ソフトウェアの動向

出席者　1. 森　口　繁　一**　2. 島　内　剛　一**
3. 和　田　英　一**　4. 矢　島　敬　二**

森口　きょうの座談会の主題は“ソフトウエアの動向”ということですが，ソフトウエアという言葉自身が人によっていろいろ解釈が違うと思うので，その解釈をめぐっての話からはいってもらって，あといくつか重要な話題について突込んでみたいと思います．

ソフトウェアとは何か

和田　僕はソフトウエアというのはプログラム全部を連想するんです．ある定義によると，ソフトウエアというのはメーカーのつくったシステム・プログラムと同義であると書いてあるんですが，僕はそうは思わないんで，計算機のカナモノをハードウエアといっているのを聞いたとき，プログラムとはソフトウエアだなと考えていたので，その後ソフトウエアということばに出会ったとき，わが意を得たりとばかりに喜びました．

島内　現在僕が思っているソフトウエアというのは，メーカーのつくったシステム・プログラムのことで，ユーザーが使っているような，一つの会社で独得のプログラムなどは，ソフトウエアには入れにくいんじゃないかという気がします．

和田　そういうのは何というのですか，何かいやらしい言葉があるでしょう．アプリケーションウエアというのが．(笑)

森口　第一，ウエアというのが，いったいくっついているのか離れているのかという点はどうなの，あれは，もともとくっついている言葉だね．

和田　ハードウエアはそうですね．

森口　ソフトウエアも少なくともアメリカの雑誌などでは離れているのは見たことがないね．あれは，やはりくっついたものとして初めからできていたんじゃないのかな，語源をさぐればウエアというのはものとか何とかいう意味だったかも知れないけれども．

島内　計算機というのは，いろんなタイプがあるわけですね．ハードウエアの段階でいろいろなことをやらせているのと，それからあっさりと基本的なカナモノだけをおいてあるものとあるわけです．ところがソフトウエアという段階になると，初めて全部が同じくらいのレベルになってくる．だから，ソフトウエアとハードウエアと合わせて初めて計算機といえるものじゃないかというように思うわけです．そして，そのうえに個々のプログラムが組み立てられていくんじゃないかという気がします．

矢島　ソフトウエアの定義ということでは，島内さんの意見に近いんですけれども，多少補足すると，線形計画法のプログラム・システムというようなものも含めたものとしてシステム・プログラムというものを考えたいと思います．

森口　なるほどね．要するに，メーカーが納めるものに，カナモノとヤワモノとがあって，そのカナモノだけを納めて電子計算機が納まったといわれちゃ困るという買手が今は非常に多くなってきたからね．必ずカナモノ以外のヤワモノの部分がついていて，それもたいへん良く適合したものがくっついていて，両方合わせて，ある知的水準に達しているものでなければならない．そういうふうに，ソフトウエアを位置づけたときに，それが“ソフトウエアの動向”というようなものを論ずるときのソフトウエアの概念と合っている．そういうことらしいね．和田君もそういうように限定して進むことに別に異論はないでしょうね．

和田　そうですね．

“ソフトウエア完備”

森口　それでは，ここで論ずるソフトウエアは，そういうものであるとしましょう．

次はいったいそういうものを大きく分けるとどんなものがあるだろうかという問題です．たいていどの計算機のカタログを見てもソフトウエア完備と書いてある．われわれからみると，その“完備”というのに引

---

* Trends in software

1. Sigeiti Moriguti (University of Tokyo)
2. Goiti Simauti (Rikkyo University)
3. Eiiti Wada (Onoda Cement Co. Ltd., Data Processing Center)
4. Keiji Yajima (Institute of JUSE (Union of Japanese Scientists and Engineers))
** 1. 東京大学，2. 立教大学，3. 小野田セメント株式会社調査部，4. 日本科学技術研修所

用符がつくわけで，その“完備”ということをもってメーカーさんが何を意味しているかを問題にしたい．

たとえば EDSAC とか TAC という段階ですと，Wilkes, Wheeler, Gill の本にあるぐらいの入出力や，関数や，その他のサブルーチンと称するものがくっついてこなければ，なかなか使いはじめることさえもできない．そういうような意味においては少なくともそういう基本的なサブルーチンは，計算機と同時に納められるべきソフトウエアであることでしょうね．

それに続く時代にも，つまり，メーカーさんがつくってユーザーさんに納めるという時代にはいっても，初期のうちはけっきょくソフトウエアとして要求されたもの，また期待されたものは，そういう断片的なサブルーチンであって，そのつなぎ方がある程度標準化できていれば上々といったくらいのものであったのではなかろうか．現在でもその程度のものがついておれば，ソフトウエア“完備”とカタログに書くメーカーさんも，まだないとはいえないんじゃないかしら．

だから，その辺がソフトウエアの動向を論ずるときの出発点みたいなものじゃないかと思うんですがね．

過去の状況を振り返ったときに，いったいどういうところから出発して，どう進んできたのかという問題，あるいはどう進もうとしているのか，どう進むべきなのかということを話し合ってみてくれませんか．

和田　どうもサブルーチンの完備というのは冷房完備みたいなもので，もう何もこっちでやらなくてもすむところまできているのかというと，そうじゃないわけですね．だからあまり完備ということを信用してはいけないと思うんです．場合によっては自分でつくればいいんですからね．

矢島　冷房完備というときには，冷房機というものがあるということであるわけですね．だから，やっぱり必要最少限のものが頭の中にあって，それだけはつくってあるというように考えているんじゃないですか．

和田　必要最少限のものが完全にあるということです．

矢島　ええ，そろっているという意味です．

島内　そういうように受け取っていますかねえ．カタログにソフトウエア完備と書いてあったときに，誰もそれを本気にしやしないんじゃないかと思うんですが．

和田　僕はその列だけ飛ばして読むことにしている．(笑)

森口　しかし矢島君の規定も現実的な一つの態度だと思うんですよ．ただ，その場合に“必要最少限”ということに対する解釈は，時とともに進んでいるんじゃなかろうか．

島内　必要最少限をつくる予定があるというくらいで完備と書いちゃうんですね．

森口　それは時間軸のほうで少しずらしたところで書いてしまうからね．このカタログがお手許に届く頃には完備しているはずですという形で原稿は書かれているから．

島内　いや，そうじゃないんです．機械が納まってから一，二年経てば完備するという感じで書いてあると思うんですよ．

森口　それはとにかく，現実にメーカーさんからユーザーさんに供給されるものと，ユーザーさんがメーカーさんから供給してほしいと思うものが時とともにどう動いてきたか，これからどう動くであろうかということを少し論じてみてくれませんか．

島内　それほど自主性をもったユーザーは少ないんじゃないですか．IBM がやっているとおりにやってくれればいいというユーザーが，99.9%くらいじゃないかと思うんです．だから，ユーザーの要求というのは，IBM の機械と同じように使えるようにしてくれと，それだけのことだと思います．

和田　僕はソフトウエアの進んでいる方向というのは，ソフトウエアがだんだん性質としてはハードウエアに似てきて，一つのソフトウエアがちょっとハードウエア的になってくると，それに使われるソフトウエアができる．あのソフトウエアを処理しようという，もう少しハードのほうのソフトウエアができて，そういう何重の構造にもなってモースの硬度計で計ればいいように，硬いほうから柔らかいほうにずっと並んでいるんだと思うんですよ．そのレベルが，EDSAC でいうと，イニシャルオーダーズと，ユーザーのつくったプログラムと，それくらいしかなかったと思うんですよ．それがやれローダだとか，何かいろんな名前がありますけれども，そういったもので，次から次へいろんな硬さの，いちばん硬いのだけがハードウエアといわれるだけで，あとはソフトといいたいんですけれども，そういう硬さの順がたくさんできたと思うんですね．

で，見方によっては，今度はプログラムの定義も問題になるんですけれども，あるプログラムを機械に入れてはいったとします．そのプログラムでデータを読

みこんで何かを処理するとしますね，すると，データというのはそのプログラムをつくったときは，これはデータだと思っていますけれども，もしかしたらそのデータもプログラムかも知れない．オートマトンのいい方をすれば，インプット（入力）というのはすべてパラメーターなんです．それよりも前にはいったものが，あとにはいってきたものを処理するということでしょう．前にはいったものが，前の内部状態を変えているわけですから．そんな見方をすると，最初にモニタだとか，ローダだとか，できるだけ硬いほうがはいって，それから柔らかいほうから二番目くらいにプログラムがはいって，いちばん最後にデータがはいってくるという具合に，昔から3レベルか4レベルあったと思う．それがいまはずいぶん細かくなっているんじゃないかという気がするんです．そういう意味で僕はやっぱり，あんまりソフトウエアと，ユーザーのプログラムの区別をつけたくないんです．もっとも，そうすると困るのは，メモリプロテクトのかかっている，絶対壊われないプログラムというレベルとやっぱりソフトウエアの間でちょっと区別する必要があると思うんですけれども．メーカーがつくったシステム・プログラムが全部メモリプロテクトされるかというと，必ずしもそうじゃないかも知れない，逆にユーザーがあとでつくったプログラムを，メモリプロテクト領域に入れても悪くはないかも知れないですね．見方によると，ソフトウエアとハードウエアの間にミディアムウエアくらいのものがたくさんできているんじゃないかと思うんですがね．

森口　和田さんのいった，たくさんの層があるというのを例示すると，こういうことになるわけです．たとえば，僕らはもうアルゴルとか，フォートランという言語で計算機を使うのが普通の使い方だと思っている．計算機というものは，数式を受け付けるものであり，if とか then とかいうような言葉を解するものであるわけで，僕は英語と数式を使ってプログラムを書き下して，そのあとにデータを少数または多数くっつけて送りこみさえすれば，あとはもうソフトウエアとハードウエアと合わせたものが処理をして，僕は結果を受取ることができる．そういうようにできているわけです．その内部では実はどういうことをしているかというと，初めはコンパイラーというものが働いて，僕が書いたプログラムを自分の言葉に近い水準に落す，それから今度はローダが働く，そしてほんとうに自分の言葉に直して命令文を自分なりにつくり出す．そして今度はそれがデータを受け入れて処理をして出すという働きをする，その間にも手足のようなサブルーチンというのはずいぶん働かされているわけです．そして最後にサブルーチンでやってもよし，カナモノでやってもよいという仕事を，相当程度カナモノが引き受けてやってるのが現在の進んだ計算機の実情である．もうその辺になってくると，先ほど島内君がいったカナモノでやるか，ソフトウエアでやるかということは，僕にとってはたいした問題ではないわけですね．要するに経済性が総合的にいってどうであるかということが意味があるわけである．

和田　たとえば，トラップ（割込み）のコントロールルーチンは非常に硬いですね．

森口　いったいトラップのコントロールというのは，そもそもハードウエアで行なわれたことがあるのかしら．あれは初めから硬いソフトウエアとしてスタートしたものかしら．

島内　ハードウエアが先じゃないですか，そしてストレッチあたりが今度プログラムでインタラプトを完全に処理するように，なった……．

森口　それ以前に，そうすると，かなり念入りなトラップ後の処理をするハードウエアが存在したと……

島内　と思いますけれども．

和田　だけど，トラップをハードだけで処理するということはできないでしょう．

島内　ええ，完全にはね．

和田　だから，やっぱり，どこかでプログラムが関与するわけですね，もちろん全部プログラムでは処理できないでしょう．だから，どの程度ハードとソフトで，そのロードを分かち持つかというところに問題があるわけですね．

森口　なるほど，そのトラップとか，インターラプションとかいわれることの歴史というと大げさだけれども，発展してきたいきさつは，それはそれでまとめてもらうとおもしろいかも知れないね．

さて，ソフトウエアの動向を論ずる際に，もうちょっと伺っておきたいのは，アルゴルとかフォートランとかいうコンパイラー，これは現在では少なくとも科学技術用の計算機を売りこむ場合には必須のものと，売るほうも買うほうも思っているんじゃないかしら．

それが僕の感じでは，そういうコンパイラーを使うことができる機種を持っている，あるいはコンパイラーを使っているユーザーと，コンパイラーというものの味を知らない，現に使おうと思っても使えないユー

ザーの間には，かなり大きい不連続があるような気がするけれども，それは一種の時代区分かしら．

矢島　別の面からみると，計算機を使うということの内容が変わってきていると思うんです．EDSAC 流のやり方でやっていたときにはプログラムを書いた人は必ず機械をいじったと思うんですが，現在の段階ではもちろん機械はいじったことがないし，機械を見たこともなくて計算機を使う人がふえてきている．そういうような形で，ソフトウエアは動いてきたと思うんですけれども．

森口　そういう非常に柔らかいところまで含めてのソフトウエアまでを計算機の側にくっつけて，そういうものを自分が使うんだという意識で，アルゴルやフォートランでプログラムを書いて使っている人たちと，それから，ソフトウエアのところは，プログラムの部分なんだという意識で，カナモノとソフトウエアとの境目のほうにかなり注意が向いていて，そういうところで仕事をしている人と，人数の比率は前者のほうがずっと多くなるんですけれども，そういう一種の分極作用が行なわれてきたというのが動向の一つでしょうね．

和田　それはそうですね．たとえば，IBM のマニュアルをみましても，アプリケーション・プログラムとシステム・プログラムの二つがありますね．システム・プログラムマーズ・マニュアルというのは，ユーザーの中の システム・プログラマーの マニュアル です．

矢島　それからたとえば線形計画法で問題を解くという場合に，計算機を使う人は答が出さえすればいいんだけれども，そのデータを実際にどういう具合に計算機にかけるかはわからない．そこで，データの順序はこういうようにしなさいというような人が必要になる．つまり計算機の側に人間がはいっている．

和田　それは変な話だな．人のグループがだんだんふえればふえるほど，平均のレベルというのは下がっていきますね．で，ほんとうに勉強している人というのは，あんまり多くないわけですから，どうしても非常にイージーに人に聞くような程度のレベルの人が最後にはだんだんふえてくるようになってくるわけでしょう．

それは何でもそうなんであって，ふえれば質が低下する．そのことじゃないでしょうか．別ないい方をすれば，計算機の人口がふえたということですね．

システム・プログラマーの教育

森口　そういう人でも使えるように進歩してきたともいえないかな．それはそのくらいにして，先ほどの話の分極の結果，システム・プログラムをいじることになった人たちというのは，特殊の専門的なプログラマーということになると思うんです．そういう人たちの教育とか，指導の仕方は一般のコンパイラー言語で使おうという人たちの教育の仕方とは，おのずから違ったものがあると思うんですよ．

それについて島内さんが常々考えておられる教育の基本方針の一端を披瀝していただけませんか．

島内　僕がいつも考えたり，人にいったりしているのはそういうようなシステム・プログラムをつくるような立場に立ったら**よく考えろ**ということです．それだけなんですけれども，要するに思いついたままサッとプログラムなんかするな，少なくとも二つくらいプログラムしてみて，どっちがいいか優劣を確かめたうえでいいほうをとれと．いつもそれをやっていれば，だんだんサッといいほうが出てくる，そうすれば，将来は一べんにいいプログラムができるから，能率はどんどん上がってくるわけです．そして，そのくらいになって両方を比べてみるとなおいいものが出てくる．ですから自分自身を教育しながら仕事を進めていけということです．

森口　なるほどね．

和田　それだけ．

島内　それだけです．

和田　島内文部大臣の教育基本方針は前文だけで終った感じですね．

森口　しかし非常に重要な前文だね．その点今のメーカーさんで，そういうカナモノにつけて出すようなヤワモノをつくっている人たちが，そういう態度で仕事をするにはあまりにも納期が迫られているとか，そういうような気の毒な事情があるんじゃないですか．

もうちょっとスケジュールに余裕をもって仕事を進めてもらったほうが，よく考えて，少なくとも二つくらいの考えの中のいいほうをとるというような態度で仕事を進めることができると，そういうことを感じませんか．

島内　それはプログラマーを管理している人の考え方一つじゃないかと思うんですよ．たとえば，1年かかる仕事があったとします，その場合にプログラマーを管理するほうの人が，そういうような考えでやって

いると，遅くなるから，要するに拙速でいいからどんどんやれというような態度でやると，それは1年の納期で間に合うかも知れませんけれども，その間プログラマーは，ほとんどレベルが上がらないわけなんですね．ところが僕がさっきいったような態度でやれば初めのうちは得るところは少ないかも知れないけれども，だんだんこう配が急になってきて，けっきょく1年後くらいには追いついて，そして次の1年くらいのときには，もうはるかに引き離してしまうだろうと，そういう意味でほんのちょっとでいいから最初は余裕をくれればいいんじゃないかと思います．

森口　そうね．工程管理している人の計画の立て方としては初めゆっくりであとほどピッチが上がるような，そういうような計画にすべきでしょうね

和田　僕は島内さんの基本方針に大いに賛成ですけど，もう一つコメントをつけたいと思うことがあるのです．何もソフトウエアにかぎらずすべてのプログラムについていえるわけですけれども，人間とのコネクションが非常にうまくならなければいけないということです．ルーチンワークで会社が動かしている仕事というのは，オペレーターのことを考えてやらなければならないんです．特にそれが重要なのはシステム・プログラムだと思うんです．それがうまくできてませんと，人間工学的なことを考えていないと，けっきょくつくってもおシャカになってしまうと思います．

それはカナモノについても全く同じことがいえます．ですから島内さんの“考えろ”といったときには僕は人間の間違いやすさだとか，いろんな癖だとかいったものも十分考えてほしいと思います．

余談ですが，僕は久しぶりに PC-1 のテープを編集するのをやってきたわけですが，昔テープを編集するのにトンツーの電鍵を使っていたわけですね．あれは非常にうまくいったんですが，きょう行ったら故障していてだめなんですよ．で，あれは横にタイトテープレバーというのですか，テープがぴんと張るととまるやつ，それを動かさないとだめなんです．それで最後には1字ずつ送れるようになりましたけれども，あれだけ長いストロークを動かして1字ずつ送るというのは，よほど運動神経がなければだめなんですよ．電鍵でトントンとやるのは非常にうまくいくんです．昔は高橋研では電鍵がこわれていたら，皆いらいらして直したんですが，そういう人間とのマッチングを非常によく考えないと，毎日使うツールというのはつまんないことだと思うんですよね．

森口　ソフトウエアで，それに似た例を指摘してくれませんか．

矢島　磁気テープを取りはずしたり，つけたりの動作がうまくいくというようなことですか．

和田　まあ，そういうこともありますし，もっと簡単なのでいえば，十進法の数を読み込むルーチンなんかが必ず一つの数字は十桁打たなければならないというのがありましたね，あんなのは最初のいらない0は落してもいいとか，何とか，できるだけ使い良いようなことを片端から考える必要があると思うのです．もっとも，あまりそれをやりすぎると，エラーがみつからないという事態が往々にして出るんですけれども，そこが今度は人間の間違いやすさをいろいろ分析して調べる必要がありますね．

だから，システム・プログラムというのをやっていると，心理学をやっているようなときもありますね．

それで僕，何かに書いたんですが，いつか，IBM Review で高橋先生が“計算機をやってほんとうに人間のことがわかった”といわれたことがあるんですけれども，僕もそうなんです．計算機をいじるようになってから人間というのはおもしろいなと思うようになった．要するに万学の基礎は人間を知ることですね，というといけないかな．（笑）自然を知ることかな．だけど，それを知るめがねとしては計算機，特にシステム・プログラムというのは非常に有効なツールだと思うんです．

基本的手法

森口　ところで話題をちょっと変えて，島内さん，いろんなシステム・プログラムの種類や程度があるわけですけれども，そこで使われているプログラミングの手法というのは，ごく限られた2種類か3種類のものにつきるというようなことをいつかいっておられたことがあるでしょう．

島内　スタック（stack）を使う話と，チェーン（chain）をつくることと，それからファン（fan）の構造，またはリスト（list）の構造ですね．僕はファンというほうが好きなんですけれども．

森口　そのチェーンをつくるというのは，リスト構造の単純な一類型とみることができるわけね．

島内　ええ．

森口　しかし，リストにはできるようなものでも，チェーンはつくることができないということもあるから，一応は区別したというわけでしょう．

スタックは僕は“棚上げ”にたとえたことがあるんですけども，要するに仕事がたまってきたときにどんどん上へのっけていく，それから処理するときは，それを上から順々に降ろし，最初に置いたものが最後に出てくる．棚の上に置くようなものがスタックというんですが，それに対してチェーンは“数珠つなぎ”，あるいは“芋づる式”といいますか，芋づるをあとからたぐっていって，最後のところまで行けば，これですんだということになる．浮動番地を処理するルーチンでよく使われるものです．

スタックはアルゴルのコンパイラーをつくるときには非常に便利なものですね．それに対して，リスト構造（本誌349ページ参照）というのは，どういう意味でそういう有力な道具なのかということがちょっとピンとこないんじゃないかと思うんですけれども，それを簡単な例で説明することはできませんか．

島内　スタックというのは僕の考えでは，リストというか，ファンの構造と，そうじゃない構造との翻訳をやるものだと思う．だから，スタックが使われているときには必ずなんらかのかっこうでファン構造（扇子のような枝分れ）が使われている．けっきょくは同じ一つなのかもしれない．

森口　ということは迷路学習で，“いつでも左手を壁に沿ってずっと迷路を行きなさい，行き止まりになったら帰ってきなさい．そして枝わかれの所ができたらいちばん左の枝からはいっていきなさい”．それだけのルールで完全に学習できるわけね．そういうような構造を頭に描いているわけですね．

そういうものを処理するときには，どうしても未処理のものをスタックに積んでおくということになるから，そういう意味で構造が今の末広，枝分れ方式である場合には，棚上げ方式になるというわけですか．

島内　ええ．

和田　要するにロジック（論理）がレカーシブ（recursive，再帰的）だということなんでしょう，レカーシブなロジックのときに，それが必要になるんじゃないかと思うんです．

森口　なるほどね．非常に複雑な構造なんだけれども，処理の原理はごく単純な，一般法則だけですんでいる．それは非常に複雑そうにみえる，システム・プログラムを理解するうえに重要な原理ですね．

さて，そこで次にそういうシステム・プログラマーたちがいろいろと努力をしてくれるお陰で，一般の利用者は次第に高級な言語を使えるようになってきたわけですけれども，そのことが，プログラムの誤りを発見して直すという場面にも当然反映していると思うんですよ．

過去を振り返ってみて，どういうように直し方が変わってきたかというようなことを回顧してみて，これからはどんなようになって行くだろうとか，行くべきだろうというようなことを少し話し合ってみてくれませんか．皮切りに矢島さんどうですか．

プログラムの直し方

矢島　異論があると思うんですけれども，普段考えていることを述べてみたいと思います．

ちょっと話がずれるんですが，プログラムの検査ということの中にある問題として，そろそろプログラムを検査する組織というものを考えていかなければならないんじゃないかと考えているわけです．特にシステム・プログラムというようなものを作り上げたときに，それのでき具合を調べる機能というものは，つくった人と別の機能で行なわれるべきじゃないかと思うわけです．で，そのためには普通の品物をつくった場合の検査機構というのと同じような形でやったらいいんじゃないかと思うんです．仕様を決める段階からその検査をやる組織というものが加わって，最終製品についてつくった側とは独立に，いろいろなテスト・プログラムをかけて検査するというようなことをやっていかなければいけないんじゃないかと思います．それはいくつかのコンパイラーを使ったり，あるいは，つくっていく過程を見て感ずることなんですけれども，それが一つ．

それから今の話にもどって，コンパイラーを使う段階にはいってから，検査というのがどういうように変化したかということを考えてみると特徴的なことは，検査のスピードが向上したということです．人間がおかす誤りというものが，働いた時間に比例するというような面があります．そこで，コンパイラーを使うプログラムに要する時間が減り，それによって誤りが減っているわけですから，その分だけスピードが向上したんじゃないかと思います．

森口　誤りが時間に比例するというのは，たとえば10分に一つくらいですか，30分に一つ．

矢島　適当な大きさのプログラム用紙に一つか二つ．

森口　そうすると，30枚書かなければならなかった頃は，一つのプログラムに30個の誤りがあったのに，

2枚ですむことになれば二つですむと，そういう意味でけた違いに誤りの数が少ないわけね．それはやっぱり相当重要なポイントでしょうね．

矢島　それからプログラムの検査の能力が非常にふえているということですね．つまり機械語で書いていたときにはプログラムを書いた以外の人が調べるときに非常に時間がかかったわけです．それは理解するのに手間どってしまっていたからだと思うんですけれども，そういう点で手工業の段階だったのが，いわばプログラムがかなり互換性をもってきたことによって，別の人が読むときに読みやすくなったということですね．

それからもう一つの特徴的なことは，数値計算の場合だと思うんですが，プログラム自身の完成というのが実質上の主要部分であったのが，数値例の検討をやって，正常なものが解けるということだけではなくて，多少その問題を広げてみて検討するというようなことが簡単にできるようになってきているということです．それによって検査の機能というのが深く広くなってきたことです．

和田　そうですね，僕はこう思いますけれども，たとえば，コンパイラーを使うことによってエラーが減ったということは，フレキシビリティを犠牲にしてエラーを少なくしていると思いますね．だから矢島さんがいわれるように，時間が短くなるというファクターのほかに，それがあると思うんです．それから，シンボリックにすることによってエラーが減りましたね，それは，レダンダンシーを入れることによって減らしたと思いますね．アセンブリーとコンパイレーションとわざわざ分けるのはそういう意味では意義があるような気がします．物の進歩というのは，やはりパターンを認めることだと思いますね．それでプログラムをいくつもいくつも書いているうちに，いろんなレベルのパターンをみつけて，そういうパターンがあるならこういう使い方をしようと考えて，たとえばコンパイラーみたいなものができたと思うんですね．あれは一つの人間とのコミュニケーションということもあるわけですけれども，それ以外にも DO のループというのは，一つのパターンをみつけたということですね．もっとも，ハードのレベルには，インデックス・レジスターをつけたということがあるのですけれども．いろいろなレベルでやっぱり，ソフトウエアの進歩というのは，いろんなパターンを認識してきたことだと思いますね．

森口　やっぱり物事を覚えるときにそういう型みたいなものを先人がみつけて組織しておいてくれれば，あとから習うものはたいへん楽になる．しかもそれをカノモノなり，ヤワモノなりとして形成してしまえば，われわれはただそれを使うだけで，それだけの複雑なことをやってしまうことができるというようにもなるわけですね．だから，パターンを認識して，そして，なるべくならば人間の外に持って行くということによって，プログラミングをうんと楽にして誤りの数を減らしてきたと，そういうことを強調されているとみていいわけだな．人間はやっぱり，なるべく不精をするように心がけたほうが進歩するんだね．

和田　不精をするためにはちょっと努力をしなければならない，その上でもっと不精をする．

ところで僕は元来過度のフレキシビリティ(融通性)というのは有害無益だと思うんですけど，ある程度のフレキシビリティは，変化する事態に対応するためには必要なことじゃないか．それがいまのコンパイラー・ランゲージにも見られることを示唆したように思うんだけれども．

森口　それを一つでいいから例示してくれませんか．

和田　メモリをたくさん使ってもいい，時間もうんとかかってもいいといえばコンパイラーで何でもできるわけですよ．要するに全部ビットのパターンか何かの1と0だけとる，あれだと思ってプログラムを書けばいいわけですからね．そういう意味では困らないんですが，だけど，やっぱりメモリに制限があるし，スピードも速くしたいというときに，やっぱり，コンパイラーを使いたくないときがありますね．実際システム・プログラム自体をコンパイラーで書くというのは，だいぶ冒険だと思うんです．できないとは思いませんけれども．ネリヤックは（NELIAC）これを企てているわけなんです．

森口　もっとも，ネリヤックは有力らしくって，この間バークレイのカリフォルニア大学で聞いたのでは，アルゴル・コンパイラーをネリヤックで書いてつくった．それがとてもよくできていて，IBM 7094 で使っているんだけれだも，普通の数値解析の実習問題を学生がやるのは平均1秒だそうです．だから，300人くらいがそのコースをとっているんだそうだけれども，その連中が出してきた出題をパッと処理するのが300秒ですんでしまう．普通のフォートラン・コンパイラーでは，長さ0のプログラムでも 25 秒かかるん

だそうですね，だから，ネリヤックで書いたアルゴル・コンパイラーが，普通のフォートランのコンパイラーに比べると，うんと早いということは，それでいえているわけでしょう．手品の種は簡単なんで，要するに 32 キロのメモリーの中に，そのコンパイラーを入れ放しにしておいて，プログラムは十分小さいからあいているところでさっとつくって，作業用番地もその残りで済んでしまうといういい条件ばかりそろっているわけだけれども，その際に，それくらいの実績を挙げられるということは，つまり機械さえ十分高性能で高容量があれば，コンパイラーランゲージみたいなもので書かれたコンパイラーが，必ずしも非実用的なほど遅くはないという一つの例にはなっているわけだね．

和田　僕はそれは考えによって，まだ進展があると思うんですけれども，何かコンパイラー一つ，一つを機械語に対応させる部分というのがありましたね．そこのつくり方がずいぶんものをいうと思うんですね．

そこに少し秘密があると思うんです．

森口　なるほど．島内さん何か．

検査部門と製造部門

島内　さっき，矢島君がいちばん最初に，メーカーがソフトウエアをつくるときに，その検査の機構をつくらなくちゃというようなことをおっしゃっていましたけれども，あれに少し異論があるんです．そういうようなことをやりますと，進歩が止まるんじゃないかという気がするんです．だいたい検査部門とか販売部門というのは，製造部門に比べて進歩がずっと遅れている．要するに勉強不足なわけです．何か進歩的なことをいっていても，それが実際に自分のものになっていなくて，ちょっと聞きかじりみたいなことをいっているにすぎない．その検査のほうから製造のほうに圧力がかかるというのは，ちょっと問題だと思う．その場合に，製造のほうが検査のほうを一生懸命教育したりすると，今度は教育のほうの時間のために製造が遅れてくる．

ソフトウエアというのは，ある程度まとまったものは一人でつくらなくちゃいけないと思う．たくさんの人が相談してやっていると，けっきょくその相談する時間または当事者を教育するのにばかり時間をとられて，実際につくる時間やものを考える時間が少なくなってくる．けっきょくは一人が中心になって全部をつくらなくちゃ，ソフトウエアというのはつくれないと思う．大勢かかればかかるほどスピードが落ちてくる，そういう意味で検査なんていうのは，できたあとでやるのはいいんですけれども，つくっている最中から横から口出しはしないほうがいいと思う．

それからそういうようにしてできたら，すぐに決定版にしないことが必要だと思います．ある程度使ってもらって，二，三カ月おいて，そしてエラーを全部はじき出してもらってそしてもう一度つくり直して決定版にする．これが大事じゃないかと思います．

森口　矢島君もそういう意味だろう．

矢島　そうなんです．ただ，確かにいまいわれた点も当っていることがあると思うんです．それは新しい製品が開発されたときには，検査なんていうのはできないということが一つ，それから非常に先端の領域のところでは，だいたい検査と製作というような機能を分けることは不可能であるし，それを分けたら能率が落ちるということも確かにあると思います．しかし，たとえば関数ルーチンなどが何カ月か使ってから間違いが発見されたということでは困るので，やっぱり検査をしておかなければいけない．そういうものを置かなくても製作者によって行なわれていればいい．だけど組織的にみると，そういうことを同じ人間に課して，同じことができているかどうかをチェックするというのはむずかしい面があるんですね．だから，何らかの形で別の人間が検査をやるというような形にしてしまったほうがいい面があると思います．

もっとも，そういうような検査は，開発的なところではいらないし，実際上できない面はあると思うんですけれども，できる部分がかなり大きくなっているような気がするんです．

島内　製造のほうから頼むというようなかっこうでの検査は必要だろうと思う．だけど，製造している人がほかの人に頼んで検査してもらうというのは，製造の一部じゃないかと思うんです．流れ作業にのっかっている製造と，別にある検査とは違うんじゃないかという気がするんです．

和田　検査っていろいろあるわけでしょう．関数ルーチンを自分でコーディングして機械にかけてみて，最初の答から人に検査を頼むわけじゃないでしょう．やっぱり最初のうちは自分で調べるわけですね．

それから僕たちのほうは事務の計算でも，一応依頼があるとおりの計算をしてみて，そのとおりになったかどうかまではこっちで調べます．それを依頼したほうへ返してこれでいいかというと，何度も何度も文句

がついてくる．

そういう意味じゃたいていの機械はそういうように検査の場所というのは別にあるような気がします．

やっぱり，つくった人は自分である程度までは調べなくちゃだめだと思う．

森口　工程内検査というものに対して島内さんが反発する気持もわかるんだ．それは近代的な品質管理が導入される以前の検査部門というのは，いやに威張っていて，ろくろくものがわからない癖に変なところに意地悪で，製造部門をいじめることばかりやっている．あるいは，そういうところのご機嫌をとっていれば大目に見てくれるといったような，非常にたちの悪い存在であった傾きがあった．そういうところが強かった会社というのは，新しい品質管理は現になかなかはいっていないんですね．

そういうのでなくて，むしろ，検査部門というのは工程の状況を把握するための情報をとる感覚器官なんだと，それを工程に正しくフィードバックして初めて工程を良い状態に維持できるし，ときには改善も可能になるんだというような解釈が，近代的品質管理における検査の職能である．だから，そういう位置づけにおいて，製造部門が工程内でやるのよりはもっと組織的に，もっと根本的に検査することができるような検査の方法をもつ．そういう意味の検査機能は，やはり考慮してもいいんじゃないかという気がするね．

だけど，たとえば関数ルーチンにしても，製造のほうでは若干のキーになる値に対して標準的なくわしい数表と照し合わせて，精度いっぱい合っていればまあ合格にして渡す．そうすると検査部門のほうでは，それをたとえば百分の一きざみとか千分の一きざみで系統的に計算させてみて，そして誤差曲線を実際に書いてみるというようなことをやって，それを添えて検査成績書として納めるといったようなことをやればね．

島内　ちょっとそれに対して異論があるんですけれども，ソフトウエアというのは一つ，一つ別なものをつくっているはずですね．関数ルーチンにしても，たとえば，サインの関数ルーチンをつくったら，もうサインはおしまい，次はタンジェントが出てくるというように，違う製品が出てくるわけです．

同じ製品が出てくるならその製品ごとに検査の方法を決めればいいんですけれども，製品は1回，1回，一つ，一つ違うものが出てくるわけです．

いまの百分の一きざみにやったときにそれでいいかというと，たとえば$\frac{\pi}{2}$なんかのときにまずいことが起こると，それが百分の一きざみのところにひっかからないかも知れない，そういう危いところが数カ所にあると，そういうところを検査しなければならない，ところが，どこが悪いか見きわめることができるのは，つくった人しかいないと思います．

森口　そうなんだ，それが非常に微妙なんで，つくった人しかいないというような解釈もできるが，反面今度はつくった人は案外そういうところが盲点にはいってしまって，自分では気がつかない，おか目八目で，ほかの人が見ると，早く気がつくということもあるわけね．

島内　だから組織的な検査じゃなくて，検査の専門家がいて全部プログラムを自分のものにするんだというつもりで．

森口　そうすると内科的な検査だけじゃなくて，やっぱり外科，場合によっては解剖までも．

島内　ええ，そこまでやらないと，ソフトウエアの検査はできないと思う．

矢島　検査に関する知識が蓄積されていくべきだという感じがするんです．これだけはやったほうがいいというようなことを，ばらばらの人がばらばらに知っているんじゃなくて，誰かのところに集まっていて，知識として蓄積されていく機能がほしいわけなんです．

森口　機械設計なんかをやっている製図室には，やかましいベテランがいて，書いてきた図を眺めて，ここがだめだとかいって検査をして指導をしているわけね．そういうような意味のフォアマン（職長）みたいな人がいて育ててくれれば，それでいいかもしれないという気がするね．

だけど，確かにそういう検査という職能が非常に大事なものであって，かつ，そういうふうに経験を蓄積することによって一般の能力を高めることができるんだということを指摘しておくのは，意義があるかもしれないな．

和田　デバギングのやり方ですが，さっき，システム・プログラムというのは人間とのコミュニケーションが非常によくなければならないということをいいましたけれども，今それは割合に悪いんですよ．

どうしてかというと，機械が大きくなればなるほどクローズド・ショップ制（閉鎖式）になり，何時間かのターン・アラウンド・タイムで戻ってくると，それは場合によって長くなるわけですね．みんなが殺到すると，なかなか返ってこない．しかも機械のほうはま

たターン・アラウンド・タイムが長いために，少しくらいのエラーは仮定してやるわけですね．

その仮定が間違っていると，うしろまでやってもけっきょく無意味なんです．で，MIT でやっているような流儀が，これからだんだん重要になってくるんじゃないかと思うんです．要するにデバッギングのレベルではなるべくタイム・シェア（時分割）で，プログラムを入れると，すぐその場でエラーをみつけて教えてくれる．返ってきたらすぐその場で直してまたつっこむというようなふうに，ここ二三年の間に大きい機械はいくんじゃないかという感じがしているんです．トラップがついていれば，昔は機械の前にへばりついて，ワンステップ，ワンステップ，プログラムを追っていったことがありましたが，もしかしたら，またああいうふうにやってもかまわないくらいなものですね．あるいは，フォートランみたいなんだったら，ステートメント一つごとにワンステップ進んでいくという方向になるかもしれませんけれども，なるべくターン・アラウンド・タイムを短くしてすぐ直せるような方法をシステム・プログラムとして用意していく必要があると思うんです．

森口　次の時代のためにね．

和田　ええ．

森口　ところで僕は誤解がないためにもうちょっと古い時代の変遷を概括しておいたほうがいいと思います．まず最初は機械語でかけて通らない場合には，機械の傍でボタンを押してみるよりほかなかった時代があった．それでもやがてポスト・モーテム（検視）だとか，トレーサー（追跡）というようなもので，その過程はもっと早くできるんだということがわかってきた．しかし，そのトレーサーの使い方さえもあんまりよく心得ないで，いつでもボタンを押しているというプログラマーが今日でもいるんですよ，つまり，そういう進歩についていけない，ごく原始的な状態です．

次の段階はもうトレーサーさえも遅いと思うほど機械が速くなっちゃって，その反面ラインプリンターがついたりした関係上，メモリダンプ（記憶ぶちまけ）がものすごく速くとれてしまう．そういう段階でそのメモリーダンプがデバッキングの有力な手段であった時期がある．現在でもものによっては，そういうやり方をしているかと思うんだけれども．そして，コンパイラーの中には，そういう機能がデバッギング用のものとして含まれているものもありますね．そういうような機械を使う場合のプログラマーの心得というのは，機械の傍にいかないで，どうやってデバッギングをするかということにおもなポイントがあったと思うんですよ．

そういう機械を使ってるのに，そのもう一つ前の段階のデバッギング手段しか知らない，つまり，機械の傍に行ってボタンを押さなければどうしても欠点がわからないというプログラマーはまず失格でしょう．

だから，そのことはやっぱりひとまずはっきりといっておいて，現在そういうメモリーダンプが楽に使える機械，あるいは記号言語によって誤りの頻度もたいへん少なくなった段階で，機械の傍でボタンを押さなければ，デバッギングできないというのは，ごく特殊な基本的なシステムプログラムを開発しているプログラマーは別として，それ以外の一般のユーザーにとっては，恥以外の何物でもないということは良く認識してもらわなければならないと思うんですよ．その教育が行なわれないか，ないしは不徹底であると，これはたいへん有害なプログラマー教育になるのではないか，それが現状だと思うんですよ．

和田　現状はそうですけど，僕はダンプをとるにも実は異論があります，というのはダンプ必ずしも万能じゃないんですね，どうして万能じゃないかといいますと，昔みたいにレジスターの数が非常に少なくて，メモリーが多ければ割合とダンプというのは有効だったと思うんです．情報はほとんどメモリーにしまっていました．けれども，いまのようにインデックスレジスタがあると，メモリーダンプにはあまり情報がでてこない．それよりもどこをとればエラーがわかるかということを早く知ることのほうがむしろ重要だと思うんです．ところが，そういう訓練をするチャンスがあまりないのです．

最近プログラムを書いている人というのはあまりデバッギングが上手でないんですね．どこをおさえればいいかというのがなかなかわからない．一歩一歩やるほうでうまくいくかどうかわかりませんけれども，何かもう少し別なトレーニングというのが必要な気がしますね．

ですから逆説的ないい方をすると，コンパイラーみたいに便利なものができたために，機械は完全にロジカルに動いているんですけれども，ロジックを追跡していく能力がかえって落ちてきた．しかも，ダンプをとったときに，ダンプを唯とっても困るのは，フォートランで書いたものはメモリーの対応というのはあまりよくわかっていない．それで役に立たないダンプを

とっちゃう，それをけっきょく捨てるということになるわけですね．その辺はもう少し考慮する必要があると思う．たとえばダンプもフォートランで使ったソースのシンボルといっしょにダンプされる，そういうことでなきゃあんまりいまのダンプは役に立たないと思いますね．

森口　そういえば普通のアルゴルやフォートランでプログラムを書いて使う人は，ダンプなんていうのはあんまり必要ないかも知れないね．

むしろ途中に手掛りになるような情報をうまく出すように，プログラムを書くということのほうが大事な心掛けなんだな．

和田　まあそうですね．要するにプログラムを書いたらまず間違っていると思うということをいわないといけない．

森口　だから，たとえば級数を計算して，級数の和を求めるというときにも，答えの和だけを出すんじゃなくて，途中である程度のものは出すとかね．少なくとも項数ぐらいは出しておくとか，そういうような配慮をいつもするように教育することのほうが大事かも知れない．つまり，プログラムの中にそういうチェックを仕込んでおく．本番で早く走らせたいところは，そういうところをとってしまえばいい．

矢島　検査に対する基礎知識の量は，コンパイラーを使うほうが，機械語でやっていたときより大きくなってしまったわけですね．つまり，機械語で書いていたときには，機械語の機能は薄いマニュアルを見ればわかったわけだけれども，コンパイラーの基礎知識としてもっているべきものの量が，多くなっているんじゃないですか．

森口　その辺は確かに微妙な問題だね，つまり，マニュアルに書いてない変なことに関する知識が必要になるような場面では，前よりもっと問題がむずかしくなってきている．ところが，マニュアルに書いてあることだったら，まあうまくいくわけだし，そういう形で使っているかぎりにおいては，もうそういう知識も全く必要を生じないというようなことになる．

だから，やはり一種の分極作用があって，普通の人はそんな立ち入った知識はいっさいいらないで，いざそれがトラブルを起こしたというときの対策は，やはりシステム・プログラムの専門家でないとどうにもならないということになってきているのでしょう．

事務計算

さて，そこで話題を変えて，事務計算の話に移りたいんですが，先ほど和田さんは，パターンを見抜くのが進歩の定石だというような意味のことをいわれましたが，事務計算をいろいろ手がけていて，それに共通なパターンといったようなものが見つかっているんだろうと思いますが，そういうのを一つ話してみてくれませんか．

和田　やっぱりいろんなレベルのパターンがあるわけですね．いちばん簡単な分け方をすると，給料計算だとか，そういう分け方．

森口　給料計算とか，株式の配当金の計算だとか，在庫管理の計算だとかいう分け方ね．

そういうパターンの分け方は，プログラマーにとって助けになりますか．

和田　今のところはなっていないんじゃないかと思うんです．だけど，やっぱり，そういうようにパターンを初めから決めてしまうと，また進歩もなくなってしまうかも知れないんですけれども，やっぱり，何となく今やっているのは昔の PCS（パンチ・カード・システム）でやっているのと感じは似ていますね．もうちょっと複雑になっているとは思いますけれども．

サマリーカットというのが，しょっちゅう出てきましてね．けっきょくそんなのが COBOL EXTENDED のリポートライターのコントロールの切れるところだろうと思うんです．

ああいうのはデータープロセシングで，やっぱり，かなり本質的なことかも知れませんね．

要するに多いものをグループに分けて，そのグループの数がどうだったとかいうことですから．

森口　分類，集計というようにいっているね．

和田　まあ，そうですね．

島内　何か，事務計算というのはディメンションが，それぞれのデータについて，はっきり決まっている計算じゃないかと思うんです．

科学計算だと何かを2乗してみたり，3乗してみたりということをやるわけです．そして2乗したものをもとのやつと足してみたり．

事務計算には絶対にそういうことは起こらないですね．

森口　月給を2乗しても全く意味はないね．

和田　そうですね．家庭の事情にひびくくらいで．（笑）

森口　時間という意味のディメンションは決まっているんだけれどもね．他面面白いのは，データがいくつあるかということが不定であることが多いね．

和田　そうですね．アルゴルでは for I:=1 step 1 until 10 といういい方をするでしょう．コボルのほうは READ AT END ですからね．

森口　あれがちょっと面白いね．事務計算に共通な特徴ではあるわけだな．ところでその事務計算のプログラムというのは，ルーチンとして日常使うものである関係上，非常に内容が良くなくちゃいけない．

和田　しょっちゅう使うものはですね．事務計算でも期毎の計算がありますから，それはどんなに遅くてもかまわない．

森口　毎日のように，やるようなものはうんと能率良くなければならない．能率が非常に良くなければならないということが，事務処理のプログラムの作成を困難にしているのですか，それともももっと別の面にむずかしいところがあるんですか．

和田　そうですね．こんなことをいうと怒られるかも知れませんが，僕が感じているのは依頼するほうがずいぶんつまらないことまで要求するんです．それをこちらは忠実にやらなければならないというんで，だいぶ時間をくいますね．一つの例は，縦にリストをとっていけばいいんですが，それは見にくいから項目を縦にとって，期間を横にとってくれというんです．そうすると期間ごとにソートしてあると，項目ごとに縦にメモリーの中に全部入れてリストをつくっておいて，あとでラインプリンターで打ち出さなければいけないんですね．そんなものがたくさんあるんですよ．

森口　そうすると，そういう例外的な処理が要求されるということが一つと，それから印刷して出す様式にややこしい注文が多い，ということが一つ．

例外の処理というのは，やっぱり，いつまでたってもたくさん要求が出てくるんじゃないかと思うんだけれども．

和田　そうなんですが，あんまり例外があるというのは電子計算機的ではないわけです．オートメーションのためには，なるべく例外を減らさなければならない．で，必ずしもその方向には驀進してはいないような気がする．

森口　むしろ，そういう意味では，計算機の能力がありすぎるものだから，そういう例外の注文を受け入れることがどうにかできているんで，なかなか改まらないんだな．

で，いろいろ文句をいって例外をなくしてもらうよりは，いうことを聞いて例外を処理するプログラムをつくることのほうが簡単だということがあるのかな．

和田　そうかもしれませんね．

森口　それは今後は相当問題だと思うな．それからその印刷様式のややこしいのというのは，何とか改善できる見込みがありますか．やっぱり，これも無理を聞いてくふうするという……．

和田　そうですね．

森口　一つには君自身がいった，人間とのコミュニケーションの問題だから，ある程度人間に読みやすいようにすることは必要だね．

和田　そうなんですよ．紙に罫が印刷してあるわけですが，それに入れるのがまた非常にめんどうなわけです．で，一つはその罫までプリントしちゃうとか何とかということだと思うんですけれども，そのへんはもっとできるだけ割り切りたいと思うんです．あんまり何でもできるというと，いろんなことを押しつけられますからね．

森口　ところで，事務処理の問題のパターンですけれども，さっき給料計算とか，何とかいう分け方は，あんまりプログラマーにとって役に立たないということをいって，そのあと分類集計というパターンはかなり普遍的であるということをいわれたんですが，その場合に分類集計に似たような別のパターンで，それ以外のものがもうちょっとあるでしょう．

和田　そうですね．たとえば，あるコード（符号）のものだけをファイルから取り出すということがあるんです．もちろんそのコードはリジッドにきまったコードですね．

どうしてそんなことがあるかといいますと，やっぱり，それは一つのすう勢というか，事務計算の動向だと思っていいと思うんですけれども．

もとはいろんな事務のプログラムというのは完全に独立していたんですね．ところが最近はもちろんお互いに関係がありますからね．プログラムも関係をもつようになってきた．たとえば営業で売ったものの代金はちゃんと経理にはいってくる．

したがって売ったという記録が営業のプロセスのほうで出てきますと，それの金額を経理のほうのデータとしていっしょにつくってしまうわけですね．また給料を支払いますと，労務費は経理のほうから出す．そういうようにお互いの仕事の間でデータをコンピューターのところで渡し合うというようなこともだんだん

はじまっています．もっとも当り前といえば当り前のことなんですけれども．前は独立にやっていて，両方がくい違ったとかいっていましたけれども，いっしょにやるものでくい違わなくなるわけです．

森口 “一貫事務処理”とか，“総合方式”とかいう．

和田 そうです，ごまかしがきかなくなる方式です．あとは僕達のほうはまだやっていないんですが，そういうプロセスをやっているだけではなくて，やっぱり行きつく先はそういうものをもとにして管理面に使えるようなデータをつくるということ．そこまでいかないといけないと思うんですが，そんなのはとってもむずかしくってだめですね．

森口 やっぱり将来の課題としては，計算機による事務処理の最終製品はどんなものであるべきなのかということの再検討が必要でしょうね．

和田 ええ，それとそんなところまでゆきますと，もしかしたら2乗や3乗の平方根の計算が出てくるんじゃないかと思いますけれどもね．

まだ今のところ掛算くらいまでですね．割算もいらない．

森口 数量と単価を掛けて金額を出す計算は今でもあるね．

和田 ええ．

森口 ところで，いつか和田君が，ファイル・メインテナンス（file maintenance）ということと，リポート・ジェネレイション（report generation）とが非常に重要な事務計算の仕事だということをいわなかったかしら．

和田 そうですね．

森口 そういう部分をこれは何のファイルを維持するプログラムだということは捨象して，ファイルを維持するプログラムという形の抽象的な，すなわち，もうちょっと一般的なプログラムに仕上げておいて，そして給与のためのファイルだとか，在庫管理のためのファイルだとか，特定のファイルにそれを当てはめるときには，パラメーターをいくつか指定することによってそういうもののためのプログラムになってくれる．そういう一般的なプログラムにそういうものを仕上げておくことがファイル・メインテナンスについても，リポート・ジェネレィションについても可能であって，たいへん有益であるということを Journal of ACM で読んだことがあります．あなたも同感ですか．

和田 「どうかなあ」です．

今の段階ではパラメーターの数が多くてむずかしいんじゃないかと思うんですね，やっぱり，いろんな仕事のアウトプットのフォームがかなり統一されるとか，さらにインプットのフォームも統一されるとか，そういう必要があると思うんです．

今の僕たちのほうのプログラムで，確かにコントロールをきったり，サメイション（合計）したりというのはたいしたことはないんですね．その辺は誰でもくめるんですよ．だから，リポートライターとかでできると思うんですけれども．

もっとめんどうなのはインプットデータのチェックなんです．インプットのデータにはあらゆる種類の間違いがあると思わなければいけない．いつまで経っても，そのエラーというのは残るわけですね．科学計算ですと，データというのはほとんどないからいいんですがデータ・プロセッシングでは毎日データを打って入れてくるんですから，データのエラーには毎日びくびくしていなければならない．それが，フォームによってずいぶんチェックの仕方が違うんです．

で，けっきょくその辺はまたさっきのシステム・プログラムで人間は如何に間違うかということを分析しているのと同じで，間違いやすさを良く知ってインプットのチェックをやらないといけないですね．

コンピューターの速いのが要求されるのはけっきょくそこなんですね．間違っていたときにどうそれを訂正するかということがまた問題です．まあいちばん簡単なのは間違っていたデータをはじき出して翌日やる．急がなければならないのはその場で機械を一度とめて直す．(するとそこで機械のロスがちょっとあるわけですけれどもね)．あるいは機械に自分で直させる．いずれにしてもインプットのチェックを徹底的にしなければならないんです．

インプットのレベルで IBM 7070 のスプールというのがありますね．それはカードを読んでどんどん磁気テープに直して行って，そいつを今度はメインのプロセスサーでプロセスアウトプットの磁気テープに書いてしまう．それをスプールでプリントアウトするという流儀です．それがチェックのことを考えると，果たして実用的かということになっているんです．科学計算で7090の両側に1401を2台置いて，プログラムを全部磁気テープに入れて，そいつを 7090 でプロセスして，1401でまたプリントするというならかまわないと思いますけれども，事務計算でスプールをやったときにエラーがあるテープというか，チェックしない

原始データをそのまま磁気テープに入れて，さて急いでプロセスするときに，エラーをみつけてそこで止まっちゃったら，けっきょく元も子もないんですね．

それで，うんとインプットのフォームが違うときに，今何のデータであるということをまず確認して，それに見合うチェックのプログラムをまずロードしてスプールをやる．何でもかんでも同じスプールで同じルーチンで呼びこむというのは困るということなんですね．その切替えが果たして簡単にいくかどうかということが少し問題になっているんです．それから紙テープをスプールでプロセスしている最中に，同じところからコントロールテープをインプットしなければならないというのも問題です．

要するに過ちは人の常，許すは神の何とかですね．間違いだらけで嫌になってしまいますよ．

森口　そうすると，ファイルメインテナンスとリポート・ジェネレイションのほかにインプットのチェックという重要な段階があるわけね．

和田　そうですね．

森口　さて，今や科学技術計算をやるのにアルゴルかフォートランを使わないことは考えられないという話になっていたと思いますが，コボルについて，事務計算にはコボルがいいんだということがいえるのか，いえないのか，その辺はどうですか．

和田　コボルについては僕は，やっぱりこの次に僕たちのセンターに計算機がはいってきたらプログラムはコボルで書きたいと思うんです．

それはなぜかといいますと，もちろんデイリー・ルーチンですから，効率がよくなければならないということはあるんですけれども，さっきちょっと話したように，トラップでインプットを処理しようということ，スプールというのはそうですね，つまり，インプットと計算をやろうということをやるわけですけれども，そのインプットのルーチンと同時に動く計算のメインのプログラムですね，メインのプログラムに何を持ってくるかわからないんですね．メインのプロセスは，どんな仕事をしていても，すべてのプログラムとインプットでは，コンパティブルに動かなければならないという要請があるわけです．

場合によっては，メインのほうがアセンブリーやコンパイレーションをやっているときには，インプットのルーチンを動かしたいわけです．そういうことをしますと，そのメインプログラムのインプットやアウトプットの部分をいっせいに変更しなければならないということがあるわけですね．そういうトラップを使っていないときのプログラムの蓄積を，センターがある日突然にインプットルーチンが並行処理でできるようになったときに，それ以前にプログラムを全部書き直して，コンパティブルにいくようにしようというときに，それがもういろんなランゲージで書かかれていて入出力の力が勝手気ままになっていると，そういうことができないんですね．

それを統一的にやるためには，やはりコボルみたいなもので書いておいて，インプットのコンパイラーをちょっと直して，インプットの部分は，コンパティブルになるように，ソースから直してしまうということをやる必要があるんじゃないかと思っているんです．

それが今できていないものですから，こうすればインプットと並行処理できると思っているんですけれども，そのために今まで半年以上かかって書いた 7040 のプログラムを全部そこの部分をつくり直すというのはたいへんなんですね．

それをつくり直すことによってかえって混乱が起きると僕はみているんです．そういう混乱を起こさないためには，やはりコボルみたいにインストレーションが相当はっきりした方針をもって，プログラムのパターンを統一しておく必要がある．そういう意味でコボルは，ちょっといいような気がするんです．

それともちろん，さっきもいいましたように，一つのプログラムから次のプログラムにデータを送るというようなときには，同じファイルのかっこうをしているわけですね．そのときに同じ名前をつけておくという，ファイルが独立しているプログラムの書き方というのはかなり有効であろうと思います．だけど，その前提条件は効率のいいコボルのコンパイラーであるということですね．

それから，インプットのチェックは，やっぱりさっきいったように，コボルの細かいプログラムで書くということが大事なんですけれども，そこはどのくらいフレキシブルに書けるか疑問があります．

それから，ソーティングとかリポート・ライティングというのは，コボル・エクステンデッドでやれば，スマートにいくと思いますけれども，インプットのチェックがほんとうにたいへんでして，コントロールが切れる部分というのはどの部分，どのプログラムでも同じなものですからみなうまくなっているんですね．コントロールが切れれば新しいコードをセットしておいてまた比べると，イニシャルの最後のセットを気を

つければいいということになるんですけれどもね.

そのへんはたいていのプログラムが同じようなロジックで,同じようなフロー・チャートを書いていますからね.そんなようなことで僕は,コボルは使うのがいいんじゃないかと思います.

森口 そうすると将来の方向としては,コボルを使うようになるべきであるか,なるだろうか,どっちか知らないけれども,そのためには,いいコボル・コンパイラーをつくることが前提条件になるわけだな.

和田 そうですね.

むすび

森口 最後にまとめというか,いい落したようなことで,ソフトウエアの動向に関していい足しておきたいことはありませんか.

島内 さっき和田君がちょっといったような,絶対にこわれないシステム,こういうようなものがどんな機械にも備わってほしいと思いますね.

これはハードウエアのほうで,それだけの手を打っておかなければいけないと思いますけれども,システム・プログラムのある部分は絶対にこわれない,ユーザーがそこだけは絶対信頼できるんだという部分ができるようなハードウエアをつくって,そのうえにそういうようなソフトウエアをつくっておいてくれないと,これからプログラマーが分業ということは全然できなくなってしまう.

システム・プログラムとユーザープログラムの境をはっきりさせなければいけないということです.

森口 こわれないというのはメモリー・プロテクトの…….

島内 それだけじゃなくて,割り込み機構その他を完全にすること.

森口 矢島君はどうですか.

矢島 ユーザーの要求というのは,だんだんぜいたくになっていくというのがすう勢でしょうけれども,そのぜいたくさ加減というものの中から,パターンというものをつかみとっていく努力というのが非常に必要であり,重要な段階にきているんだろうと思うんですよ.

森口 そうね.先ほど和田君もいったんですが,自分のところで使うプログラムまでメーカーさんにつくらせたいというユーザーが非常に多い.今までメーカーさんのプログラマーはそういう要求に振り回されていたわけですね.

そういうのではいいシステムプログラムをつくる力が出てこない.そういうことがわかってきたんじゃないかな.

それは,やっぱり,ユーザー側でも考えを改めて,勝手気ままなプログラムをメーカーさんにつくらせるということは今後はしないで,その代わりメーカーさんからはいい基本的なシステム・プログラムを供給してもらうように希望することが必要ですね.

和田 そうですね.それとプログラムをメーカーとユーザーとに分けたくないんですよ.最初のソフトウエアの定義のときからそうですけれどもね.

ユーザーのほうでも自分のインストレーションのために,システ・ムプログラムをつくってかまわないんです.いろんなパターンをみつけたら…….

要するに,プログラマーというのは,ハードウエアじゃないほうをつくるんで,それがたまたまどっちからか給料を貰っていたかにすぎないんですから,そんなにはっきりした区別はないわけで,僕はもっとユーザーのほうも野心をもってシステム・プログラムをつくるといいと思いますね.

森口 そうだな.それもやっぱり基本方針は君がさっきいった,パターンを認めるということだね.

和田 まあそうですね.

森口 僕はこの間,UCLA の医学部の付属の計算センターに行ってみましたが,そこでやられている計算の半分以上が BMD (Biomedical) というシステム・プログラムを使っているんです.それはデイクソン所長とマッセイ次長がつくり出したシステム・プログラムで,要するに統計的なプログラムを全部パラメーター式で,何でもとり出せるようにして組にしたというだけだけれども,それは立派なシステムで非常に便利なオペレーティング・システムになっているわけです.そういうのは,やはり自分のところにくるたくさんの計算の要求の中から共通のパターンを見つけ出して,それを自分のところの一つのオペレーティング・システムという形に仕上げるという着意をもつことの例になると思いますね.

和田 そうですね.それから,メーカーは,変ない方ですけれども,あんまり効率のいいシステム・プログラムをつくると損すなんでね.つまり,機械がたくさんさばけない.で,ユーザーのほうはできるだけ効率のいいのを自分のところでつくるほうがむしろ得なんで,やっぱり立場が違いますからね.(笑い)

森口 なるほど,なるほど.しかし,そのときにユーザー専用のシステム・プログラムがうまく作れるような,もう一段基本的な水準,もう一段ハードに近いようなシステム・プログラムを作っておくということは,相当売込みの上に重要なポイントじゃないかな.

和田 そうかもしれないですね.

森口 そんなところにしておきますか,どうもありがとうございました.