単純さと高機能性を両立した小規模バージョン管理ツール

三上 裕明　　坂本 大介　　五十嵐 健夫 *

概要. プログラミングにおいて，ソースコードを過去の状態に戻す作業はしばしば必要となる．しかし，トライアルアンドエラーのような細かい調整を行うような場面において，既存のバージョン管理システムはあまり使いやすくはない．本論文では，まずこのような小規模バージョン操作を行う際の問題点を把握するためのインタビュー調査を行った．この結果，1) ユーザの記憶に頼らずバージョン管理操作を行える事，2) 様々なバージョン管理操作を実行できる事，3) 素早く操作できる簡単なインタフェースを持つ事の 3 つを満たすツールが望ましいという結果を得た．本研究においては，この結果を反映した小規模バージョン管理のためのインタフェースを提案する．このインタフェースではバージョン管理操作の内容をテキストエディタ上にパネルとして表示する．このパネルをクリックすることでユーザは操作を実行する事ができる．また，ユーザが実行する可能性の高い操作を推測し，それをテキストエディタ上に表示することで，様々なバージョン管理操作の実行を可能にする．

1　はじめに

ソフトウェア開発においては，バージョン管理システム（Version Control System; VCS）が広く用いられている．過去のソースコードの情報を保存し，問題発生時にその情報を利用するため，あるいは多人数での開発を効率的に管理するために，Git[1] などのバージョン管理システムが利用されている．

バージョン管理システムは大規模なソフトウェア開発で利用されるものであると思われがちだが，一方，小規模なプログラミングの最中でも，バージョン管理操作をし，テキストを過去の状態に戻す操作がしばしば必要になる [10]．例えば，パラメタを変更した後，プログラムの動作が不安定になったので元に戻す操作や，既存のプログラムを新しい関数を利用するコードに書きなおした所，上手く動作しなかったので，元のプログラムに戻す操作などが考えられる．このようなバージョン管理操作は，Git など既存の VCS が想定する操作よりも規模が小さく，小規模バージョン管理と呼ぶ事ができる．これらの操作は，過去の研究における *backtrack*[10][8] や，*exploratory programming* [5][4][9] と呼ばれているものとほぼ同じである．

このような小規模バージョン管理のためのツールはいくつか提案，利用されている．しかし，現在のツールには，単純なインタフェースと十分な機能の両立ができていないという問題点がある．多くのプログラマは，線形 undo を小規模バージョン管理のために用いている [8]．線形 undo はほとんどのテキストエディタに実装されている機能であり，直近の編集を取り消す事ができる（Windows であれば Ctrl+Z，Mac OS であれば Command+Z のショートカットでよく知られる）．一方で，機能は限定されており，直近の編集しか取り消すことができない．そのため，取り消したい編集と現在のテキストの間に何らかのテキスト編集が存在すると，取り消すことができない．

線形 undo の問題点を解決するために，選択的 undo[9]，木構造を持った undo[12] が提案されている．また，既存の VCS も使用することが可能である．これらのツールは小規模バージョン管理の典型的な操作を行うのに十分な機能を備えている．しかし，これらのインタフェースは複雑である．例えば，典型的な選択的 undo や VCS では，特定の操作を取り消すために，1）履歴画面を開き，2）取り消したい編集を履歴画面から探し，3）操作の取り消しを実行する必要がある．既存研究では，プログラマの多くはダイアログやメニューを用いる複雑なインタフェースよりも，キーボードショートカットやマウスクリックだけの単純なインタフェース（Eclipse の Quick Fix など）を好む傾向がある事が知られている [7]．このため，現在の典型的ツールはプログラマには好まれないインタフェースを利用しているといえる．

上記の問題点から，3 割近くの小規模バージョン管理操作が，プログラマ自身がコードを直接書き直すことで行われている [8]．しかし，自分でコードを書き直す事は，不適切な編集によるバグや，テキストの消し忘れ，戻し忘れといった問題を引き起こす危険性がある [10]．したがって，小規模バージョン管理操作を確実に，効率的に行うためには，単純なインタフェースと十分な機能を両立したツールが必要である．本研究では，まずプログラマの小規模バージョン管理の方法や，その問題点をインタビューに

* 東京大学大学院

[1] https://git-scm.com/

よって調査した．そして，その結果に基づき，新しいユーザインタフェースを提案する．

このインタフェースは，図 1 のように，プログラマが行う可能性のあるバージョン管理操作を示すパネルを，テキストエディタ上に直接表示する．そして，各パネルをクリックすると対応するバージョン管理操作が行われる．また，エディタ上のパネル数を抑えるために，過去のバージョン情報から，プログラマが行う可能性の高い操作を推測し，可能性の高いものだけを表示する．

本論文の主たる貢献は，以下の通りである．

- プログラマへのインタビュー調査により得られた小規模バージョン管理用システムに必要な要件についての知見
- バージョン管理操作を示すパネルを用いる新しいインターフェースの提案，実装
- 提案インターフェースのための，プログラムの実行に着目したバージョン管理手法の実装

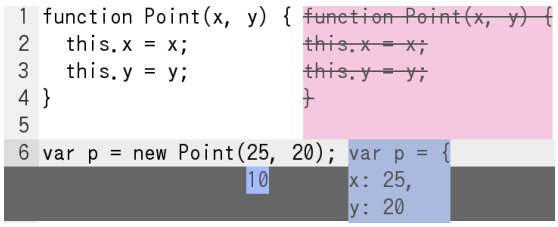

図 1. 提案システムにおけるテキストエディタ画面．エディタ上のパネルをクリックする事で小規模バージョン管理を行うことができる．

2 関連研究

2.1 undo システムの拡張

undo システムは広く使われているツールであるが問題点も多いため，様々な拡張が提案されている．Yoon らは，より使いやすい選択的 undo の実現のために，新たな選択的 undo アルゴリズムと，テキスト編集内容の可視化手法を提案している [9][11]．

Vim の Gundo[2] などは，テキスト編集情報を木構造で管理することによって，テキスト編集中の任意の時点に戻ることを可能にしている．大江らは，木構造を持った undo システムに，テキストの内容による検索機能を追加することを提案している [12]．また，現在広く利用されているテキストエディタの 1 つである emacs[3] は，選択した範囲内に適用された最新のテキスト編集を取り消す，範囲内 undo（undo-in-region）を実装している．

木構造を利用した undo や範囲内 undo は，テキストの編集過程によっては undo できない編集が存在する．すなわち木構造を利用した undo は選択的 undo を行う事ができず，範囲内 undo は一度 undo した編集を元に戻す事が難しい．例えば，あるパラメタの変更を範囲内 undo して新しいパラメタを書くと，範囲内 undo を用いて元のパラメタに戻すことはできなくなる．そのため，本研究で目指す小規模バージョン管理に常に利用できるとは限らない．

また，選択的 undo や木構造を利用した undo は，undo の拡張として重要であるが，線形 undo の制限の解決に主眼が置かれており，操作性の改善はほぼ行われていない．例えばこれらのツールでは，編集履歴一覧から undo する編集を選択するというような複雑なインタフェースが利用されている．本研究で提案するシステムは，プログラマが使いやすいインタフェースというユーザビリティに注目している点でこれらの研究とは異なる．

2.2 IDE におけるバージョン管理システムの利用

バージョン管理システムと IDE を組み合わせることにより，ユーザのバージョン管理操作を支援する研究も行われている．Seinert らは，IDE から素早く過去のバージョンのプログラムの実行時情報を利用できるべきだと主張した [6]．そして，この目的のために，バージョン間での AST の違いや，テストの成功率などを利用してバージョン情報をユーザに提示する新しい IDE の拡張を提案した．

Lee らは，現在の IDE は，コードの履歴情報の管理とコード編集の機能が分離されていることが問題であると主張した [3]．そして，Tempura という，コード補完機能に，バージョン管理システムの情報を組み込んだ IDE の拡張を提案した．このシステムでは，過去に存在したメソッド名を用いて現在の対応するメソッドを補完することなどができる．

これらシステムは，一部の重要な操作（コミットや履歴の閲覧など）において既存の VCS と同じ複雑なインタフェースを利用している．そのため，本研究で目指す小規模バージョン管理に適しているとは言えない．

2.3 プログラマ向けインタフェースデザイン

Brandt らは，プロトタイプを素早く作るような場合のプログラミング方法を Opportunistic Programming と名付け，これをする際のプログラマの行動を研究した [1]．その結果，頻繁にプログラムを実行する，print 文によるデバッグを良く用いる，などの傾向があることがわかった．また，Opportunistic Programming をしている際のプログラマは，数年単位の開発をサポートする既存の VCS よりも，より短時間の開発をサポートするようなバージョン管理を望んでいることもわかった．Vakilian らは，自動リファクタリングツールをプログラマが利用する理由を調査した [7]．結果として，プログラマは軽量なイ

[2] http://www.vim.org/scripts/script.php?script_id=3304
[3] https://www.gnu.org/software/emacs/

ンタフェースを好むこと，主に小さい変更を，自動ツールを用いて行うこと，そして結果のプレビューを見ることは殆どないことを明らかにした．

本研究で提案するシステムでは，これらの研究で得られた知見を利用し，小規模バージョン管理に適したアルゴリズムのユーザインタフェースを構築する．

3 要件明確化のためのインタビュー調査

小規模バージョン管理をプログラマがどのように行うのか，またその問題点は何かを把握するために日頃からプログラミングを行っているプログラマに対してインタビュー調査を行った．インタビューの対象者は，筆者らの所属する大学の学部生 4 人である．全員 2 年以上のプログラミング経験とバージョン管理システム（Git は全員，2 人は Subversion も）の使用経験がある．

参加者全員が，小規模バージョン管理にあたる操作を経験したことがあると答えた．参加者それぞれのプログラミング経験と，小規模バージョン管理に主に用いるツールを表 1 に示す．P2 は，Git を利用できる場合は Git を利用する事もあると答えた．また，参加者全員が小規模バージョン管理を行うためにコメントアウトは利用したことがあると答えた．

表 1. インタビューの参加者と小規模バージョン管理に使用するツール．

参加者	プログラミング経験	主に使用する方法
P1	3 年	ツールは使用しない
P2	4 年	ツールは使用しない
P3	2 年	Git
P4	4 年	線形 undo

現在の方法の問題点としては次のようなものが挙げられた．まず，ツールを利用しない場合，プログラマの記憶をもとに小規模バージョン管理操作をする必要があるため，ミスが発生するという問題である．P1，P2，P4 は，コメントの消し忘れ，コメントアウトの戻し忘れや，以前書いたソースコードを忘れて戻せなくなると言った問題に直面したことがあると答えた．また，P3 は，しばしば意味のない文言 (例: hoge) をコミットメッセージとして Git レポジトリにコミットするため，小規模バージョン管理操作時にどのコミットを操作対象とするかは自分の記憶に頼っていると答えた．しかし，直前数回のコミットなら内容を覚えているため，問題になったことは少ないと答えた．

P1，P2，P4 は，線形 undo の問題点として，編集の仕方によっては使えない場合があることを挙げた．P2，P4 は，この問題点を解決する手段として木構造を持った undo の存在を知っていたが，使用したことはないと答えた．

P1，P3 は，Git の問題点として，コミットや履歴を参照する作業は少々面倒であることを挙げた．P1，P2，P4 は Git を利用しない理由として，Git を利用するよりも線形 undo やツールを利用しない方法の方が，小規模バージョン管理操作を素早く行うことができる事を挙げた．また，P1 は Git を小規模バージョン管理に利用しない理由として，Git レポジトリなどのバージョン管理システムの履歴には小さな試行錯誤の後を残したくない点を挙げた．

インタビュー調査によって，全参加者がコメントアウトと，テキストエディタでの通常の削除を使い分けている事がわかった．使い分ける基準として，書いた内容を覚えておくことが難しい複雑なソースコードなど大切な箇所をコメントアウトする場合（P1）と非常に小さな変更でなければ基本的にコメントアウトを利用している場合（P3) があることがわかった．

上記の結果から，小規模バージョン管理操作はプログラミングに必要な作業であるが，現在のツールにはいくつかの問題がある事がわかった．そして，小規模バージョン管理のためのツールには，

- プログラマの記憶に頼らず操作できるインタフェース
- 多くの小規模バージョンを行う事ができる機能
- 素早く操作できる簡単なインタフェース

が必要であることがわかった．

4 提案手法

インタビュー調査の結果をもとに，単純で素早く利用できるインタフェースと，十分な機能を両立した小規模バージョン管理システムを提案し，実装する．

このシステムでは，テキストの編集中に，図 2 のようにパネルがテキストエディタ上に表示される．パネルは，図 2 A，B のような，エディタ上に表示されるコンポーネントであり，対応するバージョン管理操作後のテキストが表示される．各パネルをクリックすることで対応するバージョン管理操作を行うことができる．そして，ユーザが行う可能性が高い操作を推測することで，テキストエディタ上に表示するパネルの数を抑制しつつ様々な状況に対応する．

図 3 のプログラミングを例として，このシステムの動作を説明する．プログラマは，HTML を操作する JavaScript のプログラムを記述している．ユーザは，プログラムが完成した時点で HTML をプレビューする事で動作を確認する（図 3 ステップ 1）．次に，ユーザは Web ページのデザインを調整するために，テキストの大きさを変更する．プログラム中のパラメタ 16px を 32px に書き換えることで，特

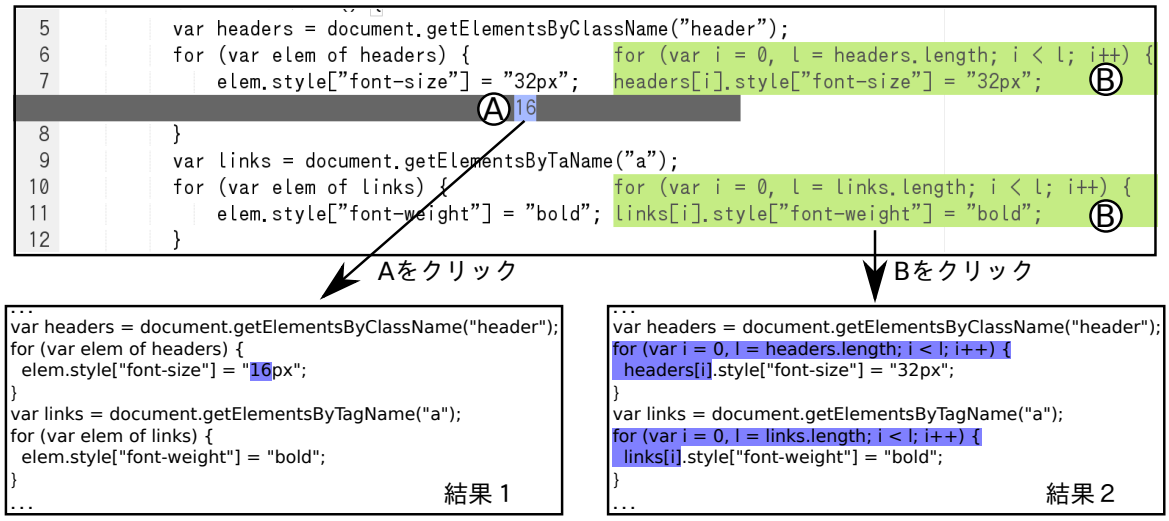

図 2. 提案システムの動作例.

定のテキストを大きくする（図 3 ステップ 2）．そして，変更後の Web ページを確認するために HTML をプレビューする．最後に，プログラムを単純にするため，プログラム中の for 文を for ... of 文に書き換えることにする．for 文はプログラム中の 2 箇所で使われているため，これを書き換える（図 3 ステップ 3）．そして，リファクタリング後のプログラムが動作していることを確認するため，HTML をプレビューする．

このプレビューにおいて，プログラムが適切な動作をしなかった場合を考える．例えば，Web ブラウザが for ... of 文をサポートしていない場合は適切な動作をしない．この時，ユーザはステップ 3 の変更を取り消したいと考える．提案ツールは，図 2 の画面を表示しており，ユーザは，B のパネルのどちらかをクリックすることで，ステップ 3 の変更を取り消すことができる（図 2 結果 2）．また，ユーザが図 3 ステップ 2 で変更したテキストの大きさの変更が不適切だったと感じる場合も考えられる．この場合は，ユーザは図 2 A のパネルをクリックすることで，ステップ 2 の変更を取り消し，テキストの大きさを 16px に戻す事ができる（図 2 結果 1）．

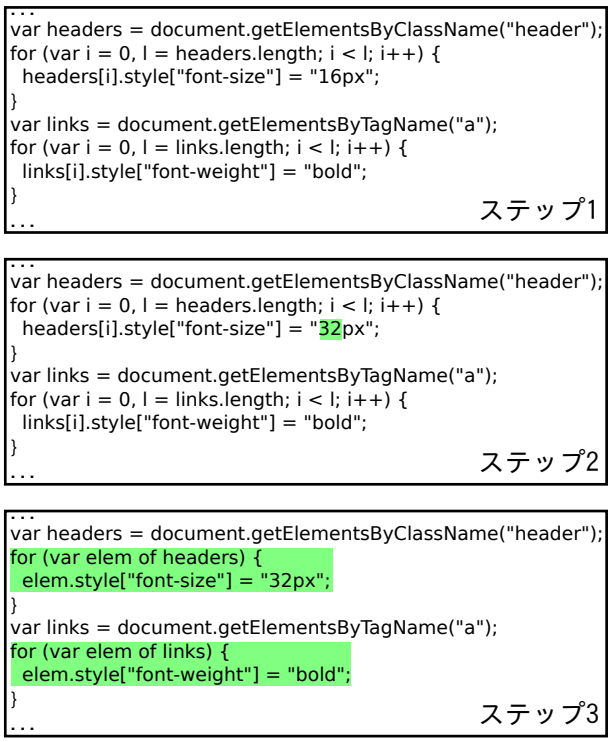

図 3. 提案手法を利用できるテキスト編集の例

5 実装

プロトタイプは，Ace テキストエディタ [4] を利用し，HTML や JavaScript をテキスト編集の対象とするツールとして実装した．しかし，提案手法は特定の言語の機能を利用したものではないため，全てのプログラミング言語に対して用いることができる．また，テキスト間の差分を得るために，EGit[5] をライブラリとして利用した．

提案手法の実装には，次の 2 つのアルゴリズムが必要である．テキストエディタには編集中のソースコードが表示されているため，多数のパネルを表示することは困難である．そのため，ソースコードのバージョンを管理し，ユーザが行う可能性の高いバージョン管理操作を推測するアルゴリズム（セクション 5.1）が必要である．本プロトタイプでは，このアルゴリズムは既存研究の手法を利用して実装した [9][2]．

[4] http://ace.c9.io/
[5] http://www.eclipse.org/egit/

また，表示するバージョン管理操作を，どのようにパネルとしてテキストエディタ上に表示するかを決定しなければならない．そのため，前記のアルゴリズムで推測されたバージョン管理操作から，表示するパネルを生成するアルゴリズム（セクション 5.2）が必要である．

5.1 バージョン管理アルゴリズム

実装するプロトタイプでは，プログラムの編集履歴の管理のために，動的セグメント（Dynamic Segment）を利用する [9]．この手法は，Git などの現在一般的な編集履歴の管理方法と異なり，編集のコンフリクトの解決を自動で行うことができる．提案手法では，コンフリクトを対話的に解決することは難しいので，自動でコンフリクトを解決できる動的セグメントを利用する．

動的セグメントによる編集履歴の管理では，編集の粒度は自由に決める事ができる [11]．本プロトタイプでは，編集中のプログラムを実行した時，それまでの編集を保存するように実装する [2]．既存研究では，実験的なプログラミングをする際やプロトタイプを作る際，ユーザは頻繁にプログラムを実行することが知られている [1]．そのため，プログラムの実行時に編集を保存する手法は，線形 undo などのツールよりも粗く，Git などの VCS よりも細かい粒度でソースコードの編集を管理することができると考えられる．

プログラムが実行され，それまでの編集が保存された後，ユーザが次に行う可能性の高い小規模バージョン管理操作を推測する．本プロトタイプでは，まずテキストエディタ上に表示するバージョン管理操作の最大数を定数パラメタ N として定める．そして，最新 N 個のテキスト編集をそれぞれ選択的 undo する操作が，ユーザが次に行う可能性の高い操作であると推測する．現在の実装では，フォーカスされている箇所などに関係なく，単純に操作履歴全体中の直近 N 個が表示される．動的セグメントを利用する選択的 undo アルゴリズム [9] は，コンフリクトがある場合，その解消方法として，最大 3 つの undo 操作を出力する．そのため，推測した操作の結果が N 個を超えてしまう事がある．この場合は，新しい方から順に N 個以内の操作のみを推測結果とする．なお，プロトタイプの実装では $N = 5$ とする．

この実装アルゴリズムは，プログラマは最近のテキスト編集を操作対象にする可能性が高いこと [9][1] に着目したものである．

5.2 パネル表示アルゴリズム

上述のアルゴリズムにより，テキストエディタに表示するべき小規模バージョン管理操作を得ることができる．本プロトタイプでは，これらの操作から，その操作を表すパネルを生成し，表示する．このアルゴリズムの入力となるのは，テキストエディタに現在表示されているソースコードと，小規模バージョン管理操作後のソースコードの 2 つである．差分を取る既存アルゴリズム（Histogram Diff）を用いて，2 つのソースコードの変更点を取得する．現在の実装では，操作の内容を「行への操作」と「単語への操作」の 2 種類に分けて，それぞれに応じたパネルを生成する．まず，1 行を単位として差分を取る．変更箇所が 1 行未満に収まらない場合には，行への操作とする．変更箇所が 1 行未満に収まる場合は，単語へ操作とする．単語への操作と判断された場合には，変更前後の当該行を比較して、その単語単位の差分を取得する．

変更点の種類に応じて，パネルは 6 種類存在する．図 4 は，パネルの種類とそれが利用される条件を示したものである．例えば，図 4 A のパネルは，**挿入される行**を削除する編集を undo し，その行を再度挿入する，というバージョン管理操作を表示する場合に用いられる．

各パネルに対し，変更点に対応するテキストの右または下（これはパネルの種類に応じて決定される）に表示するための領域が確保され，パネルはそこに表示される．もし対応するテキストの右や下に既にパネルがあった場合は (同一の行や単語に対して複数回の操作が行われた場合)，そのさらに下や右に領域を確保する．また，テキストの下にパネルを表示する場合（図 4 A，B，D，F）や，変更前と変更後のテキストの行数に差がある場合は，行間に適切な長さの余白が挿入される．

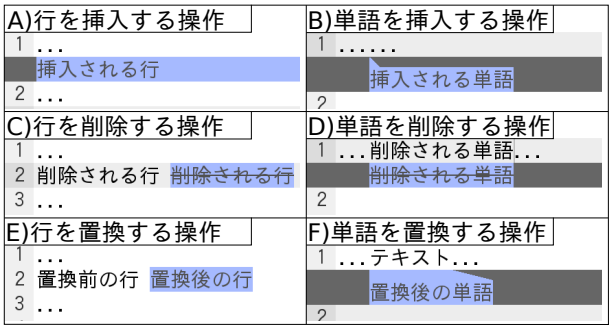

図 4. 提案手法で用いるパネルの種類．テキスト間の変更点の操作に応じてこれらを使い分ける．

例として，図 3 のステップ 3 において，ステップ 2 の変更を取り消すバージョン管理操作に対応するパネルを生成する場合を考える．テキストエディタに表示されているソースコードはステップ 3，ステップ 2 の変更を取り消した後のソースコードは図 2 の結果 1 となるので，これら 2 つのテキストが入力として与えられる．まずこれの行を単位とする差分を取ると，headers[i]. style から始まる行を置換する

という変更点が得られる．次に，この変更点は 1 行未満に収まっているので，1 文字を単位とする差分をとると，for 文内の代入文の右辺の 32 を 16 へ置換する，という変更点が得られる．これは単語の置換に当たるので，図 4 F のパネルを利用し，置換される単語（32）の下に置換する単語 (16) を表示することを決定する．他に同じ位置に配置されるパネルはないので，このパネル表示アルゴリズムによって，図 2 A のようにパネルが表示される．

小規模バージョン管理操作によっては，図 2 B のように，1 つの小規模バージョン管理操作に対応するパネルが複数になることがある．同じ操作に対応するパネルは背景色を同じとすることで，ユーザが操作とパネルの対応を理解できるようにしている．

6 現在の実装の問題点

現在のインタフェースでは，表示されたパネルがテキストエディタの使用の邪魔になる可能性があり，パネルの表示方法には改善の余地があると考えられる．例えば，変更点を部分的にパネルに表示できるようにする，ドロップダウンメニューを用いるといった改善方法が考えられる．

バージョン管理アルゴリズムについて，本論文では非常に単純なアルゴリズムを使用しており，改善の余地がある．まず，新しい編集をユーザに提示するようにしているが，実際には古い編集を操作したいと考える可能性があると考えられる．このような場合は本論文のアルゴリズムは上手く働かない．また，ユーザの編集作業の情報を推測に利用する事で，より効果的なアルゴリズムにできると考えられる．例えば編集中のテキストに似ている undo 操作を優先的にユーザに提示する，などの手法が挙げられる．

7 結論と応用可能性

本稿は，プログラミングの中でも特にパラメータ調整などで利用可能な小規模なバージョン管理操作を支援するための新しいインタフェースを提案した．このインタフェースでは，最近の編集に対するバージョン管理操作がテキストエディタ上にパネルとして表示され，ユーザはそれをクリックすることで操作する．開発するインタフェースの要件を明確にするため，プログラマに対してインタビュー調査を行った．この結果，彼らはバージョン管理を記憶に頼って作業しており，もしくはバージョン管理のために複雑なインタフェースを使う必要があるといった問題点がわかった．このため，提案手法ではユーザの記憶に頼ること無く単純で素早く操作できるインタフェースを用いて，小規模バージョン管理操作を行うことができるようにした．

本稿ではシステムの動作例として Web システムの開発を例に挙げたが，これに限らず幅広く応用が可能だと考えている．例えば，論文執筆で LaTeX を用いる場合，原稿の修正履歴をビジュアルに表現することは難しいが，本研究で提案したシステムを利用すれば Microsoft Word のように編集履歴を表示しながら LaTeX で論文執筆が可能となると考えられる．再利用をしないようなスクリプトを書く場合，VCS を利用することは考えられないが，本稿で提案する小規模バージョン管理システムを利用する可能性はあるだろう．今後はこれらの応用可能性を追求しつつ，さらに実用性の高いシステムを開発していきたいと考えている．

参考文献

[1] J. Brandt, P. J. Guo, J. Lewenstein, S. R. Klemmer, and M. Dontcheva. Writing Code to Prototype, Ideate, and Discover. *Software, IEEE*, 26(5):18–24, 2009.

[2] M. Goldman, G. Little, and R. C. Miller. Collabode: collaborative coding in the browser. In *the 4th international workshop on Cooperative and human aspects of software engineering*, pp. 65–68. ACM, 2011.

[3] Y. Y. Lee, S. Harwell, S. Khurshid, and D. Marinov. Temporal code completion and navigation. In *ICSE 2013*, pp. 1181–1184. IEEE Press, 2013.

[4] J. Sametinger and A. Stritzinger. *Exploratory software development with class libraries*. Springer, 1992.

[5] D. W. Sandberg. Smalltalk and exploratory programming. *ACM SIGPLAN Notices*, 23(10):85–92, 1988.

[6] B. Steinert, D. Cassou, and R. Hirschfeld. Coexist: Overcoming aversion to change. *ACM SIGPLAN Notices*, 48(2):107–118, 2012.

[7] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson. Use, disuse, and misuse of automated refactorings. In *ICSE 2012*, pp. 233–243. IEEE, 2012.

[8] Y. S. Yoon and B. Myers. A longitudinal study of programmers' backtracking. In *VL/HCC 2014*, pp. 101–108. IEEE, 2014.

[9] Y. Yoon and B. Myers. Supporting Selective Undo in a Code Editor. In *ICSE 2015*. IEEE, 2015. to appear.

[10] Y. Yoon and B. A. Myers. An exploratory study of backtracking strategies used by developers. In *CHASE 2012*, pp. 138–144. IEEE Press, 2012.

[11] Y. Yoon, B. Myers, and S. Koo. Visualization of fine-grained code change history. In *VL/HCC 2013*, pp. 119–126. IEEE, 2013.

[12] 大江 龍人, 志築 文太郎, 田中 二郎. UnReC: 文字入力に基づいた Undo/Redo. In *WISS 2013*, pp. 187–188, 2013.