

座 談 会

ソフトウェア・エンジニアリングをめぐる

石 崎 純 夫 石 田 晴 久 上 条 史 彦 首 藤 勝

(株)富士銀行

東京大学大型計算
機センター(特)情報処理振興
事業協会

三菱電機(株)

三 浦 大 亮 山 地 克 郎 吉 村 鉄 太 郎 (司会・編集) 米 田 英 一

東レ(株)

富士通(株)

(株)管理工学研究所

東京芝浦電気(株)

(1975年3月24日 於機械振興会館会議室) (氏名左上から右に50音順 敬称略)

米田(司会) ソフトウェア・エンジニアリング特集号で、いろいろ解説記事が出るのですが、解説には書けないような生々しいお話しをお話し頂きたいと思います。

まずオンラインなどビッグシステムの開発のあたりからお話し頂きたいと思いますが。

ビッグシステムの開発体制

石崎 私共の方では、リアルタイム・プログラムが非常に大きなウエイトを占めており、現在走っているプログラムは、220支店、900万口座を対象に、40万ステップからできています。これが一本のプログラムに合成されているため、開発もメンテナンスも大変なわけです。パッチですと、ある程度細切れにデバッグや開発もできるわけですが、当座預金、定期預金、普通預金、通知預金等、預金の全科目、それに内国為替、外国為替、貸付、手形割引といったものを全部一本のリアルタイム・システムに乗せなければならないので、開発に何年という年月がかかると同時に、常時百人以上の人間を投じて開発せざるを得ないわけです。しかも各々の連携動作がうまく保たれていないと、誰かのバグが他の人のプログラムに影響して来ということで、非常にチームワークが要求されるわけですね。

リアルタイム・システムの開発にとりかかったのが昭和37年で、41年からテストを始めて、42年から実施して現在に至っております。49年からコンピュータの機種を切替えつつあるところですが、こうした膨大なプログラムの切替えは非常に難しい作業で、飛行中の飛行機から別の飛行機に移り移るようなものだという人もあったくらいです。リアルタイム・システムですから、土曜日の12時過ぎ位までは旧システムでお

客さん进行处理して、月曜日の朝9時からは新システムで処理しなければならないわけです。

米田 旧システムで処理している支店と、新システムで処理している支店が共存しているわけですか。

石崎 そうです。預金システムについては、毎月数店ずつ旧システムから新システムに、2年位かけて切替えて行くわけです。一方、為替システムになりますと新旧併行ということはあるので、ある日一斉に全店を旧システムから新システムに切替えなければならない、これはまた大変な作業です。昭和44年に正月の連休を利用してこのような切替えをやったことがあります。

米田 メーカーの協力体制はどうですか。

石崎 大変協力してもらっていますが、それでも大変でしょうね。

米田 切替えに伴う変更量や、開発期間の見積りはかなり厳格に要求されているわけですか。

石崎 これはメーカー側の人達に、機種決定の前に一年半、プロポーザルを作ってもらった段階があったのですが、その間に繰返し繰返しディスカッションをやっています。メーカー側から頂いたプロポーザルの厚さだけでも50センチ以上になった位です。

米田 石田先生のところではいかがですか。

石田 私の所のOSは数十万ステップの巨大なもので、システム自身はメーカーの方で開発し、できましたといって持ち込まれたものを、我々が使ってみるわけですが、問題点はやはり虫がとりきれていないということですね。メーカーが自分でやるテストの状態というのは、ユーザが実際に使っているときの状態とは違う。メーカー側で十分テストができないことがはっきりしているわけで、ユーザが使うと虫が出る。

米田 メーカーの立場で山地さん、いかがですか。

山地 やはり高名の方が言われているように、テストというのは、バグの存在の証明はできても、バグがないことは証明できないということを実感として感じています。テスト方法も、作成技術も進歩しなければいかなはずですが、現実には性能を厳しく追求されるために、やはり読みにくいアセンブラも使うことになる。そういうことを改善して、読み易いプログラムを、間違いにくい言語で書くようにして行きたい。試験方法／内容も経験を反映して充実させてきたと考えていますが、完璧は期しがたいという所です。

米田 高級言語も使っておられるわけですか。

山地 部分的に実験も兼ねて使っていますが、中級言語を別にすれば、実用には先程の問題を抱えている段階です。

米田 吉村さんの方はいかがですか。

吉村 我々の所は大規模ではなく、10万ステップ以下というスケールが普通で、仕事としてはシステムプログラムや言語プロセッサなど、ある程度の規模の大きさのものが多くあります。このクラスの仕事ですと、そんなに沢山の人数はいらないわけで、十人以上になるとかなり多いという感じです。もう少し正確に言うと、チームワークをうまくとるためには、できるだけ有能な人なるべく少数使うのが開発段階ではいいし、そうやってできた製品は品質も安定するし、保守もし易いのではないかという気がします。

米田 チーフ・プログラマ・チームなどは？

吉村 ああいうのはやっていません。

少数精鋭主義

米田 規模によっては少数精鋭ではどうにもならない場合もあると思いますが、石崎さん、いかがですか。

石崎 プログラムの複雑さによって、夫々何万ステップかにクリティカル・ポイントがあるのではないかと思いますね。もう一つは、私の方ではリアルタイムをやっているのですが、最初から200人いてもしょうがないわけです。最初はごく数人のベテランのシステム・エンジニアが構想を次第に具体化しながら、3人ふやし4人ふえて結局200人位になったわけです。最初のシステム分析の段階は数人でいいが、その代り最優秀の人間でないと難しいのじゃないでしょうか。

米田 トップダウン・アプローチということが言われていますが、富士銀行さんではいかがですか。

石崎 結果的に見るとそうなっていますね。

米田 三浦さんはビッグシステムに対して批判的であると伺っていますが、いかがですか。

三浦 少数精鋭でできる程度のシステムでやめておくべきだというのが、私の基本的考えなんです。大きいシステムを多勢の人が作ってでき上ったものは沢山存在するわけで、それはそれでよいと思うのですが、その柔軟性といったものについて疑問を持っているわけです。これから世の中が変化した場合に、それがうまく対応して乗り切れて行けば、私の意見は間違いであったということになりますが、そういったことについて非常に神経質に考えた場合には、少数精鋭でできるような小さなシステムの集りでできるような仕事のやり方を考えた方がいいのではないかということです。例えば、10個のシステムで1個の大きなシステムができていけるとすると、その10個の中の1個がトラブルしても他に影響しないようなインターフェイスを工夫することが大切だと思います。

石崎 少数精鋭でできるようなアプリケーションもあれば、逆に多勢でなければできないというアプリケーションもあるわけですね。銀行の場合は、例えば、為替で振込まれて来た資金は預金に振り替えられなければならない。しかも、それは当座預金かも知れないし、普通預金かも知れない。いろいろなオペレーションが全部リアルタイムにリンケージされていないと困るわけです。個々のアプリケーション一つずつを取上げていけば、全部モデル化されて独立したパッケージになっているわけですが、そういうリンケージをとって行くと全体として40万ステップになるということであって、逆に言うと、我々のクラスの規模の所で少数精鋭で独立したプログラムを10本か20本作ってそれで商売ができるかという、それはできないわけです。

三浦 石崎さんの言われた、いろいろなリンケージですけれども、それが本当にリアルタイムでやらなくても支障がないようにできないものかなということです。例えば一つのアプリケーションから他のアプリケーションにうまく移すのは、夜のバッチ処理で翌朝までにやって、というのが従来のやり方だったと思います。

石崎 それができれば問題ないのですが、900万口座もあると、例えば、今この手形を割引いてもらわないと小切手や約束手形が決済できないというお客さんが何人もあるわけです。仮にそういう口座が100口座あったとしても、夜中のバッチでやるということでは

きないわけです。三浦さんが言っておられることはアプリケーションによって、随分違うと思います。およそコンピュータ・システムは少数精鋭でやるべきだとは言えないのであって、少数でできるケースもあれば、リアルタイムで、一つの大きなシステムを多勢で作る必要のある場合もある。

三浦 もちろんそれはそう思います。逆に僕らの考え方としては、やらなくても済むものまで、一見便利だというだけで大きなシステムにすることの危険性をよく考えた上で、システム設計をした方がいいのじゃないかと思うわけです。大きなことがよいことだとは限らない。

米田 少数精鋭について上条さんいかがですか。

上条 現在私の所でやっているプロジェクトは約40件で、数万から数十万ステップの間です。

今のお話と関連して最近の傾向を考えますと、ファイルのネットワーク体系をとる設計が多くなっています。データ・ベース手法ではある意味での階層構造を基本にしているのですが、それを利用手法と組合せると、殆んどの場合、ネットワーク的な使い方をしないと一つのシステムとしてまとまらない傾向があるわけです。従って、トップダウン・アプローチもある意味では、私は疑問を感じています。製造段階ではトップダウンで、きれいに階層構造的な作業に分かれて行くでしょうし、システムそのものもサブシステムに分けられるような気がするのですが、ある程度分けてしまうとその間の情報のやりとりということが当然出て来る。通常これをファイルを通してやるのですが、その結果ファイルはネットワーク的な使い方をされてしまうことになる。プログラムだけでなくデータも考え合わせると、サブシステム単位に独立性を保って行くということがだんだんできなくなって行くと思います。

米田 首藤さんいかがですか、メーカ側として。

首藤 トップダウン手法というのは比較的大きなシステムに対する手法として出て来ているわけですが、私のところでは小規模なシステムばかりですが、トップダウン手法を少しやってみた範囲では、上条さんがいわれたように、プログラムそのものを階層的に組立てるということと、一方プログラムに関係するデータをどのように構成するか、という問題が別個に存在するという、その辺りの問題に突き当たっている所です。

石崎 三浦さんの話に戻りますが、抽象的な話としては、小規模のシステムで組むようにすべきだとい

ことは分るのですが、具体的なアプリケーションをよくお調べになって、その上でブレークダウンできるという形で御指摘頂かないと、説得力に欠けるのではないかと思います。抽象論としてはブレークダウンすべきだということは分りますし、我々もそうしたいのですが、モジュールのリンケージというのは、今のような商取引の中ではやらざるを得ない。またそのようにしなければ、お客さんが満足してくれないというところに難しさがあるのですね。

システム仕様の問題

山地 僕もある意味で、今の御意見に同感なのです。ソフトウェアの場合には、モジュール化なりブロック構造なりをつめて行くのは、確かにいいアプローチだと思うのです。“インターフェイスまずありき”ということでスタートするならば、かなりそれに近い筋に行くと思います。ところがハードウェアの場合には、チャンネル・インターフェイスがどうであるというようなことが、既にあって、それに合うようにハードを作って行けばいいようになっているのですが、ソフトウェアの方は、インターフェイスをどうするかを考えると同時に、モノを作って行くというか、機能分担が不明確なまま進んで行くのですね。結果として、やってみないと分らない。

米田 お客さんとの間で、仕様がなかなか決まらな

いままでも走り出すことがあるのではありませんか。

吉村 現実問題としては、開発方式が問題になるような大きさのソフトウェアですと、かなり馬力を上げて、最低半年、一年からそれ以上になると思うんです。そういう長い開発期間の場合、最初に凍結した仕様が最後まで全部意味を持つことは、ほとんどあり得ないわけですね、作成者がいくらそうしたいと思っても、依頼者にとって意味のないものが出来上がったのでは仕方がないので、どうしても途中でかなり大きな仕様の変更をせざるを得ない場合がシステム・プログラムの場合でもあります。おそらく大型のアプリケーションの場合はもっと深刻にあるだろうと思います。そういう場合に教科書的なトップダウン・アプローチというのは、ナンセンスとは言わないまでも、それによって問題が解決するだろうとは思えない。むしろ、そういう静的なやり方ばかりではなく、もっと開発環境が動的に変化する場合でも、それにきちんと追隨して行けるような人なり、開発のための道具なりがある方が、大事だと思うのです。

メンバーの交代の問題

米田 環境の変化の中には、開発期間中のメンバーの交代、例えばあるモジュールの担当者が急に代ったというようなことはありませんか。

石崎 あまりありません。私の所では、一旦コンピュータ部門に入ってしまうと、支店や本部に行くということはあまり行われていません。ただ、コンピュータ部門の中では、本人があまり飽きないようにして分担の変更をするというようなことはやっています。

米田 銀行によっては、コンピュータ部門から外へどんどん出すところもあると聞いていますが。

石崎 そういうこともやってはいるのですが、膨張につぐ膨張でなかなか余裕がないという感じです。

米田 山地さんの方は、人数も多いので、交代はかなりあると思いますが、引き継ぎはどうされていますか。

山地 普通は作業を引き継ぐ期間を1ヶ月から3ヶ月、大きさによって違いますが、数ヶ月間とるのです。頭の中に残っているものだけしかないとまずいので、とにかくドキュメントを残せと言っているのですが、その書き方が問題でして、個性が非常に強く出る。なるべく個性を出してはしくないのですが、現実にはそうは行かない。そういう意味では構造化プログラミングは非常にいいと思っています。自分の思考過程を記録に残しておくことは、仕事の引き継ぎを考えると非常にいい。しかし、あれには相当の訓練が必要でしょう。いいからすぐやれるかというところはいかない、徐々にそういう方向に持って行きたいと思っています。

納期の問題

三浦 僕の所で最近稼動しはじめたオンライン・システムの場合には、たまたまそういうことができたアプリケーションかも知れないです。かなり少数精鋭でやって、幸か不幸か、仕様が決まるまでの期間が長かったのです。その間に、やらなければならないことを徹底的に調べることができたのです。それで、単体モジュールを沢山作りまして、変化があってもそのモジュールだけ取替えればよいということで、モジュール化を徹底したわけです。仕様がなかなか決らないことを覚悟した上で、そういうアプローチをとった。決ってから稼動するまでは、なるべく短期間でパッと作り上げ納期通り稼動した。

米田 少数精鋭というのは何人位ですか。

三浦 大体 10 人位ですね。ですから準備期間が非

常に長かったということが、逆によかったようです。

米田 その場合、開発の途中で人が足りないといって、人を追加することはしないのですか。

三浦 それはしないのです。

米田 360 の設計者のブルックスは、途中からいく人々を注ぎ込んでもダメだと書いていますが。

石崎 途中からではダメだということはないと思います。最初から 40 万ステップのプログラムを組む人間をつぎ込んでバカバカしいわけですね。

吉村 あれは納期に遅れそうになって、慌てて人間を注ぎ込むのはよくないという意味でしょう。

石崎 逆に言うと、銀行の場合、新しいアプリケーションを何年何月何日からやらないと、お客様が一斉に他の銀行に行ってしまうということはないわけですね。

上条 私の所は納期が絶対なのです。国の予算制度の関係で3月31日から4月1日に移れば、お金がなくなってしまう。ところが、普通のやり方をとっていると、ドキュメントは絶対に納期通りにできてこない。納期一杯ぎりぎりまでプログラム作りに追われるのが実情ですから。そこで一計を案じまして、昨年からは、仕様が後から変えるということは承知の上で、ドキュメントを先に作らせ、とにかく体裁を整えて、プログラムができ上ってから、レビューしてもらって現実のものと合わせることにした。こういう順序にしてから納期が守られるようになりました。もう一つは、ある分野の開発を初めてやる場合と、二度目とでは、プロジェクトの内容や見積りの内容が非常に違うということです。最初は試行錯誤が多くて、いつまでたっても仕様が決まらない、仮りの仕様を作っても途中で何度も変更されて行くわけです。二度目になりますと、最初にドキュメントに三分の一位の期間をかけて固める、それからプログラム作りに入って、最終段階に入ってデバッグをやりながらドキュメントを見直す。これで納期に納まります。

コストの見積り、工数の見積り

米田 コストの見積りは、最初にやるのですか。

上条 一種の競技設計方式ですから、競争的要素は非常に多いのですが、私の所はおおむね類似品との比較です。

米田 吉村さんの所ではどうですか。

吉村 我々の所では、ほとんど一握りの人間が経験をもとにして、いろいろな意見を参考にやってい

る場合が多いですね。それが習熟して行くと、競争入札に勝つ場合もあるわけです。むしろ問題は、そこから後の、見積った範囲内のコストにいかにか納めるかに頭が痛いわけです。

上条 実際問題として、コストの見積りは非常に難しい。一致させるにはタイム・アンド・マテリアル方式しかないということが言われていますが、現実にはタイム・アンド・マテリアルは少ないと思うのです。大抵は上限をつける。その上限をどういうふうに定めるかで、我々はいつも悩んでいるのですが、二つのアプローチがあると思います。一つは厳しい方法で、そのソフトウェアが、どれだけ世の中に受け入れられるか絶対評価をしようという立場がある。もう一つは、コストの積み上げで比較する方法です。私の所はソフトウェア・ハウスではないので、自分の所では作りませんが、理想をいいますと、自分の所で製造部門を持っていて、そこで積算したらいくらになるという値段があれば、それと仕様を突き合わせることができる。

三浦 費用の見積りよりも、プログラムのステップ数とか、メモリ・サイズとか、オンラインのトランザクション当りのステップ数の見積りがなかなか当らない。処理能力とプログラム・サイズ、この見積りが致命的ですね。

米田 三浦さんの所はうまく行っているのですか。

三浦 うまく行っていないので困っているのです。過去にも困ったし、今も困っている。こうなると、先ほどのモジュール化ということも、そのとたんにぶっこわれてくるのです。せっかく整理してあるが仕方がないというので、もう少しくっつけてメモリを減らすとかそういうことで非常に大変なことになって来ます。まず、分り易いストラクチャを作って、方針をごまかして速くしろということになるわけです。

ドキュメンテーション

吉村 ドキュメントのことで最近痛感するのですが、基本的には適切なドキュメントが必要な時点で出ること、品質が安定するということは確かに否定できない。ソフトウェアが大きくなれば、そういう必要性は益々高まる。ところが今の日本における一般的傾向としては、ドキュメントをきちんとすることが、どの位製造者側のコストと工数に反映するかについての発注者側の認識が極めて少ない。ドキュメントを本当に真面目にやると、かなりの場合作成側は赤字になってしまう。そうすると、中には、ほどほどのところ

でドキュメントの体裁だけ出すということが起ってくる。これは作成者側、発注者側いずれにとっても不幸なことだと思います。

米田 石崎さんのところでは、ドキュメントが多いと言われましたが、どの位活用されているのですか。

石崎 それを読まないで 40 万ステップのプログラムは書けないと思いますよ。全部に目を通す人間は数人だと思いますが、部分部分については、担当者が隅から隅まで読んでいないと作れないと思います。

石田 ソフトウェアは、ドキュメントに縛られて、よけい固くなる傾向がありますね。ユーザ側がメーカに対して、あるソフトウェアの改造要求を出すと、それをやるのは何でもないが、ドキュメントを直すのが大変だからということが時々ありますね。ドキュメントを変更するのが、いかに大変であるかということをしみじみ感じているのではないかと思います。

米田 そういう場合、東大ではどうするのですか。

石田 その場合は、力関係で。(笑い)。それとユーザがいろいろあった場合に、メーカとしてはできるだけ共通な形にしておきたいということを言っている場合もありますね。そのためにパラメタを沢山設けておいて、それをユーザに勝手に選定させる方法もありますが、ユーザの希望を言えば、将来、ユーザが作ったルーチンを適当にはさめるようになってるのが望ましい。パラメタが多いのは困ります。パラメタを設定するのなら、設定の仕方を明示してほしい。

モニタリング (測定)

山地 メーカとしては、メジャメント・ファシリティを入れる必要があると思います。

米田 富士銀行さんでは、メジャメントを行って、何か変えるということがあるのですか。

石崎 メーカによって、OS というのは、全くの聖域になっていて、ユーザはアンタッチャブルのところもあるのです。OS の中がモジュール化されていて、ある程度ユーザが入り込めれば、汎用 OS の不用の部分をカットして、OS のパフォーマンス・オーバーヘッドを減らすことができるのではないかと思います。OS については、一切アンタッチャブルのメーカがあるので、そこのところをもう少し公開してもらえば、ありがたいと思います。

上条 リアルタイム専用の OS というのがあってもよいのですが、メーカ側は何種類も作るのはいやだから、大体一種類に統一するわけです。小型機の場合に

は、特殊用途向けの OS が出て来てよいと思います。

石田 モニタリングの機能、あれは重要だと思うのですが、次の世代のシステムには、ぜひ大幅に入れていただきたい。そのためにはハードウェアのサポートがないとだめですね。メーカーの人に言っているのですが、次の世代の大型システムにはカウンタを千個位つけておいて、必要なとき見られるようにしたい。

三浦 時計の刻みも、もう少し小さい方がよい。

石田 OS を作っている人が、時間に余裕のないせいか、自分の作った OS のパフォーマンスがどうなっているか無関心なことがある。何を測定すればいいのですかということを書いてくる人もいます。ユーザの方から注文をつけて、こういう項目をぜひとも測れるようにしてほしいということを、仕様の中に入れる必要があるのではないかと思います。

首藤 ソフトの規模が大きくなって来ると、分業体制になってくるため、全体を眺めていて、もっと改善をしるとか、性能がどうだとかいえる人が管理者になってしまって、ユーザとの接触がなくなるようになるのも、一つの原因かも知れませんね。

石田 そういう人も、こちらがコンタクトしていると、いつのまにか次のプロジェクトに移ってしまう。

米田 パフォーマンス・モニタリングをやるのに、どの位のオーバーヘッドが許されるのでしょうか。

石田 モニタリングというのは、必ずしも毎日やる必要はない。必要なときに測ればよい。そのときには 5 パーセント位はいいのではないかと思います。不要なときには、取外せることが大事ですね。

テストデータ

米田 オンラインの場合のテストデータの作り方はどのようにしておられますか。

三浦 僕は細かいところは知らないのですが、テストデータというよりは、実際に小規模に端末をくっつけて、実際に動かしちゃおうというように、今のところは開発しているようです。

現在開発中のものは端末が百数十台ですが、最初は 10 台位でテストするわけです。それによってトラフィックが集中したとき、うまく処理できるかどうかというテストは、別の段階の問題です。

石崎 私の経験から言えば、すべてのサブルーチン・リンケージを全部通るようなテストデータを作れば、バグは大幅に減ると思います。我々の場合、トランザクションの量が多いからプログラムが複雑になって

いるのではなくて、対象業務が非常に多方面にわたっているからプログラムが複雑になってくるのです。だから 40 万ステップのすべてのステップを全部通るようなテストデータが作れたら、こんなありがたいことはないのですが実際にはなかなか作れないんですね。

米田 ダイラストラなどは、テストデータをいくら作っても、プログラムの正しさは証明できない云々と言っていますが、大きなシステムの場合、彼の言うような方法は実際には適用不可能なんでしょうね。

上条 私もそう思います。設計者が思いもかけなかったことが起きたときにトラブルが発生する。一番困るのはファイルを通してのバグですね。思い違いということがあるわけですから、何人もが関係していますとファイルが壊れても、誰が壊したか分らない。

三浦 現在開発中のものは、大体半年間、小規模の端末をくっつけてテストする。その間、本番と同じデータを代入して行こうと考えていますが、現場の速からは、手間が増えると文句が出ることもあります。

要員のモラル、教育の問題

米田 話は変わりますが、要員のモラルの向上は大問題だと思うのですが、千人もいるとシステム全体のごく一部しか見ていない人も多勢いるわけですね。そういう人のモラル向上についてはどうですか。

石崎 私の方は、コンピュータ部門が二つに分けられると思います。一つはオペレータの方ですが、これは単調な仕事ですから、組合との関係もあって 2 年ということになっています。最悪の場合でも 2 年以上はオペレーションをやらないという保証がありますから救いがありますね。問題は SE とかプログラマの方で、これは一部分だけという人がだんだん出て来ますので、コンピュータの部門の中で、できるだけローテーションを行って、モラルアップを図ろうとしています。先憂後楽ということをよく言うのですが、所詮はやはり支店の第一線の人達が楽をするために俺達が先に苦しんでいるのだ、という意識が支えになっていると思います。ああいうコーディングをやって自分自身が納得して行くためには、やはり数百人の SE なりプログラマが先に苦しむことによって、一万八千人余の人達がオペレーショナルな仕事から解放されているのだという自覚がないと、なかなかしんどいのではないかと思います。

米田 そういうことを、この頃の若い人に徹底するのは難しいではありませんか。

石崎 そうですね。

米田 吉村さんはワインバーグの本を紹介されたりして、要員問題に関心をお持ちのようですが。

吉村 教育の方から先に言いますと、私の経験ではソフトウェア・ハウスでは企業内の教育というのは、あまり期待できない。しかしソフトウェア・ハウスで働く人達は会社の本業と自分の仕事が大体において一致しているわけですので、ユーザの場合よりは基本的にはモラルのよくなる条件がある程度あるはずなんです。しかし最終的には本人の興味と適性ということに絞られるのではないかと思います。

米田 管理対自発性という点で、吉村さんは自発性の方をとる立場にあるように感じられるのですが。

吉村 管理が不要だと言っているわけではありません。担当者の能力を十分引き出して、いかにそのプロジェクトを進めて行くか、ということを管理と称するのであれば賛成です。

一方、管理のための管理というのがあるわけです。これはプログラミング以外の技術職でも同じだと思いますが、生産性を下げる以外の働きは何もしていないと思うのです。だから、最初に私が言った意味での管理であれば、それは必要だと思います。

首藤 要員のローテーションの件ですが、銀行さんではユーザ部門としての計算機要員、プログラム要員ですから、ローテーションというのは業務の種類の変更であるということになると思うんですが、私共のようにメーカで、しかも比較的小規模でいろいろなことをやっている、リアルタイム・プロセッサ、OS、データ通信と各々一つの専門職をなしているわけです。従ってローテーションをやっても、即戦力になるとは限らない。全体が大きくて各々の部門がかなり大きい場合はよいのですが、そうでない場合は機動性の制約のような感じになることがあります。

吉村 教育の件ですが、会社の費用で大学などへ通学させている所があるわけですけど、これは私の目から見ると非常に成功率が高い。基本的な知識と素養を身につけていれば、例えば、本人の仕事が言語からOSに移るような場合にも比較的スムーズに行く。

よく OJT という言葉が使われますが、ほとんどの場合、単なる徒弟奉公的な使われ方をしているのを OJT と称している。あれは特定の部門の約束事などには強くなるとは思いますが、あまり成功していない。OJT という言葉は安易に使われすぎていると思う。

三浦 OJT については全く同感です。OJT は、教

育カリキュラムを作ったり、実行したりするスタッフがないため、それをサポートしたときに使われる言葉ではないかという気がします。ところが一般のユーザの場合には非常に忙しいので、なかなか教育ということができないわけです。その僅かな解決策ですが、僕の所では、一部の人間を社外である程度活躍させる。外で仕事をやらせるというのではなく、若干遊ばせるといふか、社外の研究会に参加させたり、場合によっては講習会の先生をやらせる。そうすると、その人間は夜でも休日でも勉強しますから、適性のある人間ですと非常にレベルアップに役立つわけです。

米田 教育に関連して、大学における計算機教育について、大学側に要求することはありませんか。

三浦 ユーザの場合は全員が専門家であることは無理ですし、その必要性もないのです。むしろあまりスペシャライズしないことによるローテーションのしやすさがあります。東大なんかの場合、どの学部においても全部コンピュータを勉強しろという形になっているようですが、これはこれでよいと思います。一方、中核になってもらう少数については、実用的で、かつ、ハイレベルの教育をするというように、はっきりと目標を分けるのがよいと思います。東大では、そういう傾向を目指しているようですが、結構だと思っています。

処遇の問題、適性

石田 今のスペシャリストの件ですが、日本と比べるとアメリカの方がかなりスペシャリスト制度をとっているといえる。日本では、少しベテランになるとすぐ管理職になって、自分ではプログラムから離れてしまう。アメリカでは 50 才過ぎになってもプログラムをやっていて、少数精鋭で非常によいものを作る。日本では若い人を多勢集めて、OJT を兼ねながらやる傾向が強いようですね。

上条 少数精鋭ということに大分関係があると思いますが、どうもこの世界というのは知識の世界ではなくて適性の世界ではないかという気がするのです。50 歳、60 歳になっても、なおソフトウェアが作れるというタイプの人は限られている。そういう人を発見するというのが非常に重要な問題だと思います。どうやって発見できるかと言われると、私自身分りませんが。

吉村 スペシャリストをそのまま伸ばそうとすると、日本の場合、本人自身、あるいは彼らを何と育てて行こうとする使用者側いずれに対しても、社会的圧力がすごく大きいのです。税制の仕組や社会制度が

そういうことをするのは不便にできていると思います。

石田 チーフ・プログラマの話を会社でもち出すと、趣旨は結構だが、チーフ・プログラマがいないのではないかというオチになる。(笑い)

三浦 プログラムを書いているだけでも 40 歳になれば、課長級以上の給料をもらえるという制度ができれば、かなり解決するのではないかと思います。課長になると技術力は墮落するのです。逆に墮落しないと課長になれないという面もありますから。(笑い)

上条 今の体制では、チーフ・プログラマという名前をつけたところで、実際のプログラミングからは遊離してしまうでしょうね。

三浦 アンチピーターの法則というのが DATA-MATION に載っていましたね。プログラムができないから SE になる。SE もろくにできないと管理職になる。

首藤 私共の所で、小さい計算機でいろいろなハードウェアをくっつけたシステムを作って行く仕事があるのですが、その場合にソフトウェアだけしか知らない人は、極めて不適であるということがあります。システム全体の最終調整の段階になりますと、ハードウェアの虫なのか、ソフトウェアの虫なのか分らない。そのために、ハードウェアの経験を何年か持ってから、ソフトウェアをやるということも必要なのではないかということを感じています。メーカで特殊なシステムを開発して動かすという場合に、特にいやらしいのは、ハードウェアで、デジタルの技術というより、アナログの領域に半分入ったような、ノイズの問題とかそういうものが、どのようにプログラムに影響して現象として現われるか、そのあたりについて、かなり幅を持った知識、経験の必要性を最近感じています。

上条 最近はよくなっているでしょうが、以前はモデムがノイズを出して、それがデータとして混入してくるというケースがありました。これのデバッグは大変でしたね。誰が、どこから出しているのか分らない。

石崎 私なんかは両極化して行くような感じを持っているのです。私もオンラインの端末装置をメーカと何年間か共同開発しましたが、最近マネジメント・サイエンスをやっている人達を見ていると、ハードウェア離れ、コンピュータ離れの現象が強く出ています。そういう人達は、時系列分析とか計量経済学なんかには非常に強いのですが、コンピュータのハード、ソフトの知識はあまり必要とはしていません。ところが今

やコンピュータはそれでも使えるようになって来ているといえると思います。

プログラム管理、システム変更、機種変更

米田 また話を少し変えて、プログラムはどんな形態で管理しておられますか。物理的な形、ロジカルな形といろいろあると思いますが。

三浦 プログラムは一応、ソース・プログラムでもディスクに入れてあります。原則としてカードは開発段階が終わったら捨てるという形にしています。もちろん、ドキュメントも何部か作って分散して保管するという形にしています。そうするとプログラムを変更してもドキュメントは変更されないことがある。それでは非常に困るというので、オートフローなどを使って逆にプログラムからドキュメントを作る必要性を感じています。今のところは、プログラマ、SE はドキュメントも直してくれるであろうと、若干あやしいな信用をしてやっています。

米田 管理者の承認なしで、SE とかプログラマが勝手にやればやれないことはない環境なのですか。

三浦 手続き上はちゃんと変更依頼書とか承認書とかいう伝票はあるのですが、100% 実行されているとはいえない。見たことはない。(笑い)

米田 オンラインの場合はもっと厳しいのでは。

石崎 変更についても、かなりきっちりやっています。プログラマには原則として自分でデバッグさせていません。よほど緊急事態でない限り、オペレータに委託してデバッグをやらせています。そこでプログラム・デバッグ依頼書を書くわけです。オペレータは中身は知らないで流していますが、当然バグが残っているわけですからループに入ったりする。その時点でメモリ・ダンプをとり翌日つき返す。ドキュメンテーションのプロセスはかなり固く運用している方ではないでしょうか。

米田 TSS をプログラム・メインテナンスのための道具としてお使いになっておられますか。

石崎 一部 RJE という形では使いましたが、会話型という形では使っておりません。うっかり本番のデータをいじると危険なので。

三浦 先ほど、あまり徹底的な管理はしていないと言いましたが、実務上不可欠と思われる点については一生懸命やらせているわけです。

二年ばかり前に、うちの部の約半分位に相当する業務を一度に、全然別の機種に変換して、プログラムを

全部書きかえたのですが、そのときは、そのドキュメントがあったために非常にうまく行きました。ドキュメンテーションをやっていたよかったと思ったわけです。そのときの変換作業は予定した日時に一日も遅れずに完了することができました。

米田 その間システム変更はないのですか。

三浦 もちろんありますが、一年半前に決めた期日通りに一日も遅わずにできました。それはドキュメントがあったことと、その変更手続きを守ったというのが大きな理由だと思います。

米田 機種変更の場合、いわゆるコンバージョン・エイドというソフトウェアは役に立ちますか。

三浦 ええ、役に立ちました。徹底的に使いました。

米田 オンラインの場合の機種変換については。

石崎 エミュレータのようなものもプロポーザルにあったのですが、全然使いませんでした。今までに8年位オンラインを運用して、多少知恵もついていますから、あそこも直したい、ここも直したいという所があるわけです。そんなわけで、せっかく巨額の投資をして新しいフィロソフィーに基づいたコンピュータを入れるのに、エミュレータなどを使うのは勿体ないという感じがしたわけです。一時的には人は沢山いますが、プロジェクトが終われば他の仕事に回せるわけですから。

米田 ピーク時には他の業務の人もオンラインの方に集中して、中で工面するわけですか。

石崎 そうですね。ピーク時には傘下の計算センタの方からも人をやりくりしてもらわなければ。また、一部はメーカの方にお金で解決できるものは委託して開発してもらったこともあります。

米田 先ほど、支店によって新旧いずれかのシステムが使われているというお話をされましたが、その切替えはどのようにして行くのですか。

石崎 新旧両システムのコンピュータは設置場所も離れていますから、モデムを介してメッセージのやりとりをしています。

米田 ある支店のお客さんのデータを他の支店からアクセスするようなことは、あまりないのですか。

石崎 ありますよ。旧システムの支店のお客さんが新システムの支店の窓口で預金を引き出すこともあるわけです。この逆も可能です。モデムを介したシステムにしたことが、大きなシステムでありながら意外にプログラムを簡単にしている原因かも知れません。

首藤 ファイルが大きな慣性を持っていると思いま

すが、新旧それぞれダブルせて持っているのですか。

石崎 ダブルせていません。一つのデータは常時いずれか一方にあるだけです。土曜日まで旧で、月曜日から新のファイルに移し変えられています。

米田 日曜日に移し変えるのですか。

石崎 そうです。

上条 残高照合だけ共通に出来るシステムもありますね。

石崎 日本のお客さんはそれでは満足しないですね。やはり、リアルタイムに通帳をアップデートしてくれる銀行を選ぶということになるわけですね。

米田 メンテナンス要員は開発要員と別ですか。

石崎 短期的には同じですが、長期的には変わっています。ユーリ・パーマンなどが新規業務を開発する人間の割合は、だんだん減って行くと言っていますが、あれは実感ですね。メンテナンスの人員がエスカレートして行くのは一つの悩みです。それと、アンコントロールなファクタが多い。例えば、くじつき定期預金が販売されると、好むと好まざるとに拘らずシステム変更せざるを得ない。こういうのが意外に多いのです。

米田 日常業務として動いているプログラムを変更する場合、変更が正しく行われたということ、どのようにしてテストしておられますか。

三浦 いろいろ、がたがた、テストデータを入れてやっているようですが、結局、本番を二回やるという形になっているようです。実情は。

米田 変更のヒストリーは全部とってありますか。

三浦 それはとってあります。最終的にどう整理するかということはややこしいことになりますが。

石田 機種変更や OS のバージョンアップに関連して、今後データ通信の方式がどんどん変わってくると、古くからある端末装置をどこまでサポートするか、我々の所でも頭を悩ませています。

今後の計算機技術への要望

米田 最後に、ソフトウェア・エンジニアリングの立場から、今後の計算機技術への要望を簡単に。

石田 一つは先ほどのモニタリングは、次の世代のシステムからぜひ入れてほしいということです。

もう一つは、ソフトウェアに虫がいるということは将来とも覚悟する必要があると思います。ストラクチャード・プログラミングとか言われていますが、実際の立場として虫はつきものだという前提で考えざるを

得ない。例えば、OSの一部に虫があっても、全体としてはダウンしないような設計を考えてみたい。

三番目は、将来のシステムはハード的に、ある種のマルチプロセッサ形式になるのではないかと思うのですが、プロセッサが1台ダウンしたとき、切離していかうまくやるか。また、メモリについてもページングをやっているとき、OSが実メモリ全体に広がってしまっているとき、切離しをどうやるかなど、複雑なシステムの信頼性を高めることを真剣に考える必要がある。

米田 プログラミング言語についてはどうですか。

石田 我々のセンタでは、95パーセントがFORTRANです。FORTRANがなくなるとは、考えられません。

米田 MTCについてはいかがですか。

石田 あれは期待外れだといっているのですが、実用レベルで役に立つようなものは出て来そうにない。

三浦 メーカーの利害が相反することになるかも知れませんが、ハード、ソフト共に標準化ということを考えてほしいですね。機能が非常に豊富になるということよりも、標準化することの魅力の方が大きいと思います。

もう一つは、事務計算などの場合でも、無人運転が簡単に安心してできるようにしてほしいということです。プロセス制御の場合は実現しているわけでしょう。

三番目は、直接コンピュータのハード、ソフトの開発技術とは無関係なのですが、要員の問題に関係して、プロフェッショナリズムというものをある程度推進して行く必要がある。最近のCACMによくプロフェッショナリズムについて載りますが、これを推進するためには、ライセンス化なども必要だと思います。

上条 私の立場から言いますと、PL/1のような言語、ある程度システム・プログラムの記述のできる言語を、ぜひどの機械にも使えるようにしてほしい。それによって、特にユーザ・プログラム・レベルでの共通化を図ってほしいと思います。あるいはハードまで含めて考えるならば、エミュレータ的なもの、例えばマイクロプログラムのユーザに対する開放ということで実現してもらってもよいと思います。

次は、何とかしてシステム・プログラムの信頼性を上げてほしいということです。

三番目は、日本では、システムのライフが一般的に短いので、これを何とかして長くしてほしいということです。

首藤 一つはFORTRANのような手続き向き言語で書かれたプログラムのデバッグをソースレベルでやりたいということです。特にオンライン・プログラムの場合について、考えてみたいと思います。

もう一つは、会話モードということをして、デバッグにどのようにとり上げて行くかということです。

米田 石崎さんの方では、ビッグ・システムをおやりになって、いろいろ注文がおりだと思えますが。

石崎 三つばかり希望があります。

一つは、一部商用TSSで提供されているような、non-EDPの人でも会話型式で使えるアプリケーション・パッケージを豊富に開発してほしいと思います。それによって、今後コンピュータが一部テクノクラートの手から解放されて、もっとコンピュータの裾野が広がるはずですね。

二番目はOSをもっとモジュール化して、それをもっと公開して、ユーザがある程度、必要なモジュールと、そうでないモジュールとを使い分けることが出来るようになると非常に有難い。今はアンタッチャブルな聖域になり過ぎているという感じです。

三番目は、テストデータ・ジェネレータのようなものが作れないかということで、私の経験でも、隅から隅までテストするためのテストデータ作りに一番苦労したので、このあたりの研究をぜひお願いしたい。

三浦 プログラムのステートメントから逆にデータを作るというのができてよさそうに思いますね。

石田 石崎さんのおっしゃったことに多少異論があるのです。non-EDPのユーザに使わせることをやみくもに推進してもよいものかどうか、non-EDPユーザがコンピュータを使って出した結果が信用できるかどうかは、どういうふうにしてテストするのですか。

石崎 例えば、調査部の人達などが商用TSSを使って、結構いい結果を出しているわけですね。

石田 そう心配はないということですね。

石崎 私はそこところは楽観的なのです。銀行のように文科系の人が多い所でも現実的に、TSSを使いこなす人が出て来ている。そういうことが、コンピュータ・アレルギーをなくして行くことだと思います。ユーザの側で発生するデिलリーの問題まで、いちいち我々のところでやるのでは、我々も大変だし、それではコンピュータ利用者の数もなかなか増えないのではないかと思います。

首藤 石田先生のおっしゃるような心配はあまりないと思います。EDPセクション以外の人達がどんど

ん入って来ましても、例えば企業体の中だと、計算機にやらせる問題の選択、費用のかけ方などについて何段階ものチェック機構が動くはずですので、ベクトルが発散する心配はあまりないといえます。

石田 最近、内外の学会などで、あやしげなシミュレーションをやって、堂々と結果を発表するような人がいるので、それで心配しているのですが。

吉村 まず一つは、会話型環境でプログラムの開発ができるようにしてほしいことです。これはある意味では、プロペラ機からジェット機に乗り換えた位のインパクトがあると思っています。現在では、ごく一部の恵まれた人達が、そういうものの恩恵に浴していると思うのですが、一般のユーザ、あるいはソフトウェア・ハウスを問わず、経済的にも技術的にも、もっと使い易い会話型の道具が使えるようにしてほしい。

二番目には、特に和文のドキュメントの作成と保守のためのシステムが、一日も早く利用可能になることを切望します。ドキュメンテーションの点で、欧米と我々との間には大きなギャップがある。その決定的な理由は、文字が計算機の中で自由に扱えるか否かにあるわけで、今の所、実験的なシステムや、一部印刷業界におけるシステムはあるものの、プログラムの道具にはなっていないわけで、ぜひお願いしたい。

三番目に、技術に対する要望ではないのですが、非常に大きな計算機を持っている少数の大学は別として、コンピュータ・サイエンスの教育に熱心な大学で一番困っているのは、計算機が十分に使えないことです。ですから特に大きい機械をお持ちになっているユーザとかメーカは、教育機関に対して少しでも計算機時間を donate して頂きたいと思うのです。アメリカの計算機教育をみると分るように、学生個人の能力には、そんなに大きな差がないにも拘らず、どうしてアメリカの方が計算機教育が進んでいるかというと、一つの理由は計算機時間がふんだんに使えるからであって、これはぜひお願いしたいと思います。

石崎 今のコンピュータ時間の donate ですが、私の所ではやっています。ある大学の先生と一緒にモデルを作っていますが、計算機時間は我々の所が donate しています。これは我々にとってもメリットがあるわけです。

山地 メーカの立場なので、若干言にくいのですが、二つありまして、一つはユーザさんの要求というのがあまりにもバラエティーに富んでいる。これを何とか少しでもよいから優先順位というか、ランク付け

に関するコンセンサスをとって頂けないかということです。その優先づけに際して、この部分は我慢してもよいというようなことを言って頂けると有難い。(笑い)例えば先ほど石田先生が、モニタリングを行うときには、5パーセントまでオーバーヘッドは我慢できるとおっしゃったわけですが、そのように具体的に言って頂けると非常にありがたい。

二番目には、特定一社のメーカだけでは決めかねるという問題がだんだん出て来ています。そういう場合に、情報処理学会とか JIS の規格とかで、一つの方向を打ち出して頂けると有難い。例えば FORTRAN 文法も JIS 化できたのだから、JCL についても何らかの標準ができるといいのではないかと思います。

米田 仮想記憶というのは、ソフトウェアにどの位のインパクトを与えているのでしょうか。

三浦 プログラミングの生産性が上がったかどうかというのは分らないのですが、今まであったプログラムを全然書き換えなくて、メモリを半分にしても大丈夫だという例はあります。

米田 ソフトウェア・エンジニアリングの立場ではいかがですか。

三浦 逆に言うと、少々下手なプログラムでも、効率低下を認めれば何とかなるということは言えそうです。

石崎 我々の方は、今までものすごいオーバレイをやって来たわけです。どこをオーバレイさせればよいか四苦八苦していたのですが、そういうことを考えずに済むようになっただけでも非常に進歩だと思いますね。

石田 仮想記憶は、私の所ではまああと評価しています。物理的なものと論理的なものとを分けるのは本質的な考え方ですから、これは将来とも定着すると思います。ただユーザに悪いクセをつけたという反面はありますね。

吉村 仮想記憶は逆にソフトウェア技術に対する挑戦でもあると思います。一つはコンパイラなどは、今まで仮想記憶を前提としないような基本的な設計技術が確立していたわけですが、今後は仮想記憶を前提としたものが要求される。ソートなどについても同様です。その意味で、ソフトウェア技術者の方が頑張らなければいけない問題が出て来たと思います。

米田 いろいろお話し頂きましたが、一応この辺で終りたいと思います。本日はありがとうございました。(終)