解　説

Software Tool とは何か？†

木　村　泉††

0. はじめに

ちかごろ,「ソフトウェアツール」ということばをよく耳にするようになった．このことばが日本語の文脈の中で何を意味するかは必ずしも明らかでなく，古来「プログラミングツール」と呼ばれてきたものを単にこういい換えている，というに過ぎないふしもあるが，この流行が“software tool”という英語に触発されてはじまったものであることだけは，たぶんたしかであろう．

英語でいう“software tool”とはKernighanらの名著“Software Tools”[1]において導入された一つの特殊な概念を指すものと考えられる．このことばは，単に（プログラミング）ツールといったときには含まれていない，一定の強いニュアンスを含んでいる．

本文では，英語でいう“software tool”が単なるツールとどうちがうかを考察し，その意義を論ずる．本文の一つのねらいは，原典[1]において多数の実例と間接的な説明によって示されている software tool の概念に，著者なりの定義を与えよう，というところにある．すなわち本文では,「ツール」を，軽工業的ツールと重工業的ツールに分類し，software tool とは前者のことである，とする．

この“software tool”と単なる「ツール」とは，ともにきわめて重要であるが，ことに前者はわが国の情報処理技術の進歩のためには避けて通ることのできない大問題を提供していると思う．読者のご参考になればさいわいである．

1. ツールにおける重工業と軽工業

ソフトウェアにかぎらず，何を作るにしても，いっさいがっさい素手でやり，それ以外の可能性を考えない，というのは賢いこととはいえない．しかるべき道具を取りよせ，または作り，それを使って仕事をするのが常道である．ソフトウェア作りに役立つそのような道具のことを「ツール」というのだとすれば，それは昔ながらの（プログラミング）ツールにも，英語でいう“software tool”にもあてはまる．

そういう広い意味では，アセンブラ，コンパイラなども「ツール」である．コンパイル技法の研究が，かつては「自動プログラミング」の名のもとにおこなわれていた，という歴史的事実からもわかるように，これらはソフトウェア作りから手作業を駆逐する，というねらいのもとに導入されたものであり，事実これらの「ツール」がソフトウェア作成過程にもたらした革新には，偉大なものがあったといえる．

この，アセンブラ，コンパイラ……という線をずっと伸ばして行くと，会社の定款を放り込めば企業情報システムがさっと出てくるSF風ソフトウェアプラント，といったものに行きあたる．もちろんこれは，少なくとも今のところまったくの夢であるが，この種の「ソフトウェア重工業」に対する渇望が世間に行きわたっている度合いには驚くべきものがある．

ところで，英語でいう“software tool”とはまさにこのような，情報処理における「重工業」の対極に立つものだ，といえる．“Software tool”の立場は，いわばソフトウェア軽工業の立場であり，そしてそれゆえに目新しいのである．

もっとも，こういっただけではまだ何も説明したことにはなっていないけれども，ツールにおける重工業と軽工業のはっきりした区別については第3節にゆずり，その前に第2節では，Kernighanらの software tool がいかなるものであるかを，実例をあげて吟味してみることにしよう．

2. “Software tool”とは

2.1 例：文型検出プログラム

文献[1]には，“software tool”の例として，大はテキストエディタ，マクロ処理プログラムから，小はファイルの中の文字数をかぞえるプログラムまで，さまざまなものがあげられている．その規模も，原プログラムの行数にして数千行にわたるものから，わずか数行のものまで，いろいろであるが，その何たるかを理解するにはやはり文献[1]の冒頭に出てくる次の話を紹

† What is a Software Tool? by Izumi KIMURA (Department of Information Science, Tokyo Institute of Technology).

†† 東京工業大学理学部情報科学科

介するのが手っとり早いであろう.

いま, ある Fortran プログラムを A 計算機から B 計算機に移したい, としよう. A計算機とB計算機では入出力の具合が少しちがっているので, プログラムの中の FORMAT 文に一応全部目を通しておく必要があるとしたら, どうしたらよいか? (これはいかにもありそうな話である.)プログラムのリスティングを取り寄せて FORMAT 文に片っ端から赤鉛筆で印をつけて行く, というのではあまりよろしくない. FORMAT 文を拾い出して打ち出すプログラムをちょっと書いて, もとの Fortran プログラムをそれに掛けてみる, という方がずっとよい.

だが, 実はそれではまだ不十分である. そうやって書いた FORMAT 文検出プログラムは FORMAT 文の検出にしか役立たない. 同じようなことは同じような状況下で READ 文についても DIMENSION 文についても, その他いろいろのものについてやりたくなる可能性がある.

そこで考えられるのは, 一般的な文型検出プログラム (pattern finder) を作っておいて, それを使う, という手である. すなわち, パラメタとして与えられた文型 (pattern——"FORMAT", "READ" など)を含む行だけを入力ファイルの中から拾い出して出力するプログラムを作るのである. こういうものを1回作っておけば, あとは必要に応じてパラメタとして適当な文型を与えてやるだけのことですむ. 別々のプログラムをいちいち作る必要はない.

この文型検出プログラムのようなものが, 文献[1]にいう software tool である. それは種々の状況下で手軽に, かつ便利に使える. こういうものがあれば人は当然, 赤鉛筆など使おうとは思わなくなるだろう.

ここでちょっとおもしろいのは, 上でみつけ出されるものは実は FORMAT 文ではない, ということである. "FORMAT" という文型をさがしたとき, ひっかかってくるものの中には, FORMAT 文のほか, たとえば

```
FORMAT(I)=0.0  とか
DATA STRING/6HFORMAT/
```

とかいうようなものが含まれている可能性がある.

だが今の場合, 打ち出した結果は人間が目で見るのだから, それでちっともかまわない. 真の FORMAT 文検出プログラムを作ることはもちろん可能だけれども, そういうものは作るのにだいぶん手間がかかるし, 一般性がない. たとえば Cobol プログラムの移植には, まったく役立たない. おおげさになって, しかも通用力が失われるのでは, 引き合わない理屈である.

2.2 "Software tool" の複合利用

ところで, この文型検出プログラムはもちろん大変便利であるけれども, これひとつころがしておいたのでは software tool として十分生かして使ったことにならない. この種の道具は, いろいろ用意しておいて自在に組み合わせて使うと, ありがたみが増す.

たとえば, もとの Fortran プログラムが, 大文字小文字の両方を許し, それらを同一視するような処理系 (よくある) のために書かれていたとする. そして "FORMAT" が, ところどころ "format" とか "Format" とかいうようになっているかも知れないとしたら, どうしたらよいだろうか? そのとき, "FORMAT" だけが拾い出されて, 他の形のものが同じ意味をもつにもかかわらず無視されてしまってはまずい.

その場合に, もし小文字を全部大文字になおすための "tool" が別に用意されていたとしたらどうか? もしそうであれば, もとの A 計算機用プログラムをまずその "tool" に通し, その出力をさきほどの文型検出プログラムに食わせる, ということでうまく行く. (Kernighan らが愛用している) オペレーティングシステム UNIX[2]の記法では,

```
translit a-z A-Z | find FORMAT          (2.1)
```

となる. "find" は, さきほどの文型検出プログラムがこういう名前でシステムに登録されている, としたものである. "FORMAT" は find に与える引数であり,

```
find FORMAT
```

は, 要するにさきほどのことをせよ, との指令である.

"translit" は文字書き換えプログラムの名前で, やはりこういうものがシステムに登録されているものとしてある.

```
translit a-z A-Z
```

は, 「小文字 a-z を大文字 A-Z に書き換えよ (transliterate せよ)」, という意味をもつ. これで format, Format 等はいずれも FORMAT となる. translit は find 同様, 文献[1]に紹介してある software tool で, 上のように文字の1対1の書き換えができるほか, 数字の並びを "0" 1個に置き換えるとか, 改行符号以外を全部消してしまう, とかいうようなこともできるようになっている.

(2.1) の中央のタテ棒 (|) は, その左側の操作によ

って作り出された出力（標準出力）を，入力（標準入力）として右側の操作に与えよ，という意味をもつUNIX特有の記号である．このように操作をタテ棒でつなぎ合わせたもののことを UNIX では**パイプライン**(pipeline) と呼んでいる．パイプラインでは，必ずしもタテ棒の左側がすっかり終ってから右側がはじまるというのではなく，計算資源に余裕があれば出力が左側からひとかたまり出次第，タテ棒の右側もそれを入力として進行できる範囲で進行し得る，ということになっている．すなわちタテ棒の左右はコルーチン[3]として働く可能性がある．

このパイプライン機能のようなものは，どのオペレーティングシステムにもあるわけではないが，コルーチン式に走れるかどうかを別とすれば同じようなことは作業ファイルを使うことによって，たいていのオペレーティングシステムで実現可能である．

ただしここで，translit の出力が find の入力にスムーズに接続できるようになっていないとこまる．たとえば translit の入力がレコード長 80 バイト，ブロック長 1040 バイト，出力はレコード長 135 バイト，ブロック長 1080 バイト，などとなっていて，接続にはレコード長，ブロック長が完全に一致していなければいけないとしたら，こまってしまう．文献[1]の意味での software tool の設計には，その点十分注意を払う必要がある．

2.3 プランニングへの応用：綴りあやまり検出用パイプライン

こういう意味での "software tool" を生かして使った例として，たぶんもっともあざやかなのは，文献[1]に出てくる綴りあやまり検出プログラムの話であろう（原著 126 ページ）．

英文原稿が計算機のファイルに入っているとしよう．もしその中に，辞書に載ってない単語が出てきたとしたら，それは綴りあやまりである可能性が高い．もっとも，人名，特殊な術語などもあることだから，辞書に出ていないからといって，必ず綴りあやまりだともいえない．そこでそういう怪しい単語の一覧表を打ち出して著者自身に見てもらえば，ほんとうの綴りあやまりを容易に見つけ出せるものと期待される．

もっとも，こういう方法でほんとうにいいかどうかは，使ってみなければわからない．辞書の作りかたによっては，少数の綴りあやまりが大量の，何らかの理由で辞書に載ってはいないが実は正しい単語群の中に埋もれて，とてもものの役に立たない，という可能性も十分ある．

そこで，とりあえず既製の "tool" をひょいひょいと組み合わせて「バラックセット」を作り，ようすを調べてみた，というのがこの話である．そうやってみて調子がよいようなら本物のソフトウェアを開発するし，調子がわるければ別の手を考える．(実際には調子がよかったので，さらに実験をかさねて改善したあと，本物を作ったよしである．)具体的には次のようなパイプラインが書かれた．

```
concat file1, file2, … |
translit A-Z a-z |
translit ¬a-z @n |
sort |
unique |
common -2 dictionary
```
(2.2)

これで，もとの英文原稿に入っている各単語はばらばらにほぐされて別行に並び，アルファベット順に整列させられ，(同一単語の重複が省かれた上で) 辞書とつき合わされることになる．

くわしくいうと，まず**第1行**では，別々のファイル *file1*, *file2*, …に（たとえば章ごとに）わかれて入っている原文が，ファイル接続用プログラム concat によってつなぎ合わされ，その結果が**第2行**の translit に送られる．そこで大文字 A-Z はすべて小文字 a-z に書き換えられる．その結果，たとえば文頭の "The" と文中の "the" は，いずれも "the" に統一される．(その代り IBM は ibm になってしまう．)

変換結果は**第3行**においてふたたび translit に送られ，そこで小文字 a-z 以外のもの（語の間の空白を含む）が全部改行符号（(2.2) では "@n" であらわされている）に書き換わる．(くわしくいうと，そのようにして改行符号に書き換わるものが引き続いてあらわれたときは，そのうちの一つだけが残され，他は捨てられる．translit がそのようにできている．) これで各単語は，別行に並んだことになる．

その結果は**第4行**においてアルファベット順に整列させられ (sort)，**第5行**において重複を除かれる (unique)．たとえば，"the" は原文には何百回もあらわれるかも知れないが，そのうちの1個だけが残される．そして，このようにして得られた出現単語の一覧表が，**第6行**において辞書（正しい英単語全部をアルファベット順に並べたもの．ファイル dictionary に入っているものとする）とつき合わされる (common)．ここで "−2" とあるのは common プログラムのためのオ

プション指定であって，ここでは入力(unique の出力)にあらわれ，かつ dictionary にはあらわれないような行（単語）だけを打ち出すように指示している．

なお，(2.2) のパイプラインの出力は，このままであればユーザの端末にあらわれるが，適当なファイルに書き出すよう指定することも，もちろん可能である．

(2.2)のパイプラインが，ずいぶん思い切ってぞんざいにできていることに注意しよう．たとえば全単語を“a”や“the”のような一目見て正しいことが明らかな常用語まで含めてともかく一度整列させてしまう，というのは，システム環境にもよるが，一般には相当な非能率である．また，語形変化 (file から files, filed が作られる，など）の問題も，さしあたっては無視している．

ポイントは，これならわずか 2〜3 分で作成でき，計画中の方法が役に立ちそうかどうか見当をつけるだけなら，これで十分間に合う．ということである．ソフトウェア作成過程の重要な一部であるプランニング段階は，こういうやりかたによって，ぐっと楽になる．

(2.2) の第4行にあらわれ整列プログラム sort が，行全体を見てアルファベット順に並べる，というようになっているのは，特に意図してそのように思い切って単純な仕様が選ばれているものである．translit をはじめとする各種の変換プログラムが手軽に使える状況下では，行全体のアルファベット順による整列さえできれば，(ソートマージのユーティリティにつきものの）複雑な欄指定などはできなくても十分役立つ．引数をつけることによってきめこまかくいろいろなことができるとすれば，たしかに便利でこのましいが，やさしいことをやろうというときに，まず複雑なオプション指定のやりかたをおぼえなければならないのではよろしくない．引数さえ省けば一番基本的な使いかたができる，というのがよい．このあたりの呼吸は，軽工業的ツールにとって，たいへんたいせつである．また，unique の機能にあたるものはソートマージのユーティリティにオプションとして組み込まれていることが多いが，それを独立させているのも，いかにもKernighan風である．

3. 軽工業的ツールの意義

ソフトウェアツールにおける重工業と軽工業の，もっとも本質的な相違点は何か？ ここまでくれば，この問いに答えることができる．それはソフトウェア作成過程における人の役割をどうとらえるかのちがいである．

重工業の立場に立つなら，人の介入は少なければ少ないほどよい．たとえばFortran プログラムを移植する場合，A計算機用のプログラムを投入しさえすれば，それ以上ユーザとしては何もしないでもB計算機用プログラムが出てくるような自動変換システム，といったあたりが一つの理想像となる．A，Bをパラメタとして与えることができれば，なおよい．

これに対し，軽工業的ツールでは，ソフトウェアの開発を本質的には人間の仕事ととらえ，ユーザが頭を使うことを大前提と考える．同じFortran プログラムを書き換えるにしても，まず一応 FORMAT 文を打ち出してみて，そのようすによって，つぎに打つ手をユーザに考えてもらう，というようにする．

これを要するに，ツールの導入によって人手を全面的に排除して行こう，という立場に立つのが重工業的ツール，人の介入に積極的意義をみとめ，人の思考過程をしっくりした形で助長して行こうとするのが軽工業的ツールである，といえる．

同じ軽工業の範囲内で，文型検出プログラムよりはもう少しましな道具を使うこともちろん考えられるけれども，そういうものを考えるときでも，作業を100% 自動化することはねらわず，むしろ同じ道具がFortran プログラムの変換にとどまらず，ソフトウェア作成のさまざまな局面で広く役立つことを尊ぶ．

なぜ一つの道具が広く役立つことを尊ぶか？ それも人間とのかかわり合い，ということを考えてみればよく理解できる．

道具が何千種何万種とあったとしたら，それらが個個の状況にいかにぴったりしていたとしても，ぴったりした一つを無数のものの中から選び出して生かして使うことはきわめて困難である．人の記憶力には限りがある．道具を選び，使うのが人間である以上，道具の種類数は少ない方がよい

あるいは，こうもいえよう．そもそも道具の道具たるゆえんは，人の身体（ないし頭）の延長部として役立つことである．そうなるためには道具に慣れ親しむことが必要である．そしてそれには，ある程度時間がかかる．道具があまりたくさんあっては慣れ親しんでいるいとまがない．

いずれにせよ，このように道具の数はむしろ少ないことが望ましい．とすれば一つでいろいろに使える道具の方がこのましい，ということになる．

では軽工業的ツールはなぜ必要か？

たとえば2.3節で触れたプランニングの段階は，とうてい重工業化はできない性質のものである．ああでもない，こうでもない，とやってみているうちに，一筋の光がさしてくる，というのはソフトウェアのプランニングにおいてしばしば経験するところであるが，そういう「発見のプロセス」は，現在の技術水準ではとうてい自動化できそうもない．事実，自動化はおろか，それを理解すること自体が，人工知能の第一線における未解決の難問題なのである．

だが，人間の発見過程を，道具を用意することによって助長することは今すぐでもできる．それが軽工業的ツールのめざすところである．

はじめに述べた企業情報システム作成用ソフトウェアプラントが，重工業の問題としてはSFだ，というのも同種の話である．企業情報システムを作るには，要員の心理，法律上の問題点，政治情勢，等々についての常識が必要である．そこのところを自動化するのは，発見のプロセスの自動化よりたぶん少なくともう1段階むずかしい．そこは人間にまかせ，人間の仕事をやりよくする方に専念する，というのが軽工業的ツールの立場にほかならない．

もちろん，ソフトウェア開発過程の中から自動化できるところをみつけだして自動化して行く，というのはたいせつなことである．アセンブラ，コンパイラなどはその種の自動化のみごとな例だ，といってよい．

だが，まだ客観的に理解されていないプロセスを自動化することはできない．そもそもソフトウェア開発における人間の活動を客観的に理解しようとする動きは，まだはじまったばかりである．(若干の失駆的な仕事を別とすれば) それは Weinberg の 1971 年の本[4]にはじまった，といっても過言でない．根本的な理解を欠いたままで完全自動化をくわだてると，必ずどこかに穴があく．それを何とかふさぐと，また別のところにあく．そういうふうにして無理に作ったシステムの開発者は，保守という名の穴ふさぎの際限のない繰り返しに悩まされることになろう．重工業的ツールの開発をくわだてるときは，そのことをつねに心にとめておかなければならない．

4. なぜ軽工業は注目されなかったか？

軽工業的ツール (software tool) が注目を集めるようになったのは，ごく最近のことである．これは軽工業的ツールの重要性から見て，ちょっとふしぎな感じである．なぜそういうことになったのだろうか？

二つの理由が考えられる．その第1は，比較的最近まで軽工業的ツールが，計算資源の関係上実現しにくいものであった，ということである．たとえばテキストエディタひとつとっても，ファイルのオンライン記憶と会話型端末は（なしですまぬ，というものではないにしても）ないと大変こまる．だが，その種のものは，(特にわが国では) ごく最近まで，導入不可能でないにしても，たいへん高価についていた．そのためテキストエディタを使うことなど人々の念頭にものぼらなかった，というのが実情である．その種の資源の価格が下落し，ソフトウェア開発における人的費用の比重が増したことによってはじめて，軽工業的ツールは日の目を見るに至ったのである．

理由の第2は，その効用が部外者にとってわかりにくいものであった，というところにある．おそらくこの方が，おもな理由であろう．

重工業的ツールのありがたみは理解しやすい．それまで人手でおこなわれていたひとまとまりの生産過程がばっさりと自動化されるわけだから，生産現場の実情を知らない人でもマクロ的な見かたにもとづいて，その効用を理解することができる．

では軽工業的ツールの効用は何であるか？ それは人手による生産過程が楽になる，というところにある．

もっとも原始的な例として，プログラミング過程におけるテキストエディタの利用について考えてみよう．コーディングシートに1字1字，活字体の大文字を書きつけて行く代りに端末からタッチタイピングによって入力し，打ちまちがいはテキストエディタの機能を使ってその場で修正し，またきまりきったところはわざわざ打鍵しないで既製のファイルからさっともらってくる，というようなことをすれば，手かずが減ってプログラマの仕事はたしかに楽になる。そのように身体が楽になる結果，プログラマの注意は仕事の本質的な部分にむかうようになり，より手早く，まちがいの少ない仕事ができるようになる．

さらに，「はてな，これと同じようなことを前にも書いたことがあるな．」などと気づいたとき，テキストエディタのサーチ指令を使って該当箇所を探しだしてたしかめ，矛盾をふせぐ，というようなこともできる．

テキストエディタにかぎらず，軽工業的ツールの効用は一般にこのようなものであり，そのありがたみは当事者にはただちに了解される．

こまるのは，それが当事者以外にはなかなかアッピールしない，ということである．プログラム作りの現

場を知らない部外者は，プログラマの身体を楽にしてやろう，という考えに，「あいつら，またずるけようとしていやがる．」というような，漠然とした不信感をもつかも知れない．ことに重工業的ツールの導入がオペレーティングシステムの入れ替えや端末の増設などをともなう場合，出費がはっきり目に見える一方，投資効果は間接的にしかあらわれないので，部外者の判断はことさら狂いやすいことになる．その効果が，数か月後仕上ったソフトウェアパッケージにトラブルが少ない，というような形であらわれたとしても，それがツールの導入によって起ったことなのか，それともたまたま当事者の「気合が入って」いたせいなのかの判断は，部外者にはつきにくい．いきおい，あとの方の解釈が採用されやすい．

なお，以上の所論に対しては，コーディングシートに大文字を書く作業は低賃金のコーダにまかせてしまえばよいではないか，との反論が出るかも知れない．だがコーダといえども，機械力による手助けがあればよりよい仕事ができるはずであるし，管理する側としても作業がオンライン化されていれば，より多様，精密な管理ができる．また，作業環境をおくれた状態に釘づけにしておけば，コーダのなり手は早晩なくなってしまうだろう，との心配もある．

一方，テキストエディタは，導入しさえすればよいというものではない．使いかたに注意しないと全体としての作業効率があがらないおそれもある．この点については文献[5]に記したから参照されたい．軽工業的ツールの問題は，それを使う人の問題でもあり，正しい使いかたをしなければ所期の効果は得られないのである．

5. ありあわせの道具について ——Snobol 言語

軽工業的ツールはよさそうだ，ひとつ UNIX でも買ってきて使ってみるか，というように考えるのは，たしかに一つの筋の通った考えかたであるが，とりあえず手もとに持ち合わせているものを集めて軽工業的に使ってみよう，というのも同程度に筋が通っている．というのは，軽工業的ツールでは，道具そのものもさることながら，道具を使いこなす人間の手ぎわが大いにものをいうからである．たとえば Snobol 言語[6]（ないし Snobol 処理系）など，使いかた次第でよい軽工業的ツールとして役立つ．

前出の文型検出プログラム find が手もとになかったとしよう．そのとき，もし Snobol が使えるなら

```
REPEAT
    LINE = INPUT        : F(END)
    LINE 'FORMAT'       : F(REPEAT)
    OUTPUT = LINE       : (REPEAT)
END
```

とでも書けば同じことがやれる．これだと，つぎに READ 文がさがしたい，となったら，'FORMAT' とあるところを 'READ' と変えなければならず，端末から

```
find FORMAT
```

と打つかわりに

```
find READ
```

と打つ，というのと比べるとだいぶごたごたするが，Snobol もなくて，たとえば Fortran か Cobol で書く，というのと比べればはるかに楽である．

もともと Snobol は文字列処理用の高水準言語であって，名前表を作る機能，データ構造を定義する機能，データとして読み込んだものを実行時にコンパイルする機能など，いろいろ大げさなものが用意されており，それを上のように，単に読んで，見て，書く，というように使うのは牛刀のそしりをまぬがれないのであるが，一方 Snobol には設計上独特の自然さがあり，上例はその自然さを生かしたことになっている．

もっとも上の場合には，find があれば問題なくそちらを使った方がよいが，やりたいことが複雑になってくると，いくら software tool 群がうまくできていたとしても，カバーしきれないところは出てくる．そういうとき（もし問題領域が文字データ処理を中心としたものであるなら）Snobol を使うというのは，たしかに有力な方法である．たとえば筆者自身文献[7]において，ひらかなカタカナ英語まじり文を打ち出すためのシステムのプランニングに Snobol をもちいて，たいへん有効であった．

なお Kernighan ら[1]の発想では，既製の tool によるパイプラインでカバーしきれない部分の処理は言語 Ratfor（または言語 C）でプログラムを書くことによっておこなう建前である．ただしその際，豊富なライブラリサブルーチン群を用意しておいて駆使する．作るプログラムは簡単なものでよいことが多い．既製の tool で間に合わないところだけ作り，作ったものをパイプライン機能によって，他の tool と組み合わせて使う．彼らの流儀がよいか Snobol によるのがよいかは，ときと場合による．パイプライン機能にあたるも

のが手軽に使えない環境下では，後者の方が楽，という可能性が大きい．

Snobol のほか，APL をプランニング段階の模型づくりに使った，という話はしばしば耳にする．また Kernighan & Cherry[8]は，コンパイラコンパイラを，数式清書用プログラムを作る際に軽工業的に使っている．コンパイラコンパイラが，きわめて強く重工業的な発想にもとづいて発達してきたものであることからみて，これはちょっとおもしろいことである．

なお，現行の標準 Snobol 処理系[6]には，軽工業的ツールとして使うには少々具合のわるいところがある．というのは，標準入力（INPUT 変数による）と標準出力（OUTPUT 変数による）が Fortran 流に，それぞれカード式，ラインプリンタ式となっているため，出力をまた入力として読み込むというときに，とかくごたごたする．またマクロ方式による標準処理系[9]は非常に遅い．軽工業的ツールとしては，少々の遅さは問題でないが，それにしても遅すぎる．これらの点では SITBOL[10]のような，方言による処理系の方がよい．文献[7]でも SITBOL を使った．

6. ソフトウェア軽工業におけるテキスト処理

ソフトウェア作成過程の不可分な一部でありながら，重工業的手法ではとうてい処理しきれないためにこれまで機械化の対象として考慮されることの少なかった分野の一つとして，文書の作成という問題がある．

プログラム本文も広い意味では文書である．その作成におけるテキストエディタの役割についてはすでに述べた．だがそれ以外にも，一つのソフトウェアを仕上げるまでに作らなければならない文書は，いろいろある．外部仕様書，内部構造の説明書，ユーザむけの使用手引書のほか，契約文書，会計報告書などもここに含めて考えてよいかも知れない．これらの文書は，しばしば付属文書の名をもって呼ばれるけれども，決してソフトウェアの単なる付属物ではない．実際たとえば，仕様書がしっかりしていないと虫が出る，というのは，よくきく話である．

もとより文書の作成を重工業的に処理することは不可能である．文書は本質的には人間が書くものであり，それを助ける軽工業的ツールこそ，この場にふさわしい．前述のテキストエディタはもちろんのこと，文章やプログラムの清書プログラム（テキストフォーマッタ，プリティプリンタ）なども大変役に立つ．その効用は，単に作業量が減るというだけにとどまらない．たとえば，清書プログラムを読みかえしやすくするために使うことが考えられる．というのは，あやまりのないソフトウェアを作りあげるための一つの基本的な方法は，真に納得するまで読みかえすことだからである．きれいに打ち出してやると読みかえし作業が楽になる．ソフトウェア軽工業におけるテキスト処理の重要性は，いくら強調しても強調しすぎることはあるまい．

なお，この面でのわれわれ日本人にとっての難問は，いうまでもなく日本語，ということである．現行の，薄紙に鉛筆で書いて青焼きをとる，という方式は安価である点はよいが，やはり前近代的のそしりをまぬかれない．この問題はむずかしいから，テキスト処理を既存の技術とみて，どこかに導入できる製品がないかさがす，というのでは，うまく行かないのではないか？ ソフトウェア作成過程において現在人手でおこなわれている文書作成作業を考察の出発点として，それを助長するにはどんなテキスト処理技術が必要か，またどの程度のテキスト処理技術ならがまんできるか，というふうに，軽工業的に考えて行くのが正しい道ではないだろうか？

付．討　論

本文の第1稿に対して寄せられたコメントからいくつかを拾って，その要約と，それに対する著者からの回答を示す．

コメント (Brian W. Kernighan[11])．

1．軽工業的ツール対重工業的ツールという対比から連想される類似の区別として，だれか他の人に作ってもらわなければならないツール対自分で作る気がするツール，というものが思い浮ぶ．PL/I の最適化コンパイラなどは前者に属するし，テキストエディタ，簡単なフォーマッタなどはみな後者に属する．

2．軽工業的ツールのもう一つの特色として，人々が人手ですることにあまり抵抗を感じないような作業（個数をかぞえること，比べること，探すことなど）をする，という特徴があげられるように思う．する作業の性質がそのようなものであるので，人々はとかく機械むきの仕事を人手でしながら，損をしていることに気づきさえしない．

回答．

1．たしかに PL/I の最適化コンパイラは，筋の通った重工業的ツールとして典型的といえる．一方，よ

い軽工業的ツールを設計するには自分で使ってみる心がけが必要であることにも注意したい．軽工業的ツールは助長しようとする作業に伴う思考過程を，深く直観的に把握してはじめて設計できるものだといえる．

2．その通りであり，そしてその種の作業を助成することは，プログラム作成のみならず，ソフトウェア開発のもっと一般的な諸局面で重要である．たとえばよいテキストエディタは概念設計に際して設計者の思考を助けるにも役立つ．

コメント（牛島和夫[12]）．

1．軽工業的ツールが少数であるべき理由として，もう一つPRのやさしさ，という問題があると思う．ツールは人に使ってもらわなければ意味がないが，盛沢山にあれもある，これもある，ということでは目移りがして，なかなか使う気になってもらえない．その意味でも，これというものが少数あることが望ましい．

2．軽工業的ツールの重要性がとかく見落される理由として，アカデミックな環境での問題点もあるのではないか？ 軽工業的ツールを開発しても，それを論文の形にして発表し，おおかたの評価を得るということは，風習上なかなかむずかしい．その辺にも，軽工業的ツールが普及しない原因があるのではないか？

回答．ご指摘の通りと思う．

コメント．現代は巨大なソフトウェアを組織によって製造する時代である．個人の能力にたよる考えは古きよき時代の遺物である．よってこの論文の主張は無視せざるを得ない．

回答．現代をそう規定することには必ずしも賛成できない．巨大でないソフトウェアがうまくできないためにこまっている，という話はいくらでもある．また本文で，個人の能力にたよるべきだと主張していると思われたとしたら，それは誤解である．

だが，それらの点はさて措くとしても，組織が人間からできていることを忘れてはならないと思う．組織の中の人がうまく動かなければ，組織は満足に機能しない．ソフトウェア作りに人手でやらなければならない部分があることはすでに述べた．そういう部分をうまく補助しよう，というのが軽工業的ツールの考えであり，このことは組織重視の考えと何ら矛盾しない．のみならず本文で説明した各種の軽工業的ツールは，情報をオンライン化することにより，組織の内部での人と人とのコミュニケーションを助長し，組織内の人のパーフォーマンスの測定を容易ならしめる面がある．組織が重要であればあるだけ，軽工業的ツールの重要性はますます増すのだと思う．

参 考 文 献

1) Kernighan, Brian W. & Plauger, P. J.: Software Tools, 338 p., Addison-Wesley, Reading, Mass., (1976).
2) Special Issue on UNIX Time-Sharing System, Bell System Technical Journal, Vol. 57, No. 6, Part 2, pp. 1897-2312 (July-August 1978).
3) たとえば Knuth, D. E.: Fundamental Algorithms, Second Edition, Chap. 1, Section 4.2, Addison-Wesley, Reading, Mass. (1973). 廣瀬健訳，基本算法／基本概念，サイエンス社，東京 (1978).
4) Weinberg, Gerald M.: The Psychology of Computer Programming, 288 p., Van Nostrand Reinhold, New York (1971).
5) Kimura, Izumi: On Proofreader's Programming, Report No. C-14, Departmet of Information Science, Tokyo Institute of Technology (1977).
6) Griswold, R. E., Poage, J. F. & Polonsky. I. P.: The SNOBOL 4 Programming Language, Second Edition, 256 p., Prentice-Hall, Englewood Cliffs, N. J. (1971).
7) Kimura, Izumi: Cheap Production of Japanese Documents—an Experiment in Programming Methodology, Report CMU-CS-78-130, Department of Computer Science, Carnegie-Mellon University (1978).
8) Kernighan, Brian W. & Cherry, Lorinda L.: A System for Typesetting Mathematics, Comm. ACM, Vol. 18, No. 3, pp. 151-157 (1975).
9) Griswold, R. E.: The Macro Implementation of SNOBOL 4, 310 p., W. H. Freeman, San Francisco, Calif. (1972).
10) Gimpel, James F.: SITBOL-Version % 4 B, SITBOL Project, Stevens Institute of Technology, Hoboken, N. J. (1976).
11) 私信，1979年3月12日．
12) 私的会話，1979年1月31日．

(昭和54年6月4日受付)