解　説

プログラミング方法論の問題点*

——超職業的プログラミングについて——

木　村　　泉**

1. 問題提起

1.1 ソフトウェア作りの現状認識

大きなソフトウェアが製造者利用者の双方にとっていらだちの原因であり勝ちだという事情は，**ソフトウェア危機**の名とともに，いまや周知のものとなっている．経験の蓄積と関係者の努力によって，この困難な状況は少しずつ緩和されつつあるようにも見える．わが国でも，一流のソフトウェア製造者とごく常識的なソフトウェアについては，ほぼ予定通りのものが予定の時期からあまり遅れずにできあがってくることを，ある程度期待できるようになったと言ってよい．

しかし，そうであるために消費されつつある関係者の精力と所要時間は膨大である．たとえばソフトウェア製造者が作成する付属文書ひとつをとっても，その量はおそるべきものであることが常識とされている．正確なデータは多くの場合公表されないので，ここに引用することはできないが，一つのオペレーティングシステムの付属文書類がトラック1杯分あったときいても，事情を知る関係者ならさほど驚きはしないであろう．それだけの文書を単に書きおろすに要する労働力だけを考えても，これはかなりおそろしい事態である．

そのため当然経費は莫大となる．いきおい，開発の範囲も限られる．よほど特殊な「金に糸目をつけない」注文でもあればいざ知らず，ごく常識的なソフトウェア以外は開発の対象外に置かれる．そうしなければ引きあわないからだが，これがよろこばしいことでないことは明らかである．

こういう事情に改善の余地はないのか．どこかに大きな無駄がありはしないか．それともこれは，ソフトウェア作りにとって本質的なことなのだろうか．考えてみたい．

1.2 職業的プログラミングとアマチュアプログラミング

この問題を考えて行く上で，まずもって注意しなければならないのは，ソフトウェアの製造と単なるプログラム書きの間の相違である．この点に関する Weinberg[1] の指摘は適切である．彼は**職業的プログラミング** (professional programming) と**アマチュアプログラミング** (amateur programming) を厳格に区別し，これら2種の活動は，一見したところ似たようでありながら，本質的にはまったく異なったものである，と説いている．彼の考えによれば，これらの活動のもっとも根本的な相違は，前者が他人に使わせるプログラムを目標としているのに対し，後者は作者自身が使うプログラムを第一目標としているところから生ずる (前掲書．p. 122)．

アマチュアにとっては，自分の当面の目的が達せられさえすればよいのであるから，場合によっては誤った入力が入って来ると暴走するようなプログラムでことをすますことも可能である．職業的プログラミングの立場からすれば，それはあり得ないことである．利用者というものは，どんな途方もない誤りをおかすかも知れないものであるから，前もってすべての「病的」な場合を考え，しっかりした誤り対策を講じておかなければならない．

また職業的プログラミングにおいては，将来利用者の要求が変化してプログラムの変更（いわゆる**保守**）が必要になるかも知れないことを見越しておく必要がある．そのためには懇切丁寧な付属文書が不可欠である．しかも，その変更作業は，作者自身がおこなうものとは限らないから，他人にやってもらうときのことも考えて，ありとあらゆることを書いて書きまくるようなことにもなる．こういう事情はアマチュアプログラミングには，ほとんど存在しない．

このように考えると，職業的プログラミング，ひい

---

* Problems of Programming Methodology, or, On Superprofessional Programming au Essay, by Izumi KIMURA (Tokyo Institute of Technology, Department of Information Science).

** 東京工業大学理学部情報科学科

てはソフトウェア製造が高価につく要因は，数え上げればきりがないほどだ，ということがわかってくる．

1.3 超職業的プログラミング

ところで，すみずみまで読まれることを予想して書かれたソフトウェアがいくつか公表されている．もちろんそれらは比較的小規模なものに限られ，付属文書とリスティングをこみにして1冊の書物が形造られる，というようなものが多い（たとえば文献 2),3)．また非常に小規模であるが文献 4) もこの範疇に属する．）が，それらを見て驚くのは，付属文書が小さいこと，および，にもかかわらず全体としてわかりやすいことである．

また，全容を公表されたものではないが，ある程度の規模をもちながら，驚くべき少人数で短期間に開発されたと報じられているソフトウェア（古くは文献 5)，最近では文献 6) など）がある．

これらのソフトウェアは，ある程度広範囲の利用者層を想定している．一つの大学，または研究所の内部で公開される，というようなものが多い．そのようなものとして一応成功をおさめている以上，これらのプログラミング活動をアマチュアプログラミングの範疇に属するものとみることはできない．

では，これらは職業的プログラミングの一種であるか？ 定義によってはそうにはちがいないが，これらのプログラミング活動を通常の意味のソフトウェア製造とみてよいかと言えば，そう言い切ってしまうにはどこかしっくりしないものが感じられる．そこでこの種の活動に，特に**超職業的プログラミング** (super-professional programming) という名をつけて，これを通常のソフトウェア製造にともなう職業的プログラミング活動と区別して考えてみることにしよう．

1.4 超職業的プログラミングの特徴

超職業的プログラミングを通常のソフトウェア製造と区別するもっとも目立った特徴は何か？ 高品質で，開発に要したマンパワーが少ない，というのはもちろん重要な特徴であるが，それは何かもっと内的な特徴のあらわれとして生じた外的な特徴である，と考えられる．では内的な特徴は何か？ それは活動の主要な動機が利潤でなく，むしろ参加者の興味にあったことだ，と言えよう．この，動機の非営利性は，たとえば Dijkstra の THE システムについて言えば，文献 5) の冒頭から明確に読み取れるところである．

ところで Weinberg は，職業的プログラミングの一つの特徴として，過剰設計 (overdesign) が許されない，ということをあげている（前掲書，p. 126)．高度の技術を駆使して有用なソフトウェアを開発したとき，アマチュアは無条件にそれを自慢の種にしてもよいが，職業的プログラミングの立場からすれば，その開発が引き合うものだったかどうかを厳しく反省する必要がある．そのソフトウェアの開発にはどれだけの費用と時日がかかり，そしてそのソフトウェアが存在することによって将来節約される計算時間，得られる利便等はそれに見合ったものであったかどうかをよく考えなければならない．

このような冷静で批判的な態度は，本格的なソフトウェア作成において欠かすことができない．実際，たとえば Dijkstra[5] は，限られた人員で必要な知見を得るため，「しなければならない労働の総量を量的に増加させることのみに終るようなシステムのいかなる拡張への圧力にも屈しない」ことを指導原理の一つとしたが，これはまさにそのような態度の表明であったということができる．

しかしながら，一面から言うと文献 5) のプロジェクトは，全体としては過剰設計そのものであったとも言える．というのは，プロジェクトは一流の人々によって遂行されたのであり，それらの人々の人件費を適正に見積ってコスト計算をするなどということは，はじめから問題外だったと考えられるからである．むしろ動機の非営利性こそ，これらの人々を振い立たせ，よい仕事に駆り立てたものであったにちがいない．「システム設計の技術に貢献すること」[5] を主目的とし，使えるものを作ることは二の次にしたことが，結果的によいものを作り出すことにつながったのである．

1.5 研究的プログラミングについて

誤解を避けるため，ここで研究的な目的をもってなされるプログラミング活動（**研究的プログラミング**——research-oriented programming——）について一言しておきたい．超職業的プログラミングの範疇に属すると見られる活動の多くは，(たとえば文献 5) がそうであったように）研究的動機を有している．だが研究的動機は必ずしも高品質をもたらさない．むしろ当面の研究目的に奉仕しさえすればよい，との考えから，きわめてぞんざいなアマチュアプログラミングがおこなわれることが多い．そのような場合にも大幅なマンパワーの節約が見られることはあるが，それはどちらかと言えば手ぬきゆえになし得られた節約であり，それゆえ本文の考察の対象とはなり得ない．

1.6 注意，本文の構成

実は，ソフトウェアのよしあしを真に深く判断するためには，文字通りそれを手にとっていじってみることが必要であり，学術論文等の記述だけにたよることは危険である．ここで超職業的プログラミングの例としてかかげたものの中にも，さわってみれば意外に手ざわりの悪いものが含まれているかも知れない．しかしここでは，ひとまずそれらが文献上の記述から推察される通りの高品質をもつものと仮定し，それらがそうあり得た要因は何であったかを考えてみることにしよう(第2章)．そしてその上で第3章では，その検討の結果が暗示するものは何であるかを，探って見たい．

2. 超職業的プログラミング ——その成功の要因

2.1 個人的能力

1.3 節にかかげた成功例において，プロジェクトの構成員が例外なく一流の人々であったことは，見逃しがたい事実である．McKeeman ら[2] をはじめとして，どのプロジェクトも，すばらしい人々で満ちあふれている．

プログラマの能力には，はなはだしい個人差があると言われている．測り方にもよるであろうが，10倍を超える数字があがっている[7]．すぐれたプログラマばかりを集めることができれば，常識をはるかに超える成績を挙げ得るのではないか，と考えたくなる．それはたしかに一面の真理を含む考えであり，超職業的プログラミングの成功の一要素として，プロジェクト構成員の個人的能力（超能力？）があったことは認めざるを得ない．

2.2 規模の小ささ，目標の明確さ

1.3 節の例を見渡して目につくのは，プロジェクトの規模が小さいことである．技術的にはどれも非常に高いところをねらっているが，開発に要するマンパワーは最小限におさえるよう，立案の段階で細心の注意が払われている．

そういうことが可能であったのは，プロジェクトの目標が明確に設定されていたことによる．たとえば Dijkstra[5] の THE システムは，利用者間のデータ交換のためのサービスをせず，また機械語で書かれた利用者プログラムを受け付けないようになっていたが，そのように割り切ったからこそ，わずか6名の，他にも業務をもつプログラマによって，ほとんど虫のないシステムを，しかも機械語によって，短期間に作り上げることができたのであった．そしてその制限は，主記憶 27 ビット 32 K 語，紙テープベースのシステムにとって，決して不当なものではなかったのである．

これに関連して，Dijkstra[5] が，システム設計の段階に長い時間をかけた，と言っていることは興味深い．

いかに超人的にすぐれたプログラマばかりを集めたとしても，OS/360[8] を6名で，短期間に，しかも虫なしに作り出すことはむずかしかろう．商用のシステムでは，あらゆる顧客のあらゆる（時には気まぐれな）要求に応えようとして，とかく目標が不明確に設定され，かつシステムの規模も膨大になることを常とする．超職業的プログラミングの成功者たちは，皆明確な基本思想に立脚し，それを押し通していることに注目すべきである．

2.3 人数の少なさ

注意深く規模を限定されたプロジェクトを，非常に優秀なプログラマによって遂行するとすれば，必然的にプロジェクト構成員の数は少なくてすむことになる．人数が少ないことのもたらす影響の深刻さは，ここで強調するまでもないであろう．1人でできる仕事なら，打合わせは不要である．6人のチームなら，毎日顔を合わせて，コーヒーでも飲みながら，こまごました相談ができる．30人までのチームなら，一堂に会して忘年会ができる．2000人のチームが揃ってできることと言えば，ラジオ体操ぐらいのものであろう．

プロジェクト構成員の人数が少ないことから来る利点は大きい．たとえば，

(1) 1人が受け持つ部分の全体に対する割合が大きくなり，プロジェクト構成員ひとりひとりにとって見通しがよくなる．

(2) 連絡がよくなり，インターフェースの規約等について食いちがいが起りにくくなる．

(3) 構成員の質を均一化しやすく，したがって構成員間の意志疎通が容易になる．

(4) 命令系統を定めてグループを組織化するようなことは必要でないから，管理のための労力を省くことができ，仕事の分担なども臨機応変に変更しやすい．

などは，ただちに思い浮ぶところである．

2.4 リーダーの立場

以上のことに関連して，リーダーの立場という問題に触れておきたい．通常のソフトウェア製造では，リーダーはあまりこまかいことに首を突っ込まないのがよいとされている．事実それによってリーダーが見通

しを失ってしまうのではこまる.

これに対して超職業的プログラミングでは，リーダーとは文字通り指導者である．たとえば文献 5) における Dijkstra はその典型例である．この場合リーダーとは，(または，少なくともそのあるべき姿は) 技術的内容をプロジェクト構成員の他の誰よりも深く把握し，プロジェクトの進行に最終的責任をもつ人物，ということになる．そういう人物の存在が他の構成員たちから実力以上のものを引き出すことは，充分予想できる．

もちろんこれはプロジェクト参加者の人数が少ないからこそ起り得ることである．2000 人の構成員をかかえるプロジェクトのリーダーにとっては，上の意味での指導者であることよりも，冷静な管理者であることの方が，はるかにたいせつであろう．

2.5 付属文書の大きさについて

ここで注意を．付属文書に向けてみることにしよう．超職業的プログラミングにおいては，付属文書がコンパクトであることを常とするが，そうなった理由としては

(1) そもそも記述すべきシステムが小さい．
(2) よく整理されているので，記述に無駄がない．
(3) プログラムリスティングそのものが，よく整理されていて，それ自体かなりの程度まで付属文書の役を兼ねる．

と言ったあたりがすぐ思いつくところである．その他見落してはならないのは，

(4) はじめからよく考えて設計してあり，またほとんど誤りを含まないので，更新の必要が少なく，したがってそのための考慮が省ける．
(5) 将来手なおしをしたいと思う人が出て来たとき，まずシステム全体を理解することを要求できるので，部分的に読んだだけでとりあえず何かできるように配慮する必要がない．

の 2 点である．

そして逆に，付属文書が小さいことが，プロジェクトに要するマンパワーを大幅に切りつめる要因となることは，言うまでもないことである．

2.6 開発者と利用者の関係

超職業的プログラミングにおいては，開発者自身が利用者でもあることが普通である．これは重大な意味をもつ．商業ベースによって生産されたソフトウェアに，とかく肌ざわりの悪いものが多いことは周知の事実であるが，これは，開発者がその生産物を使ってみる機会をもたないという事情がわざわいしている，と考えられる．

開発者が利用者でもあることによって，利用者が何を欲しているかわかれば，使いやすいものができるだけでなく，手ぬきしてさしつかえない部分をみつけ出すのも容易になるわけであり，ひいてはシステムの小ささにも寄与することになる．

2.7 プロジェクト運用上の余裕

以上とは一応別の系列に属する要因として，超職業的プログラミングにおいてはプロジェクト運用上余裕が大きい，という事実をあげることができよう．この種のプロジェクトは，その非営利的性格ゆえに，厳密な契約で縛られるということがない．むろん大学なり研究所なりの中で，ソフトウェアの完成を心待ちにしている利用者はたくさんいるわけであり，もしソフトウェアが予定の時期に完成しなかったら，それらの利用者からはとかくの非難が出ることになるであろう．そしてその非難は，たとえば大学の環境の中では，当のプロジェクトのみならず計算機というもの自体にも向けられる，というおそろしいことになるであろう．しかし，通常のソフトウェア製造の場合のように，一つ一つの成功，不成功がただちに金額に換算されるようなことはなく，またプロジェクトの進行状況を情報処理についての専門知識のない上級管理者ないし注文主に逐一説明する必要もない．したがって，設計を進めて行くうちに新しい発見があったような場合には，出発点に立ち戻って外部仕様から考えなおすことも決して不可能ではない．特に設計の段階では，こういう「あと戻り」* がきわめてたいせつであり，それをプロジェクト構成員の自主的な判断によってなし得ることは，大きな意味をもつ．

2.8 関連する活動からの寄与

超職業的プログラミングにおいては，プロジェクトそのものがプロジェクトの目的をなしてはいないことが多い．この事実をいま仮に，プロジェクトの**超計画性**と呼んでおくことにしよう．

超職業的プログラミングにおけるプロジェクトの真の目的は，「学部レベルの計算機科学科学生のためのプログラミングシステム」を作ること (文献 2)，序文) にあったり，オペレーティングシステムに関する「アイディアを現実のオペレーティングシステムの上で使ってみることによってテストする」こと (文献 3)，序

* Ross は Horning との討論の中で，この現象に**ヨーヨー運動** (yo-yoing) という軽妙な命名をしている (文献 9)，p. 363).

文）であったり，「システム設計の技術に貢献すること」[5]であったりする．できあがったシステムを納入してしまえばそれで一応けりが付く，というわけではない．場合によっては，システム構成上の新しい概念（たとえば文献 5）の P, V-operation）がみつかり，本筋のソフトウェア作りからわき道にそれて，その概念を追求してみる，などということも許される．もちろんこういうことが行き過ぎると，いつまでたってもものが出来上らないことになりやすく，事実，そういうふうにして破産してしまったプロジェクトの話もしばしば耳にするところであるが，こういうふうにソフトウェア作りという活動に枝葉が出ていて，プロジェクトをいわばより高い観点の中に位置づけることができるようになっている，ということは，対象物を他との関係においてはじめて真に深く理解するという，われわれの思考過程の特質に照して，大変好ましいことであると言ってよい．

そのようなわき道は，通常のソフトウェア製造ではほとんど考えられない．Metzger の次の警句は，この間の事情を印象的に表現している．「もし貴君が世間の技術水準を超える仕事，つまり技術的新展開を要する仕事を引き受けたとしたら，それこそまさに悪い計画なのだ．」（文献 10），p. 14）

2.9 束縛条件のゆるさ

超計画性（2.8 節）の一つの帰着として，超職業的プログラミングにおいては，技術的選択の余地が大きい．通常のソフトウェア製造では，使用する言語，流れ図の書き方等について，注文主からこと細かな指示が与えられることが多い．また既存のソフトウェアとの両立性を要求されることは，日常茶飯事である．まして使用計算機を自由に選ぶなど，まず考えも及ばない．

超職業的プログラミングではだいぶん事情がちがう．たとえば Dijkstra[5] は，指導原理の一つとして「健全な基本的特性をもつ機械を**選ぶこと**」（強調：筆者）をあげているし，Stoy[3] らに至っては，使用計算機（Modular One）の命令体系が当面の目的にふさわしくないという理由で，全システムを IC と称する仮想計算機の上で組み立てるようにしている．インタプリンタで構成された仮想計算機の上でソフトウェアを作るなどということは，通常のソフトウェア製造ではかなり考えにくいのではないか．合理的な理由のあるなしにかかわらず，注文主から「なぜそんな非能率なことをするのか？」という不満が出るであろう．

場合によってはハードウェアの選択まで考えなおしてよいのだとすれば，それをするだけの力のあるチームにとっては「あと戻り」(2.7 節）の余地が増すわけであり，その分よいソフトウェアができる可能性がある．

2.10 プログラミングの道具，手法，環境

これに関連して，超職業的プログラミング活動ではプログラミングの道具等について選択の自由がある，ということが注目される．通常のソフトウェア製造では，コーディングシートの様式，注釈の書き方などがことこまかに「標準化」されていることが多く，生れつき字の大きい人が苦労させられたりする．超職業的プログラミングではもっと個人差をおもんばかった活動の進め方がしやすく，そしてこのことが意外な能率向上につながる可能性は大きい[1]．これは，夜型の人間に能率よく働いてもらうには午前 8 時に始業ベルが鳴るのはぐわいがわるい，というような話にもつながる．

そして多分もっともたいせつなのは，超職業的プログラミングにおいて，プロジェクトの進行にともなって道具，手法が開発されて行き，それがプロジェクトの含み資産として蓄積される，ということであろう．「ちょっと便利な小物」（虫取りの道具など）が，時に非常な能率向上をもたらすことは，しばしば経験するところであろう．

2.11 そ の 他

たとえば XPL コンパイラシステムのプロジェクト[2]において，プロジェクト構成員たちがコンパイラ構成に関する理論的結果を充分認識していたことの効果は決して小さくなかったと考えられる．理論的裏づけに魔術的効果を期待するのはあやまりであるが，超職業的プログラミングが有用な理論的結果をとり入れやすい立場にあることは事実である．

そして最後に，（言葉は必ずしも適切ではないが）**士気**についても一言しておかないわけには行かない．超職業的プログラミングにおけるプロジェクトの超計画性が，構成員の意欲を高める方向にあることは否定しがたい．テストの対象になること自体が作業者の生産性を高めるという，**ホーソン効果**（Hawthorne effect）と称する現象の存在は，この間の事情を説明してあまりあるものである．なおホーソン効果については，文献 1），p. 31 または文献 11），pp. 507〜508 参照．

3. 考 察

3.1 他山の石としての超職業的プログラミング

2章の分析からわかるように，超職業的プログラミングの方法は，決して右から左へと通常のソフトウェア製造の場に持ち込めるようなものではない．たとえば優秀なプログラマを集めること，プロジェクトの規模を小さくすることなど，特に後者は，かなりの程度まで超職業的プログラミング固有の事情におぶさっていると言える．大きいものを作らなければならないが，それにはどうしたらよいか，と言うのが多くのソフトウェア製造プロジェクトにとって問題の出発点であり，その前提を考えなおさなければならないのでは，はじめから話が合わないとも言える．この意味では，2章の話は通常のソフトウェア製造にとっては，他山の石であるほかないのかも知れない．

しかし，そうではあるにしても，超職業的プログラミングの成功例[2)~6)]における成果は，あまりにもあざやかであり，したがって，2章の分析から浮び上って来る問題点を無視することは，賢明とは思われない．たとえば，システムの開発者に，作ったものを自分で使ってみるチャンスを与えるような，何らかのフィードバック機構を考える（2.6 節）などは，充分やってみる価値があろう．

3.2 教育の問題

しかし，当面のソフトウェア製造技術をどうこうする，というよりももっとたいせつと思われるのは，教育の問題である．優秀なプログラマとリーダーを多数揃えれば，それだけで話がだいぶん楽になることは明らかである．そしてそれには，情報処理の専門教育を発展させる他ない．わが国において最近まで，この点が少しないがしろにされ過ぎて来たように思うのは，筆者一人であろうか？．

なおここで言う教育は，企業内教育のみによるのでは不充分であるように思われる．たとえば超職業的プログラミングにおいては束縛条件のゆるさが生かされている（2.9 節）ことを指摘したが，逆にそのゆるさを生かすだけの自信と力があれば，通常のソフトウェア製造でも，束縛条件をゆるめることによって長い目で見た効果を上げる可能性はあるはずである．だが，現在進行中のプロジェクトのポリシーを一度くつがえしてまで新しい方法を導き入れようとしてみるだけの勇気が，企業内教育のわくの中で養成可能であるかどうかは，かなり疑問である．

一般情報処理教育の重要性もまた見逃してはならない．顧客というものは，しばしば「気まぐれ」な要求を出すものである．それは多くの場合に，ちょっとした常識の欠如による．だが外部仕様にわずかな無理があったためにソフトウェアが際限もなくふくれ上ることは，あまりにもしばしば見聞するところである．その辺の呼吸を広く一般の利用者が理解するようになれば，規模を切りつめ（2.2 節）て，すっきりした使いやすいソフトウェアに到達する道も開けるのではないかと思われる．

3.3 関連する話題

与えられた紙数が尽きたので，これ以上の検討は別の機会にゆずり，2，3 の関連する話題の存在を指摘して，本文を終りたい．

その第1は，**構造的プログラミング*** (structured programming)[12)] である．この言葉は，さまざまな使われ方をしている．少なくとも Dijkstra の言う構造的プログラミングと IBM 社のSP[13,14)] とは，一応別のものと考えた方がよいような気がする．Dijkstra の考えの中には，規律あるプログラミング，という発想が暗に含まれており（たとえば文献 15)），下向きプログラミング法，**go to** なしプログラミングなどは，それを実現するための手段として考えられた副次的存在であると見られる．彼の構造的プログラミングは，この意味で，超職業的プログラミングの発想（2章）と照し合わせながら理解されるべきである．Dijkstra が THE システム[5)] の開発者であることを忘れてはならない．

第2は**主任プログラマ制**[13)] (chief programmer team) である．これは，たとえば PL/I で数万行という程度の応用プログラムを作る際に有効な手段として考え出されたものであって，少人数のチームを考える点で2章で示したような発想に学んだふしがある．なお主任プログラマ制では，ライブラリアン (librarian)

* **構造化プログラミング**という訳語もおこなわれているが，日本語としてひびきが異様であるのでここでは採らない．実際，構造化プログラミングとは何物かを構造化するプログラミングと解せられる．では事物を構造化する，とはどういうことか？ すなおに解釈すればそれは，その事物が構造であるようにすることであろう．今の場合，その事物とはプログラム，またはプログラミングであると考えられるが，そのいずれと解しても，つじつまがあわない．
実は原語自体，若干あいまいな色彩を帯びていて，さまざまな解釈が可能であるが，文献 12) などから見て，この言葉の発案者 (Dijkstra) 自身の感覚は，「構造をだいじにしたプログラミング」というあたりにあるように思われる．とすれば「構造志向プログラミング」とでも言うのがもっとも適切であろうが，それではあまり長くてわずらわしいので，(「…的」という表現がもつ造語能力とあいまいさをむしろ積極的に利用して) 構造的プログラミングと言っておくことにする．

を導入したことが特に目を惹く．

第3に Weinberg[1] の**自我滅却プログラミング** (egolessprogramming) がある．これは「誰それのプログラム」ということを言わず，チーム全体でプログラミングを進めて行く，というやりかたであり，多くの超職業的プログラミングにおいて，自然発生的に実行されていることであると推察される．これは非常に重要な話題であると思うので，注目を呼びかけたい．

謝辞 和田英一，筧 捷彦，辻 尚史の各氏に討論をしていただいた．また 2，3 の企業関係者の方々との折に触れての座談はきわめて有益であった．ここに記して謝意を表したい．

参 考 文 献

1) G. M. Weinberg: The Psychology of Computer Programming, p. 288, Van Nostrand Reinhold, New York (1971).
2) W. M. McKeeman, J. J. Horning & D. B. Wortman: A Compiler Generator, Prentice-Hall, New Jersey (1970).
3) J. E. Stoy & C. Strachey: OS6—an Experimental Operating System for a Small Computer, Parts I & II, Computer J., Vol. 15, No. 2, pp. 117～124 & No. 3, pp. 195～203 (1972), および C. Strachey & J. E. Stoy: The Text of OS Pub, Technical Monographs PRG(t) & (c), p. 131 & p. 155, Oxford Univ. Computing Lab., Programming Res. Group, Oxford (1972).
4) 島内剛一：システムプログラムの実際，p. 222，サイエンス社，東京 (1972).
5) E. W. Dijkstra: The Structure of the "THE"-Multiprogramming System, Comm. ACM, Vol. 11, No. 5, pp. 341～346 (1968).
6) D. M. Ritchie & K. Thompson: The UNIX Time-Sharing System, Comm. ACM, Vol. 17, No. 7, pp. 365～375 (1974).
7) H. Sackman, W. J. Erikson & E. E. Grant: Exploratory Experimental Studies Comparing Online and Offline Programming Performance, Comm. ACM, Vol. 11, No. 1, pp. 3～11 (1968).
8) IBM System/360 Operating System Introduction, File No. S360-20, Order No. GC28-6534-4, p. 105, International Business Machines Corporation, New York (1972).
9) W. L. vander Poel & L. A. Maarssen, ed.: Machine Oriented Higher Level Languages, p. 535, North-Holland, Amsterdam (1974).
10) P. W. Metzger: Managing a Programming Project, p. 201, Prentice-Hall, New Jersey (1973).
11) J. J. Horning: Structuring Compiler Development, Chap. 5. B of F. L. Bauer & J. Eickel, ed., Compiler Construction, Springer, Berlin, pp. 498～513 (1974).
12) E. W. Dijkstra: Notes on Structured Programming, *In* O.-J. Dahl ed., Structured Programming, p. 220, Academic Press, London (1972). 野下他訳：構造化プログラミング，サイエンス社，東京 (1975).
13) F. T. Baker & H. D. Mills: Chief Programmer Teams, Datamation, Vol. 19, No. 12, pp. 58～61 (1973).
14) IBM Data Processing Division: Six Improved Programming Techniques (Advertisement), Comm. ACM, Vol. 18, No. 4, p. A-6.
15) E. W. Dijkstra: The Humble Programmer, Comm. ACM, Vol. 15, No. 10, pp. 859～866 (1972). 木村訳：謙虚なプログラマ，**bit**, Vol. 5, No. 11, pp. 1243～1254 (1973).

Note: A version in English of this paper will appear in *Technical Reports of Information Science*, Tokyo Institute of Technology, Department of Information Science.

(昭和50年6月23日受付)