ソフトウェア工学　52－23
（1987・2・14）

テキストエディタの基本編集作業の評価テストについて

中林祥恵　　荒木俊郎　　都倉信樹

大阪大学　基礎工学部

本報告では，設計段階でのテキストエディタの評価法を提案し，エディタのモデルとエディタコマンドの導出について述べる．評価法として，エディタをモデル化し編集作業を実現する方法（コマンド列）を求めて作業所要時間を予測する，という方法を考える．エディタをモデル化することによって，コマンド列を求めることができれば客観的な評価が容易となる．そこで，基本的な編集機能を持つテキストエディタをモデル化し，モデルの評価を行った．その結果，このモデルにより基本編集作業に対する適切なコマンド列を導出することができた．

A Method for Evaluations of Text Editors

Yoshie Nakabayashi, Toshiro Araki, Nobuki Tokura

Faculty of Engineering Science, Osaka University
1-1, Machikaneyama, Toyonaka, Osaka, 560 Japan

A method for evaluation of text editors in design phase is proposed. An editor model and how to generate a command sequence are described. The method consists of modeling editors, generating a command sequence for performing an editing task and predicting user performance time. It is possible to evaluate editors objectively and easily by generating command sequences. By using the method above, this editor model is assessed with respect to a number of basic editing tasks. It is concluded that this model can derive nearly optimal command sequences.

1．まえがき

テキストエディタは，プログラムや文書を作成するために計算機を利用する人の多くが用いるシステムである．従って，人間にとって使い易いシステムであることが望まれる．そのために『使い易さ』の向上を試みた数多くのテキストエディタが作成されている．しかしながら，『使い易さ』の実現は設計者の主観的な判断に委ねられることが多い．そこで，エディタの客観的な評価法を求める研究や設計の際に役立てるためのユーザモデルの研究が行われている．

その中で，Robertsの研究[1,2]はテキストエディタのひとつの評価法を与えている．Roberts は，『作業時間，誤り，学習時間，エディタの機能』の４つの側面によってエディタを評価している．完成されたエディタの評価にはこの方法が有用であるといわれている．

だが，設計段階でエディタを評価するための有効な方法を求める研究はあまり行われていない．ひとつの評価（時間的な側面について）の方法として，エディタをモデル化し編集作業を実現する方法（コマンド列）を求めて所要時間を予測する，という方法を提案する．編集作業とは，ある文書上での複数個の挿入，削除，置換といった一連の作業を意味する．

所要時間の予測には，Cardらが提案したＫＬモデル(Keystroke-Level Model)[3]を用いる．このモデルは，対話的な計算機システムの利用者の作業時間を予測する能力を持つ．利用者が誤りを犯すことなく最適な方法で作業を行うという仮定のもとで，テキストエディタについては，かなりの精度で予測できるといわれている．

しかし，ＫＬモデルを用いるには，編集作業を実現する方法を求める必要がある．設計者自身がこの作業を行うとすれば，手間がかかりしかも最適な方法が設計者の主観的なものになる可能性がある．ここで，設計段階でのエディタをモデル化することによって，編集作業を実現する方法を求めることができれば，設計段階での客観的な評価が容易になる．

そこで，エディタを論理型言語ｐｒｏｌｏｇを用いてモデル化し，編集前後のテキストからその編集を実現するエディタのコマンド列を導出する．コマンド導出の対象とする編集作業として，多くのエディタで実行できる基本的な作業（基本編集作業）というものを考える．

本報告では，基本的な編集機能をもつテキストエディタをモデル化し，モデルの評価を行った．その結果，このモデルにより基本編集作業に対して適切なコマンド列を導出することができた．

2．　エディタモデル

テキストの編集は,まとまった『単位作業(ｕｎｉｔ ｔａｓｋ)』を順次処理することとみなせる．『単位作業』とは，ある特定の位置へのある特定の単語の挿入やテキスト断片の移動といった作業をさす．１つの単位作業の実行は，

・エディタに位置を示すこと(ｐｏｉｎｔｉｎｇ)，

・修正をほどこすこと(ｍｏｄｉｆｙ＿ｔｅｘｔ)

の２つに大きく分けられる．エディタの基本的な機能も同様に

・位置ぎめをする機能（カーソル移動機能）

・修正をほどこす機能（編集機能）

の２つに大きく分けられる．そこで，エディタをモデル化することにより，編集機能のコマンドを導出し，カーソル移動機能としては１文字移動のみを用いる．

以下では，エディタのモデルと，編集機能のコマンド導出の手法について述べる．

2．1　モデル開発

テキストエディタの基本要素を，編集の対象となる『テキスト』と対象を操作するための『機能』とする．テキストは改行コードを含む文字列として表現され，テキスト上で現在注目している文字を表すためにカーソルを用いるとする．

テキストの先頭からカーソルの位置までのテキストをテキスト前部，カーソルの位置からテキストの最後までのテキストをテキスト後部と定義する．カーソルから見て，テキストの先頭の方向を後方，テキストの最後への方向を前方と呼ぶことにする．

エディタは，機能の実行（コマンド）によってテキストの状態を遷移させるものとみなせる．以下に述べるように，コマンドとテキストの状態遷移との関係を表すことによってエディタをモデル化する．

テキストの状態をテキストの内容とカーソルの位置によって表すために，２字組（テキスト前部，テキスト後部）で表現する．カーソルはテキスト後部の先頭の文字上にあるとする．編集機能のコマンドによるテキストの状態遷移について述べる．

まず，例として一文字挿入について説明する．一文字挿入は現在のカーソル位置に入力した文字を挿入しカーソルを一文字分前方に移動する操作である．従っ

て，現在の状態を(X,Y)とし，入力した文字をAと表すとすれば，一文字挿入insert(A)による状態の遷移は，

(X,Y)⇒(X+A,Y)

と表される．但し+は連接を表す．

同様にして他の基本的な編集機能についても以下のように表現される．

カーソル上の１文字削除　char_del(A)

(X,A+Y)⇒(X,Y)

カーソル後方の１文字削除　bs(A)

(X+A,Y)⇒(X,Y)

単語削除　word_del(W)

(X,W+Y)⇒(X,Y)

行削除　line_del(L)

(X,L+Y)⇒(X,Y)

ブロック削除　block_del(B)

(X,B+Y)⇒(X,Y)

ブロック移動　block_move(B2)

(B1+B2+B3,Y)⇒(B1+B3,B2+Y)

ブロック複写　block_copy(B2)

(B1+B2+B3,Y)⇒(B1+B2+B3,B2+Y)

また，前方への１文字移動（以後コマンド列中では→で表す）も

(X,A+Y)⇒(X+A,Y)

と表現される．

X,Y,B1:任意の長さの文字列
B,B2,B3:長さ１以上の文字列
A:１文字
W:テキスト後部の先頭の単語
L:テキスト後部の先頭の行

以上のコマンドとテキストの状態遷移との関係をホーン節表現の規則として記述する．

[記述例]

```
c_text(X,Y,[insert(A)|M]):-
                    c_text(X+A,Y,M).
c_text(X,W+Y,[word_del(W)|M]):-
                    c_text(X,Y,M).
```

このような規則の集合として表されるモデルを基本形と呼ぶ．上記の編集機能を持つエディタの基本形を図１に示す．

エディタモデルにおいて，編集作業は編集前テキストから編集後テキストまでの遷移として表現される．編集前テキストの状態を(X0,Y0)とし，編集後テキストの状態を(Xn,Yn)とする．また，この編集を実現するコマンド列をC1C2‥‥Cnとする．

(X0,Y0)から(Xn,Yn)までの遷移は次のように表現できる．

(X0,Y0,C1C2‥‥Cn)⇒(X1,Y1,C2‥‥Cn)⇒
⇒(Xn-1,Yn-1,Cn)⇒(Xn,Yn,λ)
(λは空列を表す)

ここで

(Xi,Yi,Ci+1‥‥Cn)⇒(Xi+1,Yi+1,Ci+2‥‥Cn)

は，状態(Xi,Yi)からコマンドCi+1を実行することで状態(Xi+1,Yi+1)に遷移することを表す．

そして，(X0,Y0)と(Xn,Yn)からC1C2‥‥Cnを求めることを考える．基本形をｐｒｏｌｏｇのプログラムとして記述し，さらに，編集後テキストを事実とすることで，質問?-c_text(X0,Y0,W)を行えば変数Wに(X0,Y0)から(Xn,Yn)への編集を実現するコマンド列を得ることができる．テキストとコマンド列はリストで表現する．また，ｐｒｏｌｏｇのプログラムとして探索を行うという立場から，以後コマンド列を解とも言う．

編集後テキストを表す事実

```
c_text(Xn,Yn,[]).
```

コマンドとテキストの状態遷移との関係を表す規則

```
c_text(X,Y,[insert(A)|M]):-
                    c_text(X+[A],Y,M).
c_text(X,[A|Y],[char_del(A)|M]):-
                    c_text(X,Y,M).
c_text(X+[A],Y,[bs(A)|M]):-
                    c_text(X,Y,M).
c_text(X,W+Y,[word_del(W)|M]):-
                    c_text(X,Y,M).
c_text(X,L+Y,[line_del(L)|M]):-
                    c_text(X,Y,M).
c_text(X,B+Y,[block_del(B)|M]):-
                    c_text(X,Y,M).
c_text(B1+B2+B3,Y,[block_move(B2)|M]):-
                    c_text(B1+B3,B2+Y,M).
c_text(B1+B2+B3,Y,[block_copy(B2)|M]):-
                    c_text(B1+B2+B3,B2+Y,M).
c_text(X,[A|Y],[→|M]):-c_text(X+[A],Y,M).
```

図１　基本形

２．２　モデルの評価

モデルを実際にｐｒｏｌｏｇのプログラムとしてインプリメントし，その実用性を評価した．評価実験に用いた処理系はパーソナルコンピュータPC-9801VM上のProlog-KABAである．評価実験では，次の３つの作業（タスク）を用いる．

ａ）　タスクａ

("","polorg")⇒("prolo","g")

ｂ）　タスクｂ

("prolo","g␣")⇒("prolog␣program","␣")

ｃ）　タスクｃ

("␣prolog␣program","")⇒
("program␣in","␣prolog␣")

まず初めに基本形について調べた結果，タスクａに対して解は得られず，Stack Over Flowを起こした．

このように基本形をｐｒｏｌｏｇのプログラムに書き直しただけでは不十分である．以下では，適切なコマンド列を得るために用いたソフトウェア的な手法（手法１，手法２，手法３）とその評価の結果について順に述べる．以下では

形式１は，基本形に手法１を付加したもの
形式２は，形式１に手法２を追加したもの
形式３は，形式２に手法３を追加したもの

をそれぞれ表す．

１）手法１　状態遷移数を制限する

状態の遷移数の上限，言いかえれば，コマンド数の上限を決めて，その範囲内で解を求めるようにする．そのために，入力として与える遷移数上限から遷移毎に１ずつ減じて，０になったところで探索の打ち切りを行う．形式１は，次のようにして得られる．まず，基本形の各規則の節を次のように変更する．*はその引数を変更しないことを表す．

```
N:整数
c_text(*,*,*,N):-c_text(*,*,*,N-1).
```

そして，４番目の引数の値が０になったところで，探索を打ち切るために次の節を追加する．

```
c_text(_,_,_,0):-!,fail.
```

タスクａとｂを用いて形式１を評価した．

ａ）タスクａについて

結果を表１（第１列）に示す．以下で，Nは遷移数（コマンド数）の上限を表す．

・N=1,2,3,4,5というコマンド数で作業を実現できる方法がないことがわかった．

・N=6で最初に出力された解は

→ins(r)→→→ch_del(r)　‥‥　（解１）

である．これは一番効率のよい方法であろう．解１以外の21個の解はすべてblock_delとinsertを組み合わせたものであった．例えば

block_del(polor)ins(p)ins(r)
ins(o)ins(l)ins(o)

のような確かにコマンド数は6であるが，人間がこの作業を行うとすれば普通は使用しない方法である．つまり文字レベルの操作にブロック消去を使用している．

・N=7で最初に出力された解は

→ins(r)→→→→bs(r)

である．これは人間も使用しうる方法である．２番目の解が解１である．N=6の場合になかった解としてはcopyを使用した方法がある．例えば

→ins(r)→→→copy(ol)block_del(olr)

他には

→ins(r)→→ins(o)ch_del(o)ch_del(r)

のような同じ文字を消して書くという無駄を含んだ方法である．300個以上の多量の解を出力した．

ｂ）タスクｂについて

5時間30分経過しても解が得られなかった．上限値Nが大きくなると大幅な時間がかかることがわかる．

以上のように，非常に多くの解が出力され，そのほとんどは期待しない解であることがわかった．また，大幅な時間がかかる結果となった．その原因として，すべてのコマンドを編集の実現に役立つものとして探索を行っていることがあげられる．そこで，探索するコマンドをある程度制限する．その手法について次に述べる．

２）手法２　コマンド候補を与える

編集の実現に使用するコマンドの候補を入力として与える．そして，コマンド候補の要素であるコマンドについてのみ調べる．形式２は，形式１の各規則の節を次のように変更することによって得られる．

```
c_text(*,*,[command|*],*,Candidate_list):-
        member(command,Candidate_list),
        c_text(*,*,*,*,Candidate_list).
```

述語memberは第１引数が第２引数の要素である時成功する．

タスクａ，ｂ，とｃを用いて形式２を評価した．

ａ）タスクａについて

結果を表１（第２列）に示す．候補は｛insert, char_del｝とした．N=6では解１のみを出力した．N=7で解１以外に出力された解は11個であった．N=8では全部で77個の解を出力した．N=7,8の場合に出力された解の多くは，

```
→ins(r)→→ins(o)ch_del(o)ch_del(r)(N=7)
→ch_del(o)ins(r)ch_del(l)
        ins(o)ins(l)→ch_del(r)  (N=8)
```

のような同じ位置の同じ文字を消して書くという無駄を含むコマンド列である．

ｂ）タスクｂについて

結果を表２に示す．候補は｛insert｝とした．N=7,8では解なし．N=9で僅か5秒で次の2つの解を出力した．

```
→ins(_)ins(p)ins(r)ins(o)
            ins(g)ins(r)ins(a)ins(m)
ins(g)ins(_)ins(p)ins(r)
            ins(o)→ins(r)ins(a)ins(m)
```

ｃ）タスクｃについて

結果を表３（第１列）に示す．候補は｛move, insert｝とした．N=4で出力された4個の解はいずれも適切な方法であったが，N=5において

```
move(log_)ins(_)ins(i)ins(n)move(_pro)
```

のような移動を２回に分けて行う方法が出力された．このような移動の仕方を普通は使用しないであろう．

表１　タスクａについての結果

N		形式１	形式２	形式３	形式４
1	t	0秒			
	m	0			
2	t	0秒			
	m	0			
3	t	2秒			
	m	0			
4	t	16秒			
	m	0			
5	t	1分52秒			
	m	0			
6	t１	1分10秒	2秒	22秒	1秒
	t	14分49秒	18秒	2分12秒	2秒
	m	22	1	1	1
7	t１	13分14秒	6秒	51秒	1秒
	t	1時間以上	55秒	5分26秒	10秒
	m	300以上	12	1	1
8	t１		16秒	1分47秒	1秒
	t		2分39秒	12分12秒	15秒
	m		77	11	11

以上のように，手法２を追加することによって解出力にかかる時間は小さくなることがわかった．しかし，手法２を追加するだけでは，期待しない解を除ききれないことがわかった．次に期待しない解，言いかえれば不適切な解を除くための手法について述べる．

３）手法３　探索途中のコマンド列を検査する

探索の途中でその時点までに使用したコマンド列を調べて不適切な系列(Invalid_com_list)を含んでいれば探索を打ち切る．形式３は，以下のようにして得られる．探索途中のそれまでに使用したコマンド列を保持するために，形式２の各規則の節を次のように変更する．

```
c_text(*,*,[command|*],*,*,Command_list):-
        c_text(*,*,*,*,*,Command_list+command).
```

さらに，この探索途中のコマンド列を検査するために次の節を追加する．

```
c_text(*,*,*,*,*,Command_list):-
        check(Invalid_com_list,Command_list),
        !,fail.
```

述語checkは第１引数が第２引数の部分系列である時成功する．

表２　タスクｂについての結果

N		形式２
7	t	2秒
	m	0
8	t	3秒
	m	0
9	t１	1秒
	t	5秒
	m	2

表３　タスクｃについての結果

N		形式２	形式３	形式４
4	t１	0秒	1秒	12秒
	t	52分	1時間15分29秒	22秒
	m	4	4	2
5	t１	14秒	46秒	20秒
	t	1時間27分34秒	2時間41分 4秒	33秒
	m	32	4	2

（注）　N　：遷移数の上限値
　　t１：最初の解の出力までの所要時間
　　t　：全解出力までの所要時間
　　m　：解の個数

タスクａとｃを用いて形式３を評価した．

ａ）タスクａについて

結果を表１（第３列）に示す．コマンドの候補を{insert,char_del}とし，次の４つの系列を不適切なコマンドの系列とした．

```
insert(A),*char_del(_),char_del(A)
insert(A),*insert(_),char_del(A)
char_del(A),*char_del(_),insert(A)
char_del(A),*insert(_),insert(A)
```

*A:コマンドAの任意回の繰り返しを表す

その結果，N=7では解１だけとなり，N=8では解１以外に出力された解が10個だけになった．

ｃ）タスクｃについて

結果を表３（第２列）に示す．コマンドの候補を{insert，move}とし，次の系列を不適切なコマンドの系列とした．

```
move(_),*,move(_)
```

*:任意のコマンドの系列を表す

その結果，N=5での解を適切な4つの解に絞ることができた．形式２のN=4での解と同じである．

以上のように，手法３を追加することで解を適切なものに絞れることがわかった．また，解出力にかかる時間は形式２と比較してコマンド列を検査する手間のために遅くなることがわかった．

基本形に以上の手法を付加することで適切なコマンド列を導出できることを確認した．しかし，基本形では，探索途中において編集後テキストから探索をうまく進めるための情報を得ることはなく，単純にコマンド列を編集前テキストにほどこした結果が編集後テキストと一致するかどうかを判定するだけである．挿入する文字や移動する文字列を決める情報を編集後テキストから得ていない．そこで，編集後テキストを参照しながら探索を進める工夫を行う．

４）手法４　編集後テキストを参照する

編集前テキストにおける現在のカーソル位置に対応する位置で編集後テキストを分割したものを<H,T>と表す．そして，(Xi,Yi)に対応するものを<Hi,Ti>とする．(Xi,Yi)から<Xi+1,Yi+1>に遷移するためのコマンドの決定に<Hi,Ti>を参照するように基本形を変更する．ここで用いた<H,T>の参照の仕方について述べる．

・１文字前方への移動(X,A+Y)⇒(X+A,Y)において，AとTの先頭が等しい．

・１文字挿入(X,Y)⇒(X+A,Y)において，AはTの先頭の文字とする．

・ブロック削除(X,B+Y)⇒(X,Y)において，Yの先頭の文字とTの先頭の文字が等しくなるようにBを決める．

・ブロック移動(X,B1+B2+B3)⇒(X+B2+B1,B3)において，TがT1+B1+T2となりかつB2の先頭とT1の先頭及びB3の先頭とT2の先頭が等しくなるようにB2を決める．

・ブロック複写(X,B1+B2+B3)⇒(X+B1+B2+B1,B3)において，TがB1+T1+B1+T2となりかつB2の先頭とT1の先頭及びB3の先頭とT2の先頭が等しくなるようにB2を決める．

・編集の完了は，(Xi,Yi)と<Hi,Ti>においてXi=Hi，Yi=Tiであるかどうかによって判断する．

基本形にこのような編集後テキストを参照する手法を加えた形式を図２に示す．この形式に上記の３つの手法を付加したものを形式４とする．

編集完了の条件を表す事実

```
c_text(X,X,Y,Y,[]).
```

コマンドとテキストの状態遷移との関係を表す規則

```
c_text(X,H,Y,[A|T],[insert(A)|M]):-
                c_text(X+[A],H+[A],Y,T,M).
c_text(X,H,[A|Y],T,[char_del(A)|M]):-
                c_text(X,H,Y,T,M).
c_text(X+[A],H,Y,T,[bs(A)|M]):-
                c_text(X,H,Y,T,M).
c_text(X,H,W+Y,T,[word_del(W)|M]):-
                c_text(X,H,Y,T,M).
c_text(X,H,L+Y,T,[line_del(L)|M]):-
                c_text(X,H,Y,T,M).
c_text(X,H,B+[A|Y],[A|T],[block_del(B)|M]):-
                c_text(X,H,[A|Y],[A|T],M).
c_text(X,H,B1+[A|B2]+[C|B3],[A|T1]+B1+[C|T2],[block_move(B1)|M]):-
                c_text(X+[A|B2]+B1,H+[A|T1]+B1,[C|B3],[C|T2],M).
c_text(X,H,B1+[A|B2]+[C|B3],B1+[A|T1]+B1+[C|T2],[block_copy(B1)|M]):-
                c_text(X+B1+[A|B2]+B1,H+B1+[A|T1]+B1,[C|B3],[C|T2],M).
c_text(X,H,[A|Y],[A|T],[→|M]):-
                c_text(X+[A],H+[A],Y,T,M).
```

図２　基本形に手法４を付加した形式

タスクａ，タスクｃを用いて形式４を評価した結果，表１（第４列），表３（第３列）に示すように解出力にかかる時間の改善がみられた．手法４によりコマンド導出の効率の向上を行えることがわかった．

さらに，形式４を次の３つの例題テキストに適用した．

ex1）横70文字　縦26行のテキスト
　6 行目から10行目までの段落の削除
　24行目の削除

ex2）横70文字　縦21行のテキスト
　4 行目で３単語の削除
　8 行目で単語の結合
　20行目で１文字挿入

ex3）横70文字　縦21行のテキスト
　7 行目から11行目までにわたる１文を1行前方へ移動

まず，得られた結果について述べる．

・ex1に対してN=6，候補｛char_del,word_del,line_del｝という条件で約11分で適切な5個の解を得た．

・ex2に対してN=8，候補｛insert,char_del,word_del｝という条件で最初の解の出力に３時間以上かかった．ただし，単位作業ごとに区切ってコマンドの導出を行えばそれぞれ短時間で解を得ることができた．

・ex3に対してN=1，候補｛insert,move｝とすれば約２分で解を得た．しかし，N=2,3とするとたいへん時間がかかる．

以上の結果から，編集操作の種類が同じ（ex1では削除）であれば，例題テキスト中に単位作業が複数個含まれても解が得られるが，種類が異なる場合（ex2）には単位作業に区切る必要があることがわかった．制約はあるが，形式４が以上のような例題テキストに対してもコマンドを導出できる能力を持つことを確認した．

２．３　議論

基本形に以上のソフトウェア的な手法を付加することで単位作業に対する適切なコマンド列を求めることができた．各手法について議論する．

手法１により，Nを順に大きくしていくことで最短の方法にどんな操作があるのかまた，その長さがいくらかを知ることができる．また，ある長さを想定したときに実現の方法の有無を知ることができる．人間が作業を行う場合にはあまりコマンドの長さを意識することはないが，ある作業を人間に与えたときに思いつく操作とこのモデルが出力する操作の比較によって，それが長いのか短いのかを知ることができる．

手法２は，被験者を使って編集作業の実験を行う場合に，おおまかな作業の内容を示す印をつけた文書を被験者に与えることに対応する．この手法はコマンドの種類が多くなれば有効であろう．しかし，ある作業に対して適切なコマンド候補を決める際には注意を要する．

無駄のある挿入と削除の繰り返しを含む方法やブロックの削除や移動，複写をブロックを分割して行う方法などを解の集合から除く場合に手法３が有効であることを確認できた．また，不適切な系列を知ることはエディタ利用者の持つ知識を知ることにつながると思われる．さらに，不適切な系列をうまく表現する方法を検討する必要がある．

手法４での編集後テキストの参照は，被験者が手もとの文書を見ながらエディタ内の文書を修正するという行動に対応する．

今後は単位作業をいくつも含む例題テキストを処理できるような工夫が必要である．例えば，中間テキストを与えるあるいは，テキスト中に作業の種類と範囲を示す情報を組み込んでモデルがその情報を利用するというような方法が考えられる．

３．あとがき

エディタのモデルを提案し，基本編集作業に対するコマンドの導出について述べた．基本形に３つの手法（手法１，手法２，手法３）を付加することで適切なコマンド列を導出することができた．さらに，手法４を用いることでコマンド導出の効率を上げることができた．

モデルを記述するために論理型プログラミング言語ｐｒｏｌｏｇを用いたが，ｐｒｏｌｏｇはバックトラックの機能やパターンマッチングの機能を言語機能として持つのでモデルを素直に記述することができた．

現在までに，このモデルによるエディタの評価に用いる例題について検討を終えた．今後，その例題を用いた評価を実際に行い，提案した評価法の検討を行う予定である．また，カーソル移動機能のコマンドを導出する手法についての検討も今後の課題である．

参考文献

[1] Roberts,T.L. and Moran,T.P.:
"Evaluation of Text Editors",
Communications of the ACM,26,4,pp.136-141
(1983).

[2] Roberts,T.L.:
"Evaluation of Text Editors",
Ph.D.Dissertation, Stanford Univ. report
STAN-CS-82-920,p.184 (1979).

[3] Card,S.K., Moran,T.P. and Newell,A.:
"The keystroke-level model for user
performance time with interactive systems",
Communications of the ACM,23,7,pp.396-410
(1980).