解　説

ソフトウェア製造ツールの最近の傾向†

和　田　英　一††

1. はじめに

ソフトウェア製造ツールの最近の傾向はなにだろうかと思って，いろいろ雑誌などをひっくりかえしてみたけれど，なかなかこれが傾向であると断言できるような文献はみつからない．案外ソフトウェアツール全般に関する記事は少ないようである．はじめから余談で恐縮だけれども，Software 誌に Software Tools Project という記事があった[1]．これこそ何かソフトウェアツールを計画的に作る話かと思って眺めてみたら，それはそれに違いないけれども，ここでいう Software Tools はいわば固有名詞で，例の Kernighan 達の本[2]のことであった．結論をいうならば，Newcastle upon Tyne の計算機科学科で，例の本にあったソフトウェアツールを，せっせと移植したという話で，ポイントは，もしシステムに Fortran があれば，またなにをかいわんやであるが，同所にはかの BCPL がすでにあるので，ブロック構造の Ratfor をわざわざ用意しなくても，BCPL に落す方向で実施したというところにあった．

ソフトウェアに関する記事は少ないようだと書いたけれども，本シンポジウムの発表のテーマを見ると，実に多種多様なツールが並んでいる．したがって最近のソフトウェアツールの傾向のひとつは多様化にあるともいえそうである．ところで発表の中にエディタが見当たらないが，これはやはり相当基本的なツールだと思う．エディタについてはあとでも触れるつもりでいる．

筆者は，ソフトウェアツールの最近の傾向はシステム化ではないかと思う．多様化とはむしろ反対の結論であるが，どうしてそう考えるかを2～3の例についてのべたいと思う．

2. ツールのシステム化

いま，ソフトウェアから離れて，ツールのシステム化ということを考えてみる．我々の日常生活環境には，いろいろなツールがころがっている．こういう，その辺にころがっているツールは殆ど単能である．栓抜き，つめ切り，虫めがね……といったものを連想すればよい．（ツールといった場合は，単能であるニュアンスが強い．ソフトウェア製造ツールというと，筆者はどうしてもまず 図-1 のようなドライバとスパナ的なものを連想する．シンポジウム本番では，筆者は自分で改造した8進法のソロバンを紹介した．）ところがそのうち，いくつかのツールを集めた集合ツールまたはツールシステムが登場する．スイスアーミーナイフ，回路テスタ，ツールボックス，シンクロスコープ……といったものを連想すればよい．（8進法ソロバン的なものの集合ツールとして，TI 社の電卓，Programmer を紹介した）．これらのシステムの特徴はなにかというと，

i）ある部分をすべての機能で共用する．

ii）応用レインジの少しずつ異なるものが揃っている．

ことであるらしい．アーミーナイフは柄というかケースを共用している程度であまり大したことはないが，その共用の結果，同程度の機能のツールをバラバラに集めたのにくらべ，全体の大きさが小さくなって携帯

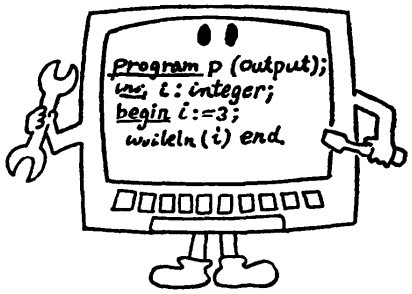

図-1

† Trends of Software Producing Tools by Eiiti WADA (Department of Mathematical Engineering, University of Tokyo).

†† 東京大学工学部計数工学科

に便利になった．回路テスタではまずメータは共用である．また測定レインジが電流でも電圧でも順次切り換えられる．ツールボックスでは筆者は順次に寸法のかわるスパナの列を想像しているのであるが，それは応用レインジの連続である．容器を共用しているというのは多少苦しいが，ひとまとめになっているのでシステムの資格を有することになる．もしこれがバラバラに買い集めたのだと，あまりシステムという感じがしないであろう．シンクロスコープは回路テスタと同様だが，システムは高級で，使い手にも充分な知識がいるという印象を与える．

システムというのは，

i）ひとつのものの使い方になれていれば，他のものも比較的容易に使える．

ii）共用部分があるし，応用レインジもうまく設計されているので，バラバラのツールよりも無駄が少ない→開発も簡単だし，費用も少ない．場所もとらない．ということになる．ソフトウェアもこの道をたどりつつあるのではないだろうか．

システムといった場合，どうしても引用しておきたいのは，高橋秀俊先生が，「数理の散策」にお書きになった「切りかえて挿しかえ」の記事である[3]．非常に示唆に富んだものであり，是非一読されるのがよいと思う．

3. ソフトウェアツールのシステム化

ソフトウェアツールのシステム化はもちろん各所のプログラムライブラリにも見られるけれど，それらはどちらかというとやや結果的にそうなったように思われ，最初からシステムを目指して作られたものとしてのツールは，前にも触れたSoftware Toolsであろう．これはしかし応用レインジが相補的という関係でシステムをなしているというよりは，あるツールを次に作るツールの下請けに使おうという，むしろ計画開発の見本的，あるいは教育的ツールの感さえあり，必ずしも実際のソフトウェア製造ツールの典型といえるかどうか筆者にはわからない．筆者は個人的にはRatforをはじめとして，Software Toolsのツールはあまり好きになれないが，プログラムの作り方の見本としては，なかなか面白いところもあると思った．

今回この稿を書くに当っては，Dolotta他のProgrammer's Workbenchの文献[4]と，SandewallのLispに関する文献[9]を参考とした．これらは簡単にいえば，それぞれ，操作システムUnixと，プログラム言語Lispのまわりに作られたフトウェアシステムである．それもそもソフトウェア製造ツールの最も基本的なものは操作システムとプログラム言語(の処理系)であり，そらを中心にしてシステム化したソフトウェアツールも，早晩われわれのまわりに登場する運命にある．多様化した各種のツールも，取捨選択整理統合されて，使いよく無駄のないシステムに生まれかわるに違いない．その方向を多少とらえていると思われるが，上記ふたつの文献で，特に前者には数篇の各論の文献が付随している．

4. Programmer's Workbench[4)~6)] (PWB)

PWBはベル研究所のプログラム開発システムである．このシステムの哲学は次のようにいえるであろう．ベル研にはIBMのをはじめ大型の計算機が何種類もある．そのそれぞれでプログラムを開発するとその対象機種によって異なるソフトウェアツールを用いなければならず，これは混乱のもとである．そこでどの機種に対しても同じユーザインタフェースで仕事ができるようなバッファシステムをより小型の機種PDP-11の上に作る．このシステムがPWBである．したがって開発者はPWBのユーティリティ，例えばエディタを一種類知っていればよい．ソースプログラムのファイルもすべてPWB上にある．PWBと対象機種との間は高速の通信回線で結合されており，PWBからRJE（リモートジョブエントリ.）で送られる．一方直接に対応機種でプログラムを開発すると，大型計算機の高価な計算時間を使用することになるが，PWBを使えば，開発中の小まわりな作業はすべて経済的なPDP上で行われることになる．さらにこれら開発用のツールはPWB上に一種だけ用意すればよい．

このような基本方針でPWBを作ったわけだが，当然のこととしてベル研のUnixの上に作られた．Unixのライブラリにはすでにいろいろなツールがあり，これだけでも評判のシステムであるが，Unixのツールのいくつかは PWB 用に改修された．そういう次第で PWB は操作システム Unix の拡張版である．したがって PWB のコマンドは Unix と同様な形をしている．システムの雰囲気は，やはり企業のものであるから，プログラム開発管理用ツール中心という感をまぬがれないが，オンライン用のテストドライバーはわれわれの研究室でも簡易版が作れないかと思うようなツールである．すなわちあるオンラインシステムを

開発したとき，開発者がテストするときの入力をあらかじめファイルに入れておき，フルスピードでそのファイルを食わせてシステムのテストを行う．システムの手直しが行われたとき，またそのファイルを入力してテストする．特に，ある入力に対して，どういう出力があれば，次にどうするかという指定ができたりで，高級な言語になっているみたいであるが，それほどまでは真似しなくても，入力がどんどんできるだけで大いに利用価値はある．われわれは，例えば Lisp の処理系を作ってディスプレイでテストするときは，毎回 (car' (foo. bar)) からやり直しているので，テスト洩れのケースも少なくないし，テストの能率も悪い．テストを一回やったとき，入力の情報がファイルされるだけでも，次回にそれを使うことができて，便利なのではないかと思う．

ドキュメントを作るための roff もいままでの Unix にあったものより改善されたのではないだろうか．大体この文献[4]と後続の関連文献はすべてこのドキュメントツール PWB/MM (MM は Memorandum Macro の意) で作成された由である．申しおくれたが，PWB には写植機があって，Software Tools の本であれ，この種の文献であれ，たちどころに作ることができる．

その他いろいろと作られているようであるが，PWB は要するにプログラム開発用ツールシステムで PDF (Program Development Facility) といわれるカテゴリーのひとつである．

PWB から多少はなれるけれども，去年の11月に IBM ワトソン研究所の Allen Brown さんが来日して，彼らの作っているシステムの一部，Lisp の Global Flow Analysis を紹介した．これは大変むづかしくて実はよくわからなかったけれど，Lisp のプログラムをいろいろな面でチェックしてくれるもののようであった．そこでこのアナライザの組み込まれる全体のシステム（つまりプログラミングの環境ということになるのだが，それ）は一体どんな形のものと想像したらよいかと質問してみた．Lisp のプログラミングシステムだから，あとでのべる Sandewall の文献のようなものかとたずねたら，Lisp はたまたまの例題の言語にすぎない (Sandewall の文献にもそう書いてあったようだが) のでそれはちがうという返事．あとはなにのようと聞いたか忘れが，ひょっとして PWB のようなものかとたずねたら，いままでの中ではそれに一番近いという答をくれた．つまり PDF のひとつということになるのであろう．そこでその全体のシステムのことを書いた文献が見たいのだが，それはまだどこにも発表されてないようである．

5. SCCS

話題をふたたび PWB にもどす．PWB のなかで面白いもののひとつは SCCS (Source Code Control System) である[7]．これはソースプログラムが次々と修正されて新しい版になってゆくとき，その経過をコンパクトに保存するためのものである．IEEE Transaction にでた報告を読んだ方もあると思うが，ここにその簡単な例を用いてエッセンスをのべておこう．

ソースプログラムの版は，release と level のふたつの番号で認別される．それをこの順にピリオドで区切って1.1とか1.2とかのようにあらわす．さてあるプログラムのテキストが，最初，つまり版1.1では図-2(a)のような4行からできていたとする．その後 CCC の行をとり，そのかわり DDD と EEE の行を入れて版1.2を作ったとする．つまり図-2 (b) である．その次の改訂では EEE と FFF をとって版1.3とし，さらに先頭に AAA を入れて版1.4を作ったとしよう．その部分は図-2の(c) (d)に示す通りである．このとき，ソースコードのファイルはそれぞれどんな状態かというとそれは図-2の (e) (f) (g) (h)に示すようになっている．これを見るには，次のことを知らなければならない．版 m, n で新しく挿入

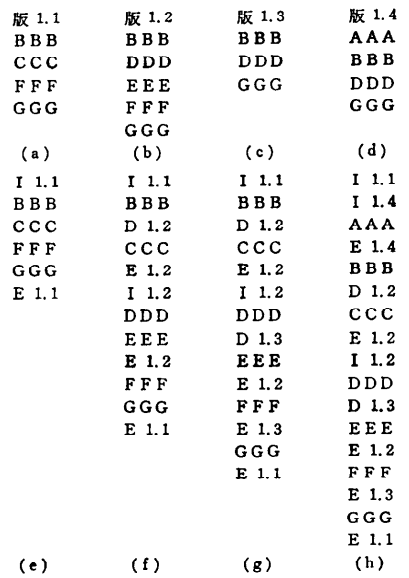

図-2 SCCS の例

された部分は I*m*, *n* と E*m*, *n* でかこまれている.

版 *m*, *n* で削除された部分は D*m*, *n* と E*m*, *n* でかこまれている.

そこで(e)を見るとすべての行は1.1で入力されたものだから, 先頭と最後に I1.1 と E1.1 がついたファイルになっている. 1.2 になると CCC の行を削除するからその前後に D1.2 と E1.2 がつく. また FFF の直前に DDD と EEE を挿入するので, FFF の直前には I1.2 と E1.2 でかこまれたそれらの行が挿入され(f)が得られる. 次に 1.3 になると EEE と FFF を削除するわけだが, この間に E1.2 があってもお構いなしにその2行の上下に D1.3 と E1.3 を入れてしまう. 最後は, 先頭の BBB の直前に AAA を入れるのだから I1.4 とも E1.4 でかこんだ AAA を BBB のまえにおけばよい. かくしてできたのが(h)で, これは1.1から1.4までのすべての版の情報をもっているから, いつでも任意の版を再現できることになる. 再現のアルゴリズムは手頃な演習問題であるから各自試みられたい.

SCCS には主要なみっつのコマンド, get, edit と delta がある. まず get で (途中の版もとれるけれども普通には) 最後の版を再現する. ついで edit のいろいろなサブコマンドでプログラムの修正を行う. edit しているそばから挿入や削除の情報がファイルに入れられるわけではない. edit が一段落したところで, 残るコマンド delta (差をとるの意) を実行すると, 最後の版と修正後の版との行単位の比較が行われ, 挿入や削除の情報が追加されることになる. ファイルの比較はこの頃いろいろな手法が知られているようである. 最近, 東工大の佐渡君, 前野君達の作ったファイル比較のプログラムが, われわれのところに入ってきてとても重宝しているが, それに似たような方法を採用しているのではないかと思う.

この種のファイルシステムないしエディタは別にこの PWB の SCCS に限らず, 他にもあるようだが, あまり知られていないようなので[8], 紹介してみた.

6. Lisp Programming System[9),10)]

Sandewall の文献の要旨は次のようなことである. プログラムの開発には各種の道具が必要だが, それもそのプログラム言語に密着したツールが自由に選べ, また必要とするツールが簡単に作れるようになっているのがよい. そういうプログラム言語と関連したツールの集りをプログラミングシステムとよぶことにするが, いまそういえるものは Lisp, それも BBN→Xerox の Interlisp と M. I. T. の Maclisp の程度しかない. そして以下主として Interlisp の諸機能の紹介ということになるのだが, その前にこのようなシステムが作れるためには, プログラム言語の記憶装置内の構造が容易にわかって自由に扱えるものであることがのぞましく, それにもっとも適しているのが Lisp と APL であろうといっている. (筆者は APL にはそれほど通じていないので, APL がどう適しているのわからないけれども, シンポジウムの予稿を書くすこし前に, APL を使いこなしている渡辺君という卒業生がたずねてきて, Sandewall の文献を読んだところ, ここも APL でできる, あそこも APL でできると思う箇所ばかりだったと報告してくれた.)

プログラミングシステムとして望ましい機能ということで最初に登場するのがエディタで, プログラム言語システムの中にエディタをとり込んでいるのがよいかそれとも, そのシステムの外に一般的なエディタをもっている方がよいかという議論になる. 結局はどちらということにはならないのだけれども, 筆者の Maclisp での経験によれば, Maclisp のなかにいるまま Multics のコマンドがよびだせて, QEDX のようなエディタが使えれば, セッション中の Lisp の環境がこわれないので, どちらでも同じだと思う.

Sandewall の文献には Lisp 用スコープエディタのよいのが M. I. T. にはあって, これからはそういうものも是非必要であると書いてあるが, これはもしかして Emacs エディタ[11)]のことかと思う. これは実は非常に難解なマニュアルで, Tom Knight の TV ターミナルを手元において使いながらでないとさっぱり様子がわからない. しかしたしかに Lisp 用コマンドの章があるので, このエディタであることに間違いはなさそうである. (Emacs エディタについてもう一言. 本年4月, 米国へ出張する機会があったので, 久しぶりに M. I. T. へたちよったら, Lisp マシンも Emacs エディタも動いていた. Emacs は BBN でも動いていたが, M. I. T. のものの方が強力である. Emacs を Lisp モードで使うと, Lisp の S 式の入力中, Pretty Reader のように働いて, 改行すると自動的に段下げがおき, また閉じかっこを入力すると, それに対応する開きかっこが点滅して, 大変便利そうであった.)

Lisp では 1.5 の時代から Tracer が用意されていた. Maclisp では grind とよぶが, Pretty Priuter もいまでは Lisp の処理系のうちのようである. 処理系

のうちと書いたけれども，Lisp の処理系はご存知のようにオープンエンデッドであるから，どこまでが処理系で，どこからがプログラミングシステムなのかあまりはっきりしない．なにしろコンパイラまで Lisp で書いてあって，それを使うまえに define してしまえば，処理系のような顔をするからである．Lisp の処理系は次々と Lisp で書いては雪だるまのように大きくなったということができる．Interlisp ではその機能を使って undo とか dwim (Do What I Mean) とかを組み込み，Lisp Programming System となった．このくらいになると高級なシンクロスコープと同様で普段はその一部ばかり使っているということになるであろう．もちろん新しい機能が簡単に追加できるところはシンクロスコープにはできないまねである．(この章を書く気を起させたのは，前述のように Sandewall の文献であるが，当の Sandewall さんは本年8月の人工知能国際会議に来日の予定であるので，直接またいろいろな意見がきかれることであろう．)

7. Lisp Structure Editor

Interlisp に直接組み込んであるエディタは，テキストエディタが文字を扱うのに対して式を扱ういわゆる構造エディタである．どんな感じで働くかをざっとここで紹介してみる．図-4 は Interlisp のマニュアル[12]から借用した例で，エディタとのセッションを示す．ただし左端の行番号は，ここでの説明用で，システムの出力ではない．セッションの目的は 図-3(a) に示すような Lisp のS式による関数 append の定義のエラーを修正して，同(b)のような正しい定義を得ることである．エラーはどこにあるかというと，

e1 パラメータ x y の y がかっこの外にでている．
e2 null の1がひとつしか書いてない．
e3 (null x) がTのときの値は x でなく y である．

```
[lambda (x)
  y
  (cond
    ((nul x)
    x)
    (t (cons (car)
        (append (cdr x y]
      (a)
[lambda (x y)
  (cond
    ((null x)
    y)
    (t (cons (car x)
        (append (cdr x) y]
      (b)
```

図-3 Interlisp のエディタでの修正前後のプログラム

```
 1 editf (append)
 2 edit
 3 *p
 4 (lambda (x) y (cond &&))
 5 *2p
 6 (x)
 7 *1p
 8 x
 9 *0 0 -1 p
10 (cond (&x) (t&)
11 *↑ p
12 (lambda (x) y (cond &&))
13 *(3)
14 *(2 (x y))
15 *p
16 (lambda (x y) (cond &&))
17 †f nul
18 *p
19 (nul x)
20 *(1 null)
21 *0 p
22 ((null x) z)
23 *f cond p
24 cond ?
25 *↑ (x x y)
```

図-4 Interlisp のエディタ (文献12)から)

e4 下から2行目の car のパラメタを忘れた．
e5 再帰的によぶ append のパラメタの閉じかっこの位置が悪い．

などにある．

さて，このエディタの促進記号は左むき矢印と*印である．1行目，append の定義をとりだす．カレントポインタは定義全体のリストをさす．3行目，カレントポインタのさすS式をプリントする．4行目に出力されているが，深さ2段をこえるリストは & で示されている．5行目，2で，カレントポインタをリストの第2要素へ移動させ，pでそのS式をプリントする．7行目，カレントポインタのリスト(x)の第1要素をプリントする．9行目，0は，カレントポインタのS式を要素とする，一番大きいリストへのポインタをあげる．0を2回で結局定義のS式までもどる．次の −1 は最後から1番目の要素のS式へ動かす意味で，プリントしてみると cond ではじまる式がとれている．&で省略される深さが，4行目のそれより一段深くなっていることに注意．11行目，↑は一番上のリストまで戻ることを示す．さて y をとり除くためには13行目のように(3)とする．これは第3要素を削除することの指示である．14行目は第2要素つまり(x)を(x y)で置き換えることを指示する．15行目でプリント指令をだし，16行目のように直っているのがわかる．17行目は nul を含むリストを見付ける(find)指示で，19行目のリストが見付かりカレントポ

インタもそこへおりている．第1要素を null で置き換え，21行目で1段あがる．23行目は cond をもつリストを探させているのだが，ないので cond? と応答した．25行目は一番上まで戻って，x をみつけてそれを y で置き換え (replace) させる．……

あまり長くなるからこの辺で打ち切るけれども，このような気分で働くのが Lisp の構造エディタである．このエディタも大変強力ではあるが，Interlisp が徐々に進化してきたの裏づけるかのように，コマンド大系はかなり雑然としているというのが筆者のうけた印象である．

先日，京都大学の数理解析研究所を訪れた際，同所の DEC の System 20 の Interlisp をほんのちょっとさわらせてもらった．なるほどエディタはとても快調であった．またタイプミスもしばしばおかしたので，dwim のお世話にも少なからずなった．便利かもしれないけれども，dwim はやはり時間が多少長目にかかるので，どちらかというと，親切すぎるのではないかと思われた．

Interlisp ももちろんスコープで対応できる．京大数解研のは普通の文字ディスプレイだが，本家の Xerox の研究所では Small Talk などに使う高分解能ディスプレイが使われている[13]．

8. Pascal の場合

Lisp がプログラミングシステムに成長していったようなことは，残念ながら Pascal ではそのままではできないようである．まず Pascal の処理系はオープンエンデッドではない．しかも Pascal は言語としてはなるべくいろいろな新しい機能を排除するという哲学であるので，プログラミングシステムとするには，処理系以外のモジュール，ツールとして構成しなければならない．新しいツールはもちろん Pascal で書いて作ることは可能である．こうして考えられる Pascal に密着したツールには Pascal 用 Pretty Printer, Pascal 用の名前置換えプログラム，Pascal 用のエディタ，そしてオプティマイザが考えられるけれども，そしてこのうちのいくつかは作ってみたけれども，Sandewall が最初にのべているように，このようにすると共通に使える筈のプログラムが重複して非常に無駄になることはたしかである．特に構文解析の部分は Pascal と名がつくからにはどれにも Pascal 用の共通の構文解析プログラムが必要である．名前置換えプログラムでは，われわれは，むしろ Pascal の処理系を手直しするという方法で開発した[14]．今後は Pascal のようなプログラム言語の処理系はプログラミングシステムの開発にも流用されることをはじめから考慮して設計するようなことになるであろう．

9. Pascal の構造エディタ

前述の Allen Brown さんは，彼らのシステムを紹介するついでに，プログラム開発システムのひとつに Mentor というのがあるともいって帰ったが，それはフランスの IRIA にある Pascal 用の構造エディタ[15]を含むシステムのことであるらしい．

Pascal 用の構造エディタについては，われわれのところでも試作してみたりしていたので，IRIA のシステムを大いに興味をもって調べてみたところ，やはりわれわれの懸念していたような問題点があるようである．それは Lisp の S 式とちがって Pascal のプログラムの，誰もが納得するような木構造のきめ方が著しく困難だということである．Pascal の文法書に従ったらよいではないかという意見もあろうが，例えば，〈項〉の構文にしたところで，

〈項〉:: =〈因子〉|〈項〉〈乗除算演算子〉〈因子〉

という書き方のものと，

〈項〉:: =〈因子〉{〈乗除算演算子〉〈因子〉}

という書き方のものとあり，困ってしまう．そういう次第で，無理に構造エディタを作るには，何とか強引に木構造をきめて，システムを作り，あとは利用者に馴れてもらうより仕方がないように思うが，IRIA でこのエディタを実際に使っている電総研の杉藤君の意見では，やはり仲々使うのも難しいそうである．でも早くなんとかして Pascal まわりにもよいプログラミングシステムを作ってみたいものである．

10. おわりに

ここにのべた以外にも，注意してみていると，ソフトウェアツールシステムといえるようなものがまだあることが段々とわかってきた．しかしそれらも，今回紹介したものも，どれをみても，まだなるほどこれなら絶対だと思うようなツールシステムは存在しない．かなりいい線をいっているのはやはり Unix と Lisp だけしかないように思う．システム作りは個々のツールを充分マスターした上で，バランスよく，センスよく設計しなければならないので，その道の名人の出現が望まれる．下手なシステムでも利用者が一旦それになれてしまうと，他のよりよいシステムへで

も，移行は極めて困難であるから，充分な注意が必要である．

参考文献

1) Snow, C. R.: The Software Tools Project, Software-Practice and Experience, Vol. 8, No. 5, pp. 585-600 (Sept. Oct., 1978).
2) Kernigham, B. W. and Plenger, P. J.: Software Tools, Addison-Wesley (1976).
3) 高橋秀俊: 切りかえと挿しかえ，「数理の散策」日本評論社 (1974).
4) Dolotta, T. A. and Mashey, J. R.: An Introduction to the Programmer's Workkench, Proc. Second Int. Conf. on Software Engineering, pp. 164-168, Oct. 13-15, 1976.
5) Dolotta, T. A., Haight, R. C. and Mashey, J. R.: The Programmer's Workkench, Bell System Tech. J., Vol. 57, No. 6, Part. 2, pp. 2177-2200 (1978).
6) Ivie, E. L.: The Programmer's Workbench-A Machine for Software Development, Comm. ACM, Vol. 20, No. 10, pp. 746-753 (1977).
7) Rochkind, M. J.: The Source Code Control Systems IEEE Trans. Software Engineering, Vol. SE-1, No. 4, pp. 364-369 (1975).
8) TSINKY: 差をとって儲けよう，プログラミングセミナー，bit, Vol. 11, No. 7, pp. 44-49 (July, 1979).
9) Sandewall, E.: Programming in an Interactive Environment: The Lisp Experience, ACM Computing Surveys, Vol. 10, No. 1, pp. 35-71 (1978).
10) Stallman, R. and Sandewall, E.: Structured Editing with Lisp, ACM Computing Surveys, Vol. 10, No. 4, pp. 505-508 (1978).
11) Ciccarelli, E.: An Introduction to the Emacs Editor, M. I. T., A. I. Laboratory, A. I. Memo 447 (Jan., 1978).
12) Teitelman, W.: Interlisp Reference Manual, Xerox Palo Alts Research Center, Oct., 1978.
13) Teitelman, W.: A. Display Oriented Programmer's Assistant, CSL 77-3, Xerox Palo Alto Research Center, March, 1977.
14) 和田英一，久保田稔: 名前置換えプログラム，Proc. TSC Symposium on Intelligent Programming Systems, IBM Japan, Nov. 23-25, 1978.
15) Donzeau-Gauge, V., Huet, G., Kahn, G., Lang, B., and Lévy, J. J.: A Structure Oriented Program Editor: A First Step Towards Computer Assisted Programming, R-114, IRIA, 1975.

(昭和54年7月11日受付)