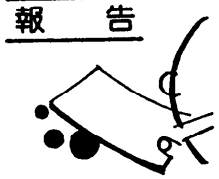
報　告



パネル討論会

設計技法の現状と将来

「プログラム設計技法の実用化と発展」シンポジウム†報告

パネリスト

紫合　治[1]，水沢　明[2]

三浦　大亮[3]，山崎　利治[4]

司会　吉村鉄太郎[5]

司会　私は司会者の管理工学研究所の吉村と申します．よろしくお願いいたします．

まずパネリストの方々を紹介申し上げます．日本電気(株)の紫合さんはメインフレーマの研究者として，ソフトウェア工学の研究をしておられます．

東京ガス(株)の水沢さんは実際に大型のデータプロセッシングの現場でいろいろと経験を積まれた方です．

東レ(株)の三浦さんは日本でコンピュータアプリケーションの実用化が始まった時代からずっとソフトウェアに関して経験を積まれ，いろいろ学会にも貢献してくださっている大先輩です．

日本ユニバック(株)の山崎さんも同じく日本のコンピュータアプリケーションの黎明期から一貫して理論と実践の両面で興味深い経験と貢献をなさってきた方です．

きょうのパネルは，この3日間の発表，討論を前提にして，ご出席の方々に設計技法についていろいろお考えを伺い，また，聴衆の皆さんとのディスカッションをできるだけ取り入れたいと思います．ディスカッションのテーマとしては，このOHPに掲げました8つばかりのポイントを中心に進めたいと思います．

2時間を30分ずつ4ラウンドに区切りまして，最初の1ラウンドは，パネリストの方々にとってプログラム設計方法論とはなにか，興味を持っている技法，推薦される技法，技法とツールとの関係などの事柄について特に発言したいと思われることをお話しいただきたいと思います．

第2ラウンドは，メソドロジの普及に関するご意見を承りたいと思います．

第3番目のラウンドは，プログラミング・メソドロジ全般と，この3日間のシンポジウムの成果に関してご意見があれば承りたい．

最後のラウンドは，それまでに発言されたことのしめくくりと会場の皆さんとパネリストの方々とのディスカッション，意見の交換にあてたいと思います．

そこでまず，第1ラウンドに関したテーマで，まず，紫合さんからお願いします．

紫合　日本電気の紫合でございます．最初に「私にとってプログラミング方法論とは」という話をしてみたいと思います．

「プログラミング設計技法論」というのは設計の定石を述べたもので，いわゆる「形」を定めています．形というのは「心」に至る近道といえます．例えば，ネクタイをしめるのは，他人を敬うとか，そういう心に対する表面に出ている形ですね．その形がないとなかなか心に至れないということで，形は非常に重要であると思うわけです．

さらに形は心を共有するツールといえます．プログラムを元の問題に反映させるべきだというよりも，その入出力データ構造を対応づけて…という形をいったほうが，いろいろな人と共通に話ができるわけです．ただし，そういう定石を使っていく場合，状況を見て選ぶことが非常に重要だと思います．私はいろいろな方法論が全部好きなんですけれども，あまり一つの方法論を過信するのは禁物であろうと思っております．ある場合にはジャクソン法がよくて，ある場合にはデータフロー設計がよくて．いつもネクタイをしめるのでなく，ある場合には寝巻きの方がいいというような意味です．

---

† 日時　昭和59年4月12日（木），15：00～17：00
場所　機械振興会館大ホール
1) 日電ソフト生産技研，2) 東京ガス，3) 東レ，
4) 日本ユニバック，5) 管理工学研

もう一つ考えていますのは，やっぱり無意識で使うようにならないと本物じゃない．名優というのは舞台に上がる前にはいったんせりふを全部忘れてしまうというのですね．で，舞台では，そのときに感じたことを口に出す．そうするとそれがたまたまセリふと同じだと．そうなって初めて名優なんだということを聞いたことがあります．無意識に設計していたら，それがある場合にジャクソン法にのっとっていたとか，それぐらいに使えるようにならないと本物じゃなかろうというふうに感じております．

推薦する方法論としましては，まず SADT のような，階層化の概念ですね．それからモジュールの分割法として，データフロー技法や，抽象化技法，複合設計など．データフローは独立した処理からなるということで，連邦制と考えられます．複合設計でのモジュール構造というのは，メインが下位モジュールを制御するというので中央集権的だとしますと，抽象化とかパルナス法というのは下位部分がかなりの勢力を持つというので地方分権的．これもやっぱりその問題によって，この場合にはこれがいい，この場合にはこれがいいという選択を行う必要があると思います．

さらにモジュール内設計では，ジャクソン法とか，ワーニエ法，それから二村さんのお話の PAM とか，その他各種図式なんかもいいと思います．

水沢 まず「あなたにとってプログラミング設計技法とは」ということなんですけれども，私どもコンピュータユーザの立場からいわせていただきますと，プログラム設計技法というのは，システム開発とか保守を効率化して，システムを安定稼働させるための道具の一つであるというふうにとらえています．

実際にいくつかの技法を使った経験からいきましても，プログラム設計技法というのは，システム開発・保守の仕事を効率化するのに非常に役立っていると考えています．しかしプログラム設計技法は，それを単独に導入して効果が上がるというものではなくてシステムの開発・保守の各ステップでの問題点を明確にした上で，長期的視野に立って設計技法というのをどうやって使っていくかということを各ユーザで考えて，位置づけて使っていかなければいけないのじゃないかと考えています．

こういったいろいろなステップの中の仕事と関連づけてプログラム設計技法を使っていくという立場からいいますと，単に良質なプログラムを効率よく開発する技法というのにとどまるのでなくて，もっと前のステップである要求仕様定義のステップとか，プログラムができた後のテストのステップ，これらを含めてシステム開発全体を考慮したプログラム設計技法が必要なのではないかと考えております．

それから「興味を持ったプログラム設計技法」ということなんですけれども，私どもが使っている手法は非常に古くて，新しい手法というのをあんまり勉強してないので申し訳ないんですが，従来技法といわれる中では，ワーニエ技法というものに非常に興味を持ちました．

理由としましては，当時，データ構造に着目してプログラムを展開していくという考え方が非常に目新しく映ったということです．

また，事務処理のプログラムというのは比較的やさしいものなんですが，そういうプログラムに適用していくのに非常に向いているのではないかと考えました．

このシンポジウムのオープニング・アドレスのときに木村先生から，プログラム設計技法というのは，100点を狙うのでなくて，60点を狙うものであり，だれにも使いこなせなければいけないのだという話があったと思うんですけれども，そういった面からも，ワーニエ技法というのは非常に使いやすいのではないかと考えています．

「推薦するプログラム設計技法」ということなんですけれども，これも勉強してないので何を推薦するかということは非常に言いにくいのですが，当社では複合設計プログラム，ワーニエ技法で，構造設計と展開を行って，あとは記述は NS チャートを使うという方法をとっていて，これは比較的うまくいっていると考えています．

ただ事務処理という簡単なプログラム設計に関してはこれでうまくいくのでしょうけれども，もうちょっと複雑なシステムをつくる場合，果してうまくいくのかなという疑問はございます．

それからツールとの関係ということなんですけれども，ツールはプログラム設計技法と関連して，必要なものと，まったく独立に設計技法とは関係なく存在するようなツールというのが両方あり得るのだと思います．このシンポジウムではプログラムの非本質的な部分というように言われている部分に適応できるようなツールでも非常に効果が上がることがあるわけで，必ずしも，プログラム設計技法と直接に結びついたようなツールではなくてもどんどん使っていけばいいん

ではないかというように考えています．

三浦　私の予稿集では，囲碁の例にたとえればとしていますが，特に囲碁にこだわっているわけではありません．あくまでもたとえということでお許しいただきたいと思います．

なぜそんな話になったかといいますと，社内で研究発表会というのがありまして，その中でたまたまソフトを主題とした発表がありました．一応話が終わった後，トップの方から，どこに研究の価値があるのか，発揮する機能そのものはごく当り前のものじゃないか，何が研究要素なのかという質問があったわけです．

それに対して発表した人は技術的な説明をしたわけですが，補足として，その発表した人の上司から，同じ碁でも名人とザルとでは違う．いま発表したソフトは名人のソフトである．同じ碁でも中身が違うというような説明がありまして，なるほどそういう話の仕方もあるかと思ったものですから，この例になったわけです．

最初の「プログラム設計方法論」ですが，方法論ですから，方法ではないと解釈します．まず，変更と誤りに強いアプローチであること．碁でいえば，優れた布石をすることだと考えます．しかしそれは碁の名人をつくることではなく，経験が少なくてもそこそこよくできる，それから不備があっても，あとで対処しやすいということです．

第1日目に，例題についていろいろ話がありましたが，あれはかなり経験があるからできたので，経験がなくてあそこまでいくかなという疑問は持っているのです．

布石の後は，局地化とか，局地戦（モジュール化）とかが出てきますが，方法論として最も重要なことは，良い布石ができるようにすること，どういう考えであったら布石がうまくできるかということを教えてほしいと思います．

布石というのは，先を見てやるということですね．ちょっと脱線しますけれども，予稿集の私の部分だけは吉村さんのスペックに合ってないんです．

なぜかというと，私はパネルの題名と予稿に書くべき分量と，原稿と，納期を教えられました．それを完全に守ったのです．郵便局に行って投函したときに吉村さんから電話がかかってきて，こういうスペックでやると伝えられたわけです．ソフトウェアというのは，常にそういう関係があるのです．私は納期を優先したために，字もきたないし，内容も完全にはマッチしていない．しかしこの種のパネルであればこんな話になるだろうという予想で書きましたから本質は合っています．書き方は違いますが内容は同じである．それが布石です．そういうことができることが重要だということです．

次はソフトウェア・ツールとの関連です．ツールはやはり布石の考え方から発生して，先ほどの紫合さんの話にもありました定石集とか，詰碁集とか，棋譜集とか，場合によっては（これは幸いにして碁と違っていて，いくらカンニングしてもいいので）適当な相談相手でもいいわけです．

そしてこれに上手に利用する仕組みもあってほしい．プログラムのデータベースとかいう話もたくさんあります．いままでの発表の中にも，端末を使ってモジュールを取り出すという話もありました．そういうような仕組みとの組合せがソフトウェア・ツールであると考えます．

ただ問題は，上手と下手では利用の効率，理解の適切さが違う．立派な定石集があっても上手な人はうまく使うが，下手な人は間違った定石を使うかもしれないということがあります．

ですから利用の仕組みとしては，その実力の差がなるべく小さくなるようなものが優れたツールになるのではないかというふうに考えています．

下手な人は下手なりに時間をかけて相談もするし，カンニングもして，ていねいにつくるという態度が一方では必要です．名人というのは短時間で，名人の心境でさっとやれば結果としてよかったというものかもしれません．

山崎　まず，設計法というのはプログラムがどうしてこう書けたのかという弁明・説明するためのものだろうと思っています．プログラムの書き手と読み手が設計法を共有していることによって同じ発想をもつことができる．そうであれば，そのプログラムの読み手がうんと楽をするだろう．またプログラムの書き手にとってはたぶん文書が少なくて済むようになる．こういったところから設計法はとてもありがたいものであると思っています．

それで「興味を覚えた設計法」ですが，紫合さんと同じで何でも飛びついてみようという野次馬ですが，一例をあげればウィーン開発技法があります．

これは実用的なコンパイラを自動的につくろう，コード生成まで含めてつくろうということが下敷にな

っているものです.

表示的意味記述における意味関数をそのまま実現するようなものをつくり，それを構文解析ルーチンにうまく組み入れればコンパイラができる．このアイデアを事務計算にも利用しよう．たとえば，酒倉庫の問題に対してもこれを真似て書くことができるでしょう．

このようなやり方がなぜいいかといいますと，課題をうまく書いて，だんだんそれを書き替えていくと実行可能な記述になって，それがプログラムなんだというようなことになりそうだからです．

米田先生が「情報処理」(81年6月号，プログラム言語特集)に，「Abstracto 84」をお書きになっていらっしゃったのですが，それがちょうどこういう枠組みだと思うんですが，それをわれわれも及ばずながら真似たいと思うのです．それはプログラムの設計段階でも課題定義が直接に現われていることが特別に大切だと思うからです．

司会 どうもありがとうございました．引き続き，「設計技法及びプログラミング・メソドロジの普及」ということについて，直接関与しておられる現場，あるいは一般の，いわばデータプロセッシング・コミュニティを見たときのお考えを承りたいと思います．

紫合 一般的にはまだまだ普及が足りないのじゃないかという話を少しさせていただきたいと思います．

まず一番大きい理由は，方法論を知らなくても仕事ができるということ．新しいものを知るよりも知らなくてもいまやっているからそれでいいじゃないかということです．ですから知っていると得だということを本当に思わせないとだめだと感じております．

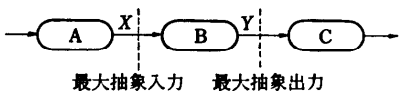
次に，知っていても各自の問題分野で十分に消化していない．さきほど，無意識に使うようにならないとだめというような言い方をしたのですが，無意識ほどでなくても，解説書に書いてある例題で理解しただけではだめで，少なくとも自分の問題でその方法論を実践してみるということをやらないと，なかなか身につかないというふうに思っております．

さらに，一つの技法だけを過信すると，かえって無理が生じる．これについては後でちょっと触れたいと思います．また，たくさんの技法がありますが，その選択基準が不明です．この辺をきっちりやりたいと思っています．

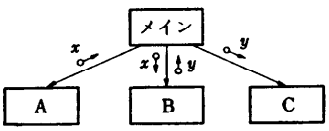
最後に，普及にはやっぱり10年はかかる．どんなものでも10年はかかるから，もうちょっと気長に待てばいいのじゃないかと．ただし，Liskovの抽象データ型の論文*が1974年に出ましてから，もう10年になります．そろそろ先達的なグループが，抽象データ型の概念を無意識に使っていてもよいというふうに考えております．

過信しすぎるとまずい例として，複合設計の一部をとり上げてみたいと思います．複合設計では，最初にデータフローを考えます(図-1(a))．これは非常にわかりやすい方法だと思います．次に，最大抽象入力点，最大抽象出力点という概念をもとに，コール関係によるモジュール化にします(図-1(b))．しかし，もとのデータフロー構造を実現するには実際にはもっといろいろなやり方があるわけです．たとえば，Aをメインプログラムにして，B，Cをコールする方法とか(図-1(c))，Bをメインにして，A，Cをコールする方法とか(図-1(d))．本当は別々のプロセスによるデータフローのままで実現するのが最も自然なわけです．ジャクソン法ですと，これをプログラムインバージョンで対処することになります．要は，場合によっていろいろ考えて，どの方式にするかを決めないといけないと思っております．

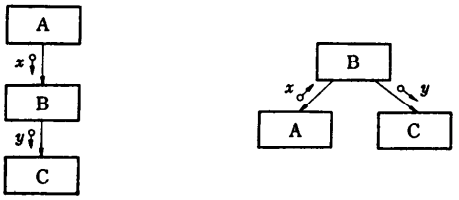
水沢 「設計方法論の普及度，あなたのまわりでは」



(a) データフロー図



(b) (a)の呼出し関係による実現(複合設計の場合)



(c) Aをメインにした場合 (d) Bをメインにした場合

図-1 データフローから呼出し関係への変換例

* Liskov, B. and Zilles, S.: Programming with Abstract Data Types, SIGPLAN Notices, Vol. 9, No. 4 (1974).

ということですけれども，私の考えているレベルの設計技法というのは，いまだ批判のありましたゴートゥーレスとか，三基本構造とかのレベルまででございまして，それをあらかじめお断りしておきます．

そういう意味では，まず「あなたのまわりでは」という意味ではたしかに普及しております．当社では1977年から導入を開始したのですが，1，2年で完全に普及しております．それ以前に作成されたプログラムも，もうすでに，ほとんどなくなってきております．

ただこれは事務処理をやっておりますセクションだけでございまして，その他のセクションではまだまだ三基本構造のレベルの設計技法に関しましても普及してないというのが現状だと思います．

その次に一般的なプログラム設計技法の普及度ということなんですけれども，同じようなレベルでございましたら，一般の大型計算機のユーザにおいては何らかの形でこのような方法を用いているのではないかと考えています．この理由としては大型計算機のプログラム言語は三基本構造の範囲はサポートしているということ，それから雑誌等に発表されたような実例を見ましても，かなりストラクチャ化されたプログラムが見られるということ，またユーザ研究会などでは最近はプログラムの構造化というのが話題にならないということがあげられます．

この件に関しまして，吉村さんの予稿にわれわれユーザがこのレベルで満足しているというお叱りの言葉があるのですけれども，これにはそれなりの事情があります．少なくともプログラム設計に関しましては，そのレベルまでにしろ，一応は効率化されたと考えております．

ところがそれよりも前のステップでありますシステム設計になりますと，三基本構造＋ゴートゥーレスほどの方法論もまだ普及してないというのが現状だと思います．一人一人勝手なことをやっているおり，出来上がったシステムもばらばらです．われわれとしては，次にそこのところの効率化を手がけていかなければいけない，そのためにプログラム設計技法に関してはとりあえず現状のままでおいておこうということだと思います．

それから最近のOAブームで，オフィス・コンピュータとか，パーソナル・コンピュータがたくさん入っているのですけれども，この分野ではなかなか，いま言った単純なストラクチャード・プログラミング・レベルの技法でも，普及していないのではないかと思います．

この理由としましては，これらのシステムを開発するような人が，プログラム設計技法教育をまったく受けていない．受けてる人もおるでしょうけれども，受けてない人の方が多い．それから機器の能力の制約上，冗長度の多いプログラム設計技法というのは，現在のところ適用できないとか，使用する言語がそれをサポートできないとかというような事情があるのだと思います．われわれが，大型計算機で10年近く苦しんだことが，このオフィス・コンピュータとか，パーソナル・コンピュータの分野で，今後くり返されるようであれば非常に問題であると思います．

三浦　まず，一般への普及を言う前に，お前自身はどうなのかといわれそうなので告白をしておきますが．

例えば構造化設計とかなどは大昔から実行していたことなので，いまも我流でやるということで徹底しております．

しかしながら，自分でときどきプログラムを組んで楽しんでいますけれども，自分がユーザで，自分が設計者で，自分がプログラマで，三位一体なのにもかかわらず，その三者はものすごく悩むわけです．その三者が同一人であっても，ものすごく時間がかかるのを，他人で，しかも複数のグループ同士だったら，とてもちゃんとしたものができるはずがない．それが現実で，だから布石とか先を見通すことが大切で，あいまいなことを全部包含したような設計をしなければいけないだろうと言ったわけです．

Ackoff も言っておりますけれども，未熟な技術者と，あまり有能でない実務家がいくら協力しても，いいシステムもソフトウェアもできないのです．そこにいくらツールや技術があっても，いいものはできない．

とはいえ，普及の方は十分ではないと思います．それから今後も難しいのではないかと思う．なぜかといいますと，まず，勉強しなければならない．

いままでの発表でも，本を見て独習でマスタできるというような類のものじゃない．高級なわけです．そうするとかなり努力をしなければならない．個人的な興味を持って努力する人はもちろんたくさんいますけれども，まず，経営者とか管理者がチャンスを与える必要がある．そのためには時間とお金がかかる．その実現のためには，それが効果的だということを多くの管理者とか経営者に理解してもらわなければならない

わけです．いまこのシンポジウムにこられない人に問題があるわけです．

それからもう一つは，ツールである定石集を見ても，理解できるかできないかということが，そもそも問題なんです．理解できる人は，すでに相当高いレベルです．そういうものを見て，やっとこさ理解できるレベルというのは初段ぐらいといわれています．初段ぐらいまでになれば，自己啓発的に上達するチャンスがある．しかしそれ以下の人は，いくらやっても，へぼ同士が毎日昼休みにやっても決してうまくならない．それは必然的な手を打たないからです．後で反省のしようがない．他の人の手を見てなるほどそうかなと思えないというレベルです．どうしても初段レベルまで持っていかなければいけない．

まず，利用者の立場とレベルに合った技術をはっきりさせる．今回発表された技術でも，だれのための技術なのかということですね．これをはっきりさせなければいけない．それからその立場に合った教育，訓練がともなわなければいけない．そうしなければ，今後も普及は困難ではなかろうかというふうに思います．

ただ，普及する必要があるのかないのかというのは，また別の判断であり得るというふうには思っています．

たとえば素質のない人に無理やり教えても役には立たないかもしれない．適性を考えて教育しなければ，その人の将来を損うことになるかもしれない．ひょっとするともう普及すべき必要のある範囲，あるいは可能な範囲は済んじゃって，相変わらず初段レベルにならないような人は対象から，始めからはずしてもいいんじゃないかというような議論もあり得ると思います．

山崎 もうあまり申し上げることがないんですが，私も設計法がはやらないという現実がある点を注意したいと思います．

それは一つには，プログラミングはどのみち，やさしいもので，実際にマイコンの普及とともに，やりたい計算はみんなできるじゃないか，答えがちゃんと得られるじゃないかという認識があって，そのせいだと思います．反面プログラムを書くのを商売にしている人に対して，生産性をよくしようとか，保守が問題だとか，信頼できないプログラムが多いとかいわれています．

ところで，プログラミングはやさしいという認識からは，生産性をよくしようとか，信頼性を回復しようというようなことは，できないのだと思います．素直に，謙虚に反省する必要がありまして，設計法がはやらないということに対してどうしてかというところで，こう思うべきではないか．ダイクストラやグリースがいうように，プログラミングは大変難しいもので，一生懸命努力しなければいけないと．

で，そこでいい設計法があるんだと，いろいろ教祖様が話してくださいます．しかしその設計法自身も難しい．プログラミングですからこれもやはり大変難しい．それでその難しさを克服したいわけですが，そのためにはどうもプログラマ仲間のエトスが不足しているのじゃないかと思います．タクシーの運転手さんのように個人による組合ができて，組合で仕事をするようになると少しは違うのかなんて思いますが．

これなんかアジテーションですが，企業に依存してやっていると，どうしても居ねむりしてもいいというようなことになるのではないか．落ちこぼれのプログラマが言うべき言葉ではありませんが，そういうふうに思います．

で，実際にどうしなければいけないかというと，設計法が難しいわけですが，それを具体的な課題に適用してみて，結果を仲間うちでいろいろ議論しなければいけないと思います．画家や詩人が同人をつくって，いろいろ批評し議論し合うというようなことがプログラマ仲間にも必要なんだと思っています．たぶんプログラマの職業意識が少し欠けてるというところが一番の理由，いい設計法があってもはやらない理由だと最近考えています．

司会 ありがとうございました．それでは設計技法についていままでお話いただいたことのほかに，日ごろ考えていること，あるいは今回のシンポジウムのいろいろな発表，討論について，パネリストとしてのお立場からであっても，まったく別のお立場でもかまいませんが，それぞれ，もう一度お話を伺いたいと思います．

紫合 それでは「日ごろ考えていること」から．これはさきほども言ったんですけれども，設計方法論というのは，設計の定石などの形を定めたものですが，この形から入って心に達するというのが非常に大切であろうと思っています．これはお茶とかお花とかも同じで，みんな最初に形だけ習いまして，だんだん上達していきますと，その心に達していくのじゃないかと．私はこのあたりの話は全然わからないのですが，日本のいろいろな古典的な作法というのは，心を究め

るために形をきっちりと整えたということがあると思います．設計方法論の場合にも，形をかみくだいて各自の胃で消化をしないとだめだと．そのためには三浦さんのお話しのように初段にならないとだめだというような感じもします．

それと流派がいろいろありますから，今度は統一流派みたいなものが何かほしいなと思っております．さらに日本人に合った方法みたいなものが何かあるのじゃないかと考えています．

この3日間を振り返りますと，まず共通問題での説明は先ほど，山崎さんがおっしゃったような共通の土台で話し合うという意味で非常によかったと思います．発表の中では，やはりデータが前面に出ているとわかりやすいという気はします．構造なんかも絵があると非常にわかりやすい．

関数型，論理型，プロローグ，オブジェクト指向型なども今回多く発表されまして，そろそろこれらも知っておかないと話ができない，同人に入れてもらえない時期かなという気がしました．これまではこの辺のことは言葉を知っているだけでよかったという感じだったんですけれども，もうそろそろ常識になってきたかなという感じがちょっとしました．

実は共通問題を昨日の晩ちょっと考えてみたのですが．私は最初 **図-2** のような絵を描きました．これは我流ですけれども．まわりを見回しますと，多くのプログラマがこんな感じの絵を書くんですね．設計の記述では，このような絵，特にデータの絵が一番わかりやすいと思っています．

この絵で，出荷処理は，出荷依頼を見て，この銘柄の酒をこの数だけ送り先に出します．(OHP の絵を指しながら) ここから見ていって，この品名だけを選んでいく．いま手でやったのですけれども，できればいま手が説明したような感じで仕様が書ければいい．これは非常に楽なんですね．いま「これをこうやって」ということを言葉で書きますと，出荷依頼の品名と同じ品名を持つ酒マスタの各レコードに対して…となるわけです．アニメーション的といいますか，「これをこうやって」というような言い方ができればいいなと思っています．

さて，この処理の中身について，いろいろ考えてみたのですけれども，昨日，二村さんの PAM のお話を聞きまして，じゃあ，スレッドでいこうということでやってみました．最初は問題を非常に簡単にして，出荷依頼と酒マスタから，出庫指示書と不足リストを

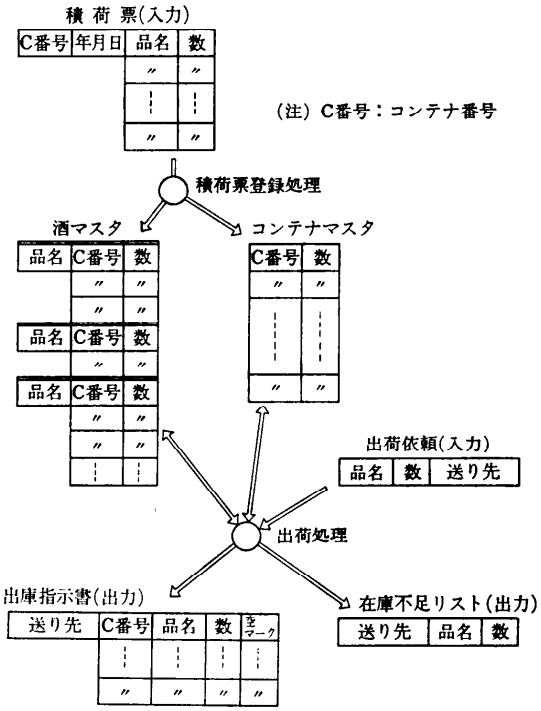
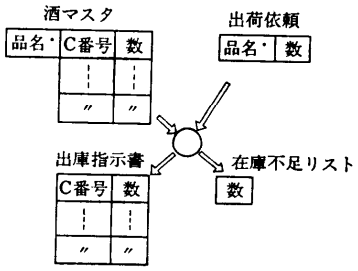


図-2 共通問題のデータ関連図




```
n=出荷依頼.数
get 酒マスタ where 酒マスタ.品名=出荷依頼.品名
while 酒マスタ.品名=出荷依頼.品名 & n≧酒マスタ.数
  | n=n-酒マスタ.数
  | put (酒マスタ.C番号, 酒マスタ.数) to 出庫指示書
  | get next 酒マスタ
end
if 酒マスタ.品名≒出荷依頼.品名
  | 在庫不足リスト.数=n
else /* n<酒マスタ.数 */
  |  put (酒マスタ.C番号, n) to 出庫指示書
end
```

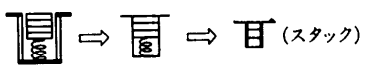
図-3 簡単にした問題の解答

出すということだけを考えます．出庫指示書もコンテナ番号と本数のペアだけという簡単な問題にかえまして，最初のプログラムの骨組みができました(**図-3**)．

```
ファイルオープン
get 出荷依頼
n=出荷依頼．数
get 酒マスタ where 酒マスタ．品名＝出荷依頼．品名
while 酒マスタ．品名＝出荷依頼．品名 & n≧酒マスタ．数
  | n=n−酒マスタ．数
  | コンテナマスタの update(数を酒マスタの数だけ減らす)
  | コンテナマスタの数をみて，空マークを設定
  | put {酒マスタ．C番号，出荷依頼．品名，酒マスタ．
  |     数，空マーク} to 出庫指示書
  | 酒マスタの今読んだレコードを delete
  | get next 酒マスタ
end
if 酒マスタ．品名≒出荷依頼．品名
  | if n≒0
  |   | put {出荷依頼．品名，n} to 在庫不足リスト
  | end
else /* n<酒マスタ．数 */
  | コンテナマスタの update (数をnだけ減らす)
  | 空マークを no に設定
  | put {酒マスタ．C番号，出荷依頼．品名，n，空マー
  |     ク} to 出庫指示書
  | 酒マスタの update (数をnだけ減らす)
end
ファイルクローズ
```

注)　送り先等の出力は省略してある．

図-4　共通問題の解答例



出＋甘 ⇨ 酣　(ポップアップ)
入＋甘 ⇨ 凩　(プッシュダウン)

図-5　新しい漢字のでき方

次に，実はこの酒マスタを更新しなければいけないというので，更新処理を加える．さらにコンテナマスタの方も更新して，空のコンテナのマークを加えていく．このようにして最終的なプログラムを完成させます（図-4）．

最後に，おまけですが．昔は漢字が発明されていました．人がいて木があって，木の横に人がいるというので「休」というような漢字の発明があったわけです．最近は新しい漢字をつくることはやめてしまって，漢字を制限することしかやっていない．けれども，ワープロ等がどんどん出てきまして自分で漢字をつくれるようになっている．そうするとそういう漢字をつくってもいいんじゃないかと．たとえば，スタックの絵から徐々に漢字ができる（図-5）．そうすると，出へんに，スタック，「酣」というのはポップアップで，入へんをつけると「凩」でプッシュダウン．ファイルなんかも，いまはこういうふうに(Q)，書いてファイルになっていますけれども，昔はちゃんとこう(Q)書いたのですね．そのうち，こんなふうに(㔟)書くと，ファイルのオープンという漢字になっていくのじゃないかと．千年ぐらいたたないとわかりませんが(笑い)．

水沢　「日ごろ考えていること」ということなんですけれども，プログラム設計技法というのは，ユーザの立場からいいますとやさしくないと使えないというのが正直なところです．実際にわれわれの会社でプログラムを設計している人間のレベルというのはどのくらいかというのをお話いたしますと，だいたい入社して2年ぐらいたった人間，これは高卒の女の子を含めてなんですけれども，そういう人がプログラム仕様書を書いているわけです．ですから非常に難しい設計手法でありますと，いくらいい手法であっても，そういう人たちに教育して，理解させて，使いこなしてもらうということが非常に難しいというふうに感じております．

したがってプログラム設計手法，いろいろあるんですけれども，だれのためにその手法が使われるのかということを意識した上での設計手法でないと意味がないのではないかという気がいたします．

また計算機のメーカさんにお願いしたいことなんですけれども，計算機のメーカさんというのは，自分のところではいろいろな設計手法を使って効率化されておりユーザにこういうものがあるよという紹介は，かなりしてくれてるとは思っていますが，それが使えるような環境を設定するというところまで，なかなかやってくれない．ツールの提供など環境整備まで含めてぜひとも普及に努めるように努力していただきたいと思います．

このシンポジウムを聞いての感想なんですけれども，共通例題というのが今回設定されているのですけれども，その内容が非常に実務的で，わかりやすくてよかったと思います．通常こういうシンポジウムですと，もうちょっと難しいあるいは抽象的な問題が出されて，それに関して議論されることが多いのですが今回の例題はわれわれからみても日ごろ接しているような問題ですので，具体的にそれぞれの手法がどういう形でこの問題に適用できるのかということが非常にわかりやすかったと思います．

われわれが日ごろ相手にしている問題というのは，実はこの例題とたいして変わらないレベルのものが多いのです．

先ほどプログラムは難しいというお話があったんですけれども，われわれからいうと本当にそうかなという気がいたします．これはあくまでも事務処理という狭い分野に限っての話ですけれども．したがって，この程度の問題を，このような手法を使って，こんな難しくなるのでは困るなという，それが実感という気持ちでございます（笑い）．

三浦　お話を聞いて実際に普及しているレベル，あるいは，私自身が知っているレベルに比べて大変進歩しているというふうに思います．

先ほども申し上げましたけれども，課題は「効果的な普及」ということです．具体的にいいますと，よい教科書，それからよい教育方法ということです．そして教育や普及が効果的にできないような技法とか手法というのは価値がないというふうに思います．

話はまた少し前に戻りますが，今回の課題ではいろいろあいまいな点があったので，こう解釈したとか，いろいろ苦労された例が示されました．私の最初に言いました布石という考え方からしますと，いままで説明された手法では，何か大変更があったときに，全部頭から組み直さなければならないようなことになる．

あるいは大変な手間をかけて修正しなければならなくなるのじゃないかな，というような感じがします．

その一つは，倉庫に残っているコンテナの数を最小化するというところはあんまり具体的には触れられてなかった．

もう一つは，オンラインの端末を使うようなお酒屋さんが，倉庫が一つしかないということはないわけです．だけれども皆さん，倉庫が一つだとして処理している．倉庫別という概念はどこにも出ていない．

先ほど，岡田さんの説明では倉庫という概念が出てきた．岡田さんの流儀でやれば，倉庫が N 個になってもすぐ対応はできそうだということはわかります．しかし他の例では，倉庫がもう一つふえたとたんに，どうやら話がごちゃごちゃしてくる可能性があるわけです．

こういう概念はどう拡大するか，ぱっとわかるのは無理なんですけれども，経験がなくても，こういうところはこうやっておくべきだよというような方法論（布石）がほしいと感じているわけです．

次の「日ごろ考えていること」ですが，技法というのは必要なときに使用できるものでなければいけない．そのためには十分に身についていることです．どうしたら身につけられるかということが問題です．繰り返しになりますが，まず素質があり，素質があって，訓練して，そのときに努力をする．このようなことを全部できるような機会が与えられることが必要です．素質が大切です．

さらに考えなければならないことは，生産性の向上とか，合理化ということですけれども，素質のない人間を，あえて無理やりにソフトウェア生産工程の一部に組み入れて，こき使ってやろうという，そういう精神がもしあるとすれば，これはきわめてけしからんのではないかというような気がします．

彼らは将来どうなるのかということを，だれが責任を持つのか，何百人，何千人と人を採って，適性があるかどうかよくわからないけれども，とにかくこれをやれ，こういうツールと設計技法を使えばできるのじゃないかといって仕事をさせておいて，30歳40歳になってからその人をどうするのかと，私は言いたいですね．先ほど本当に普及する必要があるのだろうかと言ったのは，そういう意味もあったわけです．

もっと高級な技法を，優秀な人間が使いこなすということは非常にいいと思います．しかしさらにやさしくしても，それをだれが使うのか，将来も見て考えてほしいのです．

さらにもう一つ付け加えますと，このような環境にありますと，素質のある人の素質を殺してしまうかもしれないということですね．私どもは幸いにしてそういう気風のない時代にプログラムをやっていたもんですから我流でやって，大いに楽しんでいる．しかも現在も我流で楽しませていただいている．そういうわけで，技法はそんなに必要なのだろうかと考えることもあります．

山崎　共通例題をお考えくださってありがとうございました．またよろしくお願いいたします．

司会　パネリストの方々と，会場の皆さんとの間での質疑応答，あるいは意見交換，あるいはパネリスト同士で他のパネリストのおっしゃったことに関するコメント等，フリートーキングで時間の許す限りやりたいと思いますが，いずれにしろ，まず会場の皆さんからのご発言を中心にしたいと思いますので，どなたでも，どんなトピックでも結構ですから，お手を挙げていただきたい．

久野　東工大の久野と申します．私も共通例題を解

いてみましたが，今回皆さんがおやりになった結果を見て，一つびっくりしたのは，コンテナマスタと酒マスタと，二つのデータベースに分けて解いていらっしゃるのが非常に多いことです．

私は，コンテナの中に酒がいっぱい入っているというデータ構造をつくって，何個ほしいといったら全部その数を数えて出荷するというふうに考えました．たしかにコンテナマスタと酒マスタに分ければ，在庫があるかどうかは，すぐわかるのでその点では優れています．しかし私は事務計算になじみがないせいか，そう思いつくまでには，ひらめきが必要だと思いました．

ひらめかない人に対して，手をさしのべてくれる方法はないのか，あるいは，単純にコンテナは酒の集まりで酒が何個ほしいときはその中身を数えて持ってくるというプログラムでいいのか．その辺はどうお考えかお聞きしたいと思います．

司会 おそらく共通例題の解説，説明をなさった方にお願いするのが一番よろしいかと思います．

山崎さんからお話いただいて，その後できればフロアにいらっしゃる手法の解説をなさった方にもお話いただければと思います．

山崎 たぶん，私どもパンチ・カードのころからのあまりよくない遺産を引きずって歩いているのです．それですぐマスタ・ファイルだとか，トランザクション・ファイルだとかと考えてしまいます．カード・デックや磁気テープを扱いなれていたからです．ですから逆にそれが考え方を制約するような事態になっているのじゃないかと思います．

その在庫マスタだとか，注文トランザクションだとかという思い込みが必ずしもよくないのだという反省はあって，それが算体主導型といいますか，オブジェクト・オリエンテッドに書こうというようになってきて，そうすれば，三浦さんがさっきおっしゃったような変更に対しても，うんと弾力的に応えられるようになるわけですね．過去の考え方にしばられているのであれば早く打ち破った方がいいんだと思います．

司会 ご質問に関して，ほかの技法の解説をなさった方から何かコメントがあれば，ついでに承りたいと思います．

臼井 一番最初に SP フローを使って，データ構造に基づいてフローを変えていくのだというご説明をした臼井と申します．先ほど言われましたとおり，中間にどのような形のファイルを持ってくるかということを最初から定義してしまったのでは，設計法の意味はやはりないと思うんです．ただし，積荷表を入力して，その次にいろいろなパターンで出庫していくわけですけれども，いったん，機械の中へ何らかの方法で蓄えなければならないという事実だけはあると思うんですね．

そうするとどういう方法で蓄えたらいいかということが問題になると思います．それはやはりマスタという名前をつけようと，メモリの中へストアしておこうと，いくらたまっているかというのは絶対に必要な情報ですから，それをいかに持つかということだと思うんですね．

私の場合はプログラムを見せながらお話した方がいいんでしょうが，単なる在庫マスタという形を，まず定義しておきまして，とりあえず同じ商品がたくさんあるわけですから，その商品か，それからコンテナもたくさんあるわけですから，どちらかを含むような形で，テーブル的にとる必要があると思うんですね．

商品とコンテナの関連さえ示せばいいわけです．それをデータ構造で示しますと，どちらを親にして，どちらを子供にするかという，二つの考え方が出てくる．そうするとコンテナの中に商品が入っているという形で，中間エリアに持っておくか，それとも商品をベースに持っていて，その中にこの商品はどこのコンテナに入っているかという持ち方をするかの二通りになると思います．

で，私の場合どちらでもその設計においては使えるわけです．どちらを使った方がそのデータ構造に基づいたフローがすっきりするかというのは簡単なものですから，二種類書いてみて，アウトプットの出力関係がすべて商品でコントロールされるような形で取り出すように指示されていましたので，商品を大きなキーにして，その中にコンテナを入れた方がフローが簡単になるだろうと．あとからどちらがいいかなと考えるような形で設計してみたのですけれども．

山崎 三浦さんや二村さんのご指摘があったように，倉庫，コンテナ，酒それぞれをプロセスだと思ってしまうやり方があります．そこでは倉庫もコンテナも酒も，それぞれが一つのプロセスで，それら全体が互いに通信をしあいながら現実と同じように動くのだと思うことになります．

より具体的にいえば，たとえば，communicating sequential processes で実現する．

このような考えがだんだんと効率もよくリアルタイ

ムの環境で動くようなものができるようになってきつつあると思います.

久野　もう少し補足させていただきます. 問題文を読むと, 倉庫があって, その中にコンテナがしまってあって, コンテナの中にお酒がいくつかあるというイメージが浮かびます.

それをそのままプログラミングすると, さっき言ったような解になる. その後データ構造を用意して, 表にすべきかどうかという点は, 能率が重要, またはトランザクションが多いなどの条件下では必要と思います. しかし処理能力が大きく, 仕事の量が少ない場合には, 素直に考えたほうでプログラムとしても自然だし, 保守もしやすいと思います.

山崎さんがお話になった点でも一番過激な案は, お酒は全部生きてると考えることです. ある銘柄の注文がきたら「この銘柄のお酒は皆出てきてください」というと, その銘柄のお酒が皆出てきて順番に並びます.

そこで最初の何個分かには「あなた方は出庫されます」, 残りには「コンテナのところに帰ってください」というとその通りになる. その後, 「空いたコンテナがあったら出てきてください」というと空になったコンテナが自分で出てくる. そういう言語であればそうプログラムしてもいいはずです. しかしそのままでは現在の計算機でうまくいかないかもしれない.

そういうときに技法が役立つのではないか, うまいひらめきがなくても, 使えるプログラムになってくれるのか, という点を聞きたかったのです. もし表にするのならさっきおっしゃった通りだと思いますが, 必ず表にしなければならないとは思いません.

久保　IBM の久保です. お答になるかどうかわからないのですけれども, 実際私は課題を与えられて解くときに, いろいろ努力しまして失敗した過程も書いてあるのですけれども. それは現実のプログラムを書くことを考えますと, そこにあるのは, まずコンピュータ, どれぐらい大きくて, メイン・エリアとしてどのぐらい使えるかということと, どの言語で書こうかというのが無意識で頭の中に出てくるような気がするのですね.

まず私は, PL/I で書くのでだいたい書くわけですね. とっさにこういう問題で, 品名ごとに並べるか, コンテナ順に並べるかと考えますと, ポインタを考えたわけです. ポインタで考えて構造をつくって, やれプログラムにしようと思ったら, とてもじゃないけれども, 一つのコンテナに10品目ありますので, 200種類のお酒の銘柄があって, それでエントリ・ポインタをつくってと, それでやったもんですから, にっちもさっちもいかなくなりまして, これは何がおかしいかと.

そこで少し反省をしまして, どうも在庫問題を知らないのではないかと. そこで少し回りを歩きまして, いったいこういうときはどうするか, いろいろ聞いて回ったわけです. それで私の発表で, どういうふうな形でこういう設計技法が学べるかという質問に対して, ある程度事例によって型を学ぶべきではないかという答をしたというのは, 私がそういう形である問題を解いたもんですから, ついそういう答になったわけですけれども, 結局, じゃどうするかというときは, 私が使おうとした PL/I がどういう能力があるかということと, 私はあなたと同じように, あるかどうかというのをより分けまして, あるオーダがくるたびにより分けてソートしたというわけですね.

200ぐらいの品種で, コンテナの数もそう多くないのであれば, メイン・エリアでばっとソートしてもそうたいしたことはない. まわりの皆がやっているもんですから, ついそれが浮ぶわけですね. でも, メイン・エリアで単純にソートできないエリアしかないというパソコンで, そうやったかというのは, やはり問題として残ると思うんです.

だから一般的な技法として, どういうのがいい, 悪いというのはありますけれども, 実際問題解くときはどういうコンパイラを使うかということと, コンピュータの能力とか, それからそこで使えるメイン・エリアの大きさとか, そういうものを無意識に考えるということと, やはりまわりで少し知識を仕入れるのじゃないかと私は思います.

久野　付帯条件を考えながら問題の解を考えるべきだ, ということですね.

久保　無意識に出るのじゃないかなと思うんです. それとやはりまわりで聞いた方がいいと思いますけれども.

水沢　ちょっと補足させていただきたいのですけれども, 先ほど山崎さんの方から, 無意識にマスタとかトランザクションとかいうことを考えてしまうということを言われましたが私も同じです. どうしてかといいますと, この例題は, 本来この範囲の問題というように限定して理解しなければいけないのでしょうけれども, 通常こういう業務のシステム化を行う場合は,

このような在庫管理の仕事だけではなくて他の仕事もいっぱいつながっているのですね．そうなりますと，いろいろなプログラムから，一つのデータを見にいきたいということになります．このようなプログラムとデータの関係はプログラム設計より前の，システム設計という段階で考えるわけで，システム設計のアウトプットというのが，マスタ・ファイルとか，トランザクション・ファイルとかいう形で出てきてしまう．ですからプログラム設計に入る前にそういうのがきまっているというのが，いままでのやり方の前提になっているわけです．したがって，そういう習慣がついてしまうということだと思います．

この論文集の中でもいままでのそういうステップをくつがえしたやり方を提案されている人もございますけれども，そこまで考えて初めて，マスタとかトランザクションという概念を使わないということが可能になってくるのではないかと思います．

原田 電力中央研究所の原田です．プログラム設計技法に関しまして，一昨年1年間にわたり一般企業のプログラマや SE の方々約20名と研究会をやったことがあります．

約20個の例題を集めまして，それらをジャクソン法，ワーニエ法，構造化設計法それに再利用化技法の四つの技法で設計しました．ここで構造化設計法は複合設計法の原典にあたるもので，また再利用化技法はプログラムをパターン化して登録し，それを再利用しようというものです．この方法は通常のデータ処理では有効で多くのユーザ企業で行われつつあります．

さて研究会ではこのような比較研究の成果をまとめて報告書にしたわけですが，そのときの経験を少し申しますと，構造化設計の適応分野は，モジュール分割段階です．したがって，酒倉庫の問題でありましたような，あの程度の小規模な問題では，これをさらにモジュールに分割していって，構造設計法を適用するには問題が小さすぎると思います．

言い方をかえますと，プログラム設計を二つの段階，すなわちプログラムをどういうモジュールで構成するかを定めるプログラム構造設計と各モジュールの詳細な処理手順を定める詳細設計に分けますと前者の段階には，構造化設計というのは非常に有力であると思われます．実際私が Ramamoothy とか，Raymond Yeh に聞いてみましても，「アメリカでは構造化設計というのは非常によく普及している」と，そういう意見をもらっています．

一方ジャクソン法とワーニエ法，これは当然，明らかにプログラムの詳細処理手順を設計する場合に適用する設計技法なんですけれども，基本的に両方ともデータ構造からプログラム構造を導き出す方法論でありまして，考え方はほとんど同じなわけです．ただワーニエ法の方がプログラム構造を導き出す基準が法則という形で非常によく体系的にまとめてありまして設計者を誘導するという意味におきまして，非常によくできているのじゃないかと思いました．

またワーニエ法の場合は，フェーズ化という概念がありまして，プログラムをいくつかの処理フェーズに分けてモジュールにすることもできるわけです．

一方ジャクソン法では，プログラム構造を入力データ構造と出力データ構造の対応関係から決定します．したがってこの間の対応関係がうまくとれないときはジャクソンはこれを構造不一致問題と名付け，順序不一致，境界不一致，脈絡不一致の三つに分類しそれぞれの場合にどういう具合にして解決したらよいのかを議論しています．具体的には紫合さんもおっしゃったように，ソートするとか中間ファイルを使うとかプログラム・インバージョンを行うとかということです．またバックトラッキングについても議論しています．

要するに，データ構造主導型の設計技法を使った場合に起こり得る実際的問題に対して，対処法が非常にていねいに議論されてるという点がわれわれ研究会のメンバの評価結果でした．

最後のパターンを再利用するというやり方ですが，これは設計技法というよりも，つまりプログラムを設計するというよりは，できているものを再利用するということでありまして，たとえば，この酒倉庫の問題にしますと，もしトランザクションがソートされない形で，リアルタイムに入ってくるような場合ですと，マスタ・ファイルなんかもダイレクトアクセスにしてしまえばダイレクトアクセスを使った更新パターンを利用できますし，そうでない場合も順ファイルを使った更新パターンとか，そういうものを使えばできるのじゃないかと思います．

以上が設計技法に関するその当時，1年間ぐらい研究会をやったときの私の感想なんですけれども．

あと一つ，それ以降いろいろ世の中の EDP 処理に関するツールなんかの進展具合を見てまして，今回のシンポジウムには発表されてなかったんですけれども最近非手続型の超高水準言語，というのが EDP 用にいくつかできているわけですね．

それは，たとえば，三井造船が最初につくった，データ・フロー・コボル，これは富士通からも，HYPERCOBOL という形で出てますし，日電からは，IDL という形で出ています．ここでは命令の単位が従来のプログラム言語レベルのものでなくて，照合やチェックなどもう少しマクロ化した機能を持っています．このマクロ命令を組み合わせて一つの処理を記述していくわけです．また別な取り組み方として日本システムサイエンスの JASPOL というような，非手続型の超高水準言語もあります．

ここでは，ワーニエ法とか，ジャクソン法でいう入出力データ構造間の対応関係を，データ宣言時に対応関係を表わすラベルを立てることによって表現し，照合とか，更新における入出力のタイミングをとるコードを自動生成するわけです．こうすればプログラムで記述しなければいけないのはデータに対する変換だけになってしまうわけです．したがって従来の設計技法が導出していたプログラム構造が自動的に生成されるわけです．

つまり記述が非手続型になるわけです．そういう言語を使って処理を記述しますと従来あったようなワーニエ法とか，ジャクソン法という設計技法というのは役に立たないというか，必要なくなってしまうわけです．「時代はかわっていくのだな」というのが，私の最近の実感です．

司会　ありがとうございました．原田さんのお話は独自の立場から，いろいろな技法の評価をなさったということだと思いますが，いまのお話に対して，何かコメントなり，何なりあれば一．

紫合　いまのお話と，山崎さんのお話にちょっと関連してコメントしたいのですが．問題によっては，オブジェクト・オリエンテッドや非手続型が非常にいいと思います．だけど非手続型が出たら，もうジャクソン法はだめかというと，私はそうは思わなくて，ものによってはジャクソン法の方がずっとわかりやすいと思います．

たとえば，私はお酒が勝手に動いていくよりも，倉庫番のおっさんがお酒を持ってくるというような感じの手続的な処理の方が，ピンとくるわけです．たとえばデータにしても，倉庫があって，倉庫の中にたくさんのコンテナがある．コンテナの中に，いろいろな銘柄の酒がある．そういうデータ構造でやればいいじゃないかと，私も最初はそう考えました．

次に，銘柄に対して，何本かの酒を倉庫番が持ってくるわけですが，いつもコンテナの中をのぞいて，その銘柄があるかないか見て，なければ次の所へ行ってということはしない．おそらく最初に帳簿を作っておいて，どの銘柄の酒は，どのコンテナに何本入っているかということを記録しておく．その帳簿を見ながら酒をとり出すであろうと思うわけです．

次に，銘柄によって選び出すので，銘柄を最初にキーにして選べるような帳簿があればいい．さらにコンテナが空になったら，それを運び出す必要がある．そのためにはコンテナをキーにして，そこにどれだけ入っているかという別の帳簿がいるだろうと思いました．一つの帳簿をコンテナ別に並べ替えてみればいいだろうとも思ったのですが，出庫のたびに並べ替えるのは大変だということで，コンテナをキーにした帳簿も持っておくことにしました．

コンテナをキーにした帳簿は，コンテナが空かどうかを調べるためだけだから，コンテナに入っている酒の個数だけを持っていればいいわけです．

おそらくこんなふうに，酒屋の倉庫番のおっさんが帳簿を見ながら作業するつもりでプログラムをつくるという，私はそんな感じで，比較的自然にできると思うんですけれども．

司会　まだ 5，6 分ありますので，いままで発言しておられない方に積極的に何かいただきたいと思いますが．

関　鉄道技術研究所の関です．ちょっと話題を変えまして，これからの設計技法の将来とか，そういうことについて，パネラのご意見をいただきたいと思うんですが，先ほど三浦さんから，初段になる人は少ないとか，素質のない人をやってもしようがないのじゃないかとか，あるいは，紫合さんからは，10年たったら少しずつ使われてくるよというようなお話もありましたけれども，私はちょっと見方を変えまして，これから日本でいいと思いますけれども，世界的にソフトウェアの需要がどんどんふえてくると．これがつめに対して生産性が上がってないからどうのこうのというのが，ソフトウェアの危機とか，こういったソフトウェア工学が出てきたということのもとになっていると思うんですが，いまのままで進んでいいものかどうか．

そういうソフトウェア危機がいわれてから 10 年以上たっていまして，その後いくらか進んだのか，相当進んだのかは評価はあると思いますが，最初に思ったほど進んでないとしても，何とかやってきたと．何と

かやってきたソフトウェアの技術が発展しないからといって，それによってこの分野の進歩が妨げられたとも思いませんし，これからは老人問題とか，いろいろ出てきますので，老人でもできるようなやさしいプログラムをやればいいか，あるいは，あんまり生産性を上げてしまうと，やる人がいらなくなってしまって，皆，失業してしまうのじゃないかとか，いろいろなことがありますので，皆さんのご意見をお伺いしたいと思いますが．

三浦 いままでこういうような状況であったから進歩が妨げられたかどうかという話ですが，私は十分進歩していて，過去について何ら異論はないわけです．

問題は，いまのお話の中にありましたけれども，ソフトの需要がふえるから，それに比例して人間もふやさなければいけないという発想法が不都合なのではないかということです．

需要というのは，なぜ発生しているのかということも考えなければならないわけですし，必要なものにすべて応えるのが一番いいのかということも考えたいのです．

ですから私は酒と倉庫との関係でも同じかもしれませんが，人間の方を見るか，表面に現われたソフトの需要を見るかです．だれがつくるかということは問題だと思います．

将来のソフトウェア技術，ないしは，ソフトウェア工学の進歩に関連していえば，だれが，どういう立場で使うのかということを，はっきりさせておくということが必要なのではないか．そうすれば使う方も，それなりに使うということもできますし，それからまた，もう少し普及も無駄なくできるのではないかと考えます．

どういうソフトウェア技術ができたらいいかというところまでは意見を申し上げられませんけれども，はっきり，だれが，どういう場面で使うのかということを定義した上で，かつ，それに見合った教育方法もつけて用意していただけると具合がいいんではないかと考えております．

司会 ありがとうございました．いまの三浦さん，関さんのお話に関連して，何かおっしゃりたい方ありましたら．

紫合 私は時間がたてばジャクソン法であろうと，難しいものであろうと，初段までいかなくても，結構，皆がいいものであれば無意識で使うようになるのじゃないかと楽観してますけれども．

そのうち無意識にオブジェクト・オリエンテッドとか，関数型とか，そういうのも使えるようになる．時間がたてば皆いい方式でやるようになるのじゃないかというように思うのですが．

三浦 時間に対して，自然にやっていったのでは間に合わないのじゃないかというあせりが問題の源泉のようですが．

関 高級なことはいらないのじゃないかと．

紫合 高級かどうかというのは，そのときによるという感じがするのですが．いまじゃ，サブルーチンという概念は全然高級じゃない，三基本構造という話も高級でなくなってきているという感じがします．時間がたてばというふうに．教育を頑張る必要はありますが．

司会 ありがとうございました．まだ議論するトピックはたくさん残っているかと思いますが，時間になりましたので，大変残念でございますけれども，「プログラム設計技法の現状と将来」というパネルをこれで終了いたしたいと思います．

パネリストとして発表してくださった方，及び，質疑討論に参加してくださった会場の皆様方に謝意を表して，全員で拍手をいたしまして閉会したいと思います．どうもありがとうございました．

(拍手)