 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.

[13] Jyhjong Lin and Chunshou Yeh, “An Object-Oriented Formal Model for Software Project Management”, Proceedings of APSEC, pp.552–559, December, 1999.

[14] Cashman, Paul M., and Anatol w Bolt : “A Communication-Oriented Approach to Structuring the Software Maintenance Environment”, Software Engineering Notes, 5, No.1, pp.4-17, January 1980.

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

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

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

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

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

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

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

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

[23] 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.