ソフトウェア工学　136－8
（2002．3．7）

# 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手法の提案

田原靖太[†]　松下誠[†]　井上克郎[†]

[†] 大阪大学大学院基礎工学研究科
〒 560-8531 大阪府豊中市待兼山町 1-3

近年，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い理解を得ることができると考えられる．しかし，過去の開発履歴を閲覧するための支援環境は十分とはいえない．そこで本論文では，版管理システム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して，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た結果を用いたソースコード修正支援手法を提案する．本手法によって，ソフトウェア保守において過去の開発履歴を容易に参照することができ，修正作業が行いやすくなることが期待できる．

# Supporting Method for Source Code Modification with the Changes of Existing Software

Yasutaka Tahara[†], Makoto Matsushita[†] and Katsuro Inoue[†]

[†] Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka,
Osaka 560-8531, Japan

Recently, revision management system is often employed by software development environment, for efficient management of product. We suppose that we'll get deeper understanding of past developments by refering to histories of them in revision management system. However the support environment is insufficient to do that. In this paper, we propose a method of supporting souce code modification. In this method, we build a database of revision histories of source codes, and show the results of searching the database by a source code fragment. With this method, developers can refer to histories of past developments easily in software maintenance, and source code modification gets easier.

## 1　まえがき

近年，大規模化，複雑化してきたソフトウェアシステムへの要求に対応するために，ソフトウェアの高生産性，高信頼性が要求されるようになっている．それらの要求を解決することを目的として，ソフトウェアの再利用が注目を集めている．

また，近年のソフトウェア開発では，版管理システムを用いる傾向がある．版管理システムでは，プロダクトの開発履歴を管理することができ，その中に，将来の開発に生かす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 [4].

現在，版管理システムに保存されているプロダクトの変更履歴の閲覧を支援するツールが開発，利用されている [3][11]．しかし，現在のプロジェクトの保守活動にあたり，これらのツールを用いて過去のプロジェクトから同じような処理を行うソースコードの一部分を見つけ出し，その部分の履歴を参照するのは困難である．また，現状では，開発者は過去のプロジェクトの履歴を参考にすることなく，同じような処理を行うコードを各開発者が独立に何度も書くということが多い．そしてその部分に同じような欠陥を作りこんでしまうことが多く発生すると考えられる．

そこで本研究では，ソースコードの変更履歴の中から，開発者が必要としているものを検索できるようにし，その結果を提示することにより，ソースコードの修正を支援する手法を提案する．本手法では，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，そのコードと類似した部分が過去にどのように修正されたのかをデータベースから検索できるようにする．本手法を用いることで，現在作業中のソースコードを用いて，過去の開発の履歴を参照し，作業中のソースコードの修正を支援することができる．このことにより，ソフトウェア生産性の向上が期待できる．

## 2　版管理システムと関連研究

本節では，版管理システムについての簡単な説明を行った後，版管理システムの履歴を閲覧できるツールと，版管理システムを用いたソフトウェア開発支援について述べる．

### 2.1　版管理システム

版管理システムとは，ソフトウェア開発の際に，その開発過程を履歴として管理するシステムであり，開発プロセスの作成や再利用の際に役立つ [4].

版管理システムは，リポジトリに格納されたファイルをリビジョン単位で管理する．このため，内部的にリビジョン情報を作成し，保持している．リビジョン情報として扱われる情報は，リビジョン番号，リビジョン間の差分情報，更新日時，更新者，コメント等である．

版管理システムの実装は数多く存在し，また，実際に利用されている．多くの場合，CVS や RCS[10] 等のシステムが利用されている．また，ClearCase[9]，Visual SourceSafe[8] や PVCS[7] 等，商用の版管理システムも利用されている．

### 2.2　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 [3] や ViewCVS [11] といったツールが開発されている．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の表示，リビジョン間の差分の色付表示ができる等の機能を持つ．しかし，新たなプロジェクトの開発・保守作業において，リポジトリの内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．

### 2.3　版管理システムを用いた開発支援環境

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 [12]．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．しかし，ソースコードそのものを取り出す機能は備えていない．

また，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 [2] 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 ソースコード修正支援手法

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，本手法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，手元にある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．

本手法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，得られた情報をデータベースとして保存する．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．

### 3.1 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．本手法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**
ログメッセージは，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る．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．ログメッセージ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** $q$ **までに変更された行（前後の数行を含む）** $c(F,p,q)$ **をトークン列** $t(F,p,q)$ **に変換したもの**
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．

#### 3.1.1 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．

$c(F,p,q)$ は，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．context 形式の diff 出力の一部を図 1 に示す．このように，変更が行われた行を含む部分をまとめて，変更前の状態と変更後の状態の両方が出力される．一般には，

```
*** 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 1: context diff の例

この出力を用いて，$c(F,p,q)$ を取得する．図 1 の例では，上半分のコードが $c(F,p,q)$ に対応する．

また，図 1 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始点行番号および終点行番号が出力されるので，各ブロックの開始行番号も情報として加える．これは，入力と類似した部分の強調表示の際に利用する．

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（このとき，元のソースコードとコメントを除去したソースコードとの間で行の対応がとれるように，コメント除去の際に改行文字は削除しない）．

#### 3.1.2 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．

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（標準ライブラリ関数名，標準型名についてはそれぞれを別のトークンに変換する）．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．キートークンの具体的内容と，使用法の詳細については後述する．

### 3.2 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，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．

#### 3.2.1 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．

本手法では，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 [6]” 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．

**アラインメント**

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）．

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．本手法におけるアラインメントのスコア $S_{align}$ を，以下のように定める．

$$S_{align} = n_{match} - n_{mismatch} - n_{gap}$$

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．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．

**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 $T$ との局所アラインメント $S', T'$ を上で定義したスコアに基づいて求めると，以下のようになる．

```
S' = bcbccb
T' = bcdccb
スコア = 5 - 1 - 0 = 4
```

このアルゴリズムの時間計算量は 2 文字列の長さの積に比例する．

**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．

- プログラミング言語における予約語および言語仕様やライブラリ等で定義されている手続き・関数等の名前を「キートークン」とする．
- 利用者から入力されたソースコード片に最初に現れるキートークンが含まれていないソースコード片は，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，利用者から入力されたソースコード片に最初に現れるキートークンが，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．

#### 3.2.2 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 $\alpha$ 以上の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い．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，入力トークン全体との完全一致部分を求める操作に対応する．本手法では，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 $\alpha$ 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列の長さを $L$ として，$L > 30$ のとき $\alpha = 19$ , $L \leq 30$ のとき $\alpha = 0.6 \cdot L$（整数部）と定義した．

### 3.3 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．

#### 3.3.1 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．

#### 3.3.2 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．

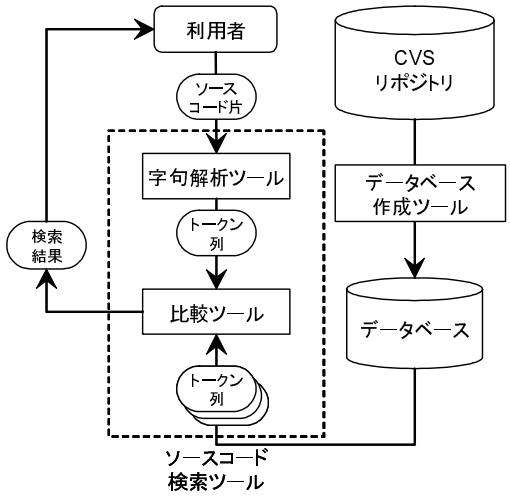


図 2: システム構成図

## 4 ソースコード修正支援システム

我々は，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の試作を行っている．本システムは，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．

本システムは，以下のツールから構成される．

- 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
- ソースコード検索ツール

各ツール間の関係，データの流れを図 2 に示す．

### 4.1 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 言語を用いて作成する．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 においてデータベースを扱うルーチンとして，GDBM を用いている．GDBM は，Perl のハッシュ（連想配列）とファイルを結びつけるものであり，キーと値の長さに制限がない．このデータベース作成ツールを用いて，CVS リポジトリ内にあるソースコードから，データベースを作成しておく．また，内部で次に述べる字句解析ツールを用いている．

### 4.2 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．

### 4.3 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 $S, T$ 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．

### 4.4 ソースコード検索ツール

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検索し，結果を一覧表示する．本ツール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 $\alpha$ 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．

本ツールは，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 3〜図 5 に示す．

図 3 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 4 は，各ファイル別の変更情報表示の画面である．また，図 4 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 5 のように表示する．

## 5 適用例

本節では，実際にどのような入力に対してどのような類似ソースコード片が検索されるかの例を

図 3: 検索ファイル名一覧

図 4: 検索結果のファイル別一覧

示す．C 言語で書かれた以下のようなソースコード片の入力に対して，

入力コード：

```
*   fp = fopen("file1.c", "r");
*   if ( fp == NULL ) {
*     perror("error.");
*     return(-1);
    }
```

例えば次のような類似ソースコード片が検出される．

類似コードの例：

```
    crash();
    }
*   fout = fopen(outfile, "w");
*   if (fout == NULL) {
*     f_print(stderr, "%s: unable
                 to open ", cmdname);
*     perror(outfile);
      crash();
    }
    record_open(outfile);
```

行頭に*を付けた部分が，類似している部分である．上のコードをそれぞれトークン列に変換すると，次のようになる（実際にはトークンは整数値として出力される）．

図 5: 差分の表示

入力コード（トークン変換後）：

```
*   i = fopen(i, i);
*   if ( i == i ) {
*     perror(i);
*     return(i);
    }
```

類似コードの例（トークン変換後）：

```
    i();
    }
*   i = fopen(i, i);
*   if (i == i) {
*     i(i, i, i);
*     perror(i);
      i();
    }
    i(i);
```

このように，標準ライブラリ関数名である fopen, perror，予約語である if, return と，演算子等の記号は区別されたトークンに変換され，その他の識別子，定数（文字列定数含む）は，すべて等価なトークン（上記の例では i と表記した）に変換される．このような変換を行った上で，2 つのトークン列のアラインメントを行うと，上記の*の部分が類似したトークン列として検出される．入力コードにおけるトークン数は 27 であるから，類似しているとみなされるスコアの閾値は 16 である．この例における 2 トークン列の局所アラインメントのスコアは 18 であるから，2 つのトークン列は類似しているとみなされ，検索結果の 1 つとして出力される．

利用者は，このようにして検索された部分が次のリビジョンでどのように変更されたのかを参照する．これにより，入力として与えたソースコード片に対してどのような修正を行えば良いのかを知ることができる．

## 6 まとめ

本論文では，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を提案した．本手法は，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．

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，実際の開発作業に適用することによる本手法の有効性の評価を行うことがあげられる．

## 参考文献


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

[2] A.Chen, E.Chou, J.Wong, A.Y.Yao, Q.Zhang, S.Zhang, and A.Michail, “CVSSearch:Searching through source code using CVS comments”, To appear in International Conference on Software Maintenance, 2001.

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

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

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

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

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

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

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

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

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

[12] 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.