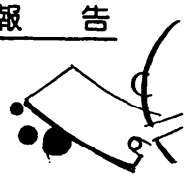
報　告



パネル討論会

要求技術の目指すべき方向†

パネリスト

野木　兼六[1]，森澤　好臣[2]，阿草　清滋[3]

司会　松本　吉弘[4]

座長　時間になりましたのでパネル討論に入りたいと思います．パネル討論の司会は東芝の松本さんにすべてお任せします．初めの私の役目は司会の松本さんを簡単にご紹介をすることです．

松本さんは昭和29年に東京大学の電気工学科を卒業後ただちに東芝に入社され，計算機制御システムのソフトウェア開発に以来従事されています．49年に東大から工学博士を受け，現在は，東芝の理事，重電技術研究所技監でございます．たぶん皆さん方は「ソフトウェア工学演習」という著書などで，ご存知と思います．それではよろしくお願いいたします．

松本(司会)　松本でございます．ソフトウェア工学研究会の花田主査 (NTT) のご要請に従ってこのパネル討論会を計画いたしました．

初めにパネラの方々をご紹介します．野木兼六さんは，昭和44年に日立製作所に入社され，その後プログラミング言語，ソフトウェア工学の研究に従事され，1年前から同社基礎研究所で仕様技術，プログラム自動生成の研究を行っていらっしゃいます．

森澤好臣さんは昭和42年日本ユニバックに入社されまして，ユニバック1100シリーズの言語プロセッサの開発，提供，保守を担当され，4年ほど前から論理型言語に興味をお持ちでございます．現在，同社の知識システム開発部でLISPマシンの提供業務をやっていらっしゃいます．それからソフトウェア工学研究会の連絡員及び今回のシンポジウムの実行委員をされています．

阿草先生は，すでに皆様ご承知のとおり，45年に京大を卒業，47年修士課程を終了され，同大学情報工学科にお入りになりましたのは49年です．現在，情報工学教室の助教授，ご専門はソフトウェア工学でございます．

このパネルは2部に分けて構成します．第1部では問題提起，第2部では将来へ向けての提言を討論いただきたいと存じます．では第1部を始めたいと存じます．

論点を4つ示します．第一は要求形成 vs. 要求定義であります．要求という概念は複雑なモデルであるために，それを形成するに当たって見方 (view) がいろいろあります．たとえば皆さまがよくご存知の酒ビン問題[1]，あれを説明するのには絵をかくとわかりやすい，たとえば図-1のようなものです．こういうものを難しくいえば形態的ビュー (morphological view または spatial view) とよびます．これに対して，よく知られたものに functional view というのがあります．これは SADT (SofTech 社登録商標) に代表され，要求を機能的見地から考えて，機能と機能相互間の関係を書きます．さらに，contexual view もあります．たとえば，E-R モデルのようなもので，ここでいう context (前後のつながり) とは，要求に含まれる単位概念をエンティティ (entity) とし，エンティティとほかのエンティティとの間の相互関係のことを指しています．このほかにも種々の見方，すなわち view があります．時間関係を入れた dynamic view も当然考えられます．種々のビューを使って要求を多面的に形成します．

次に定義，記述について考えますと現実には IEEE Std. 830-1984 をはじめ数多くの標準や規格があり，これに準拠せねばなりません．日本ではユーザの方々とメーカどもが非常に密着にコンタクトをしておりまして，同一民族であることも助けになり，意志がよく通ずるので，あまり問題にならないのですが，私自身外国のソフトウェアの仕事に巻き込まれた体験からしますと，要求仕様を規格に従って書かないと仕事が進まないという現実に直面するわけです．要求仕様書が

† 日時　昭和61年4月17日 (木)
場所　機械振興会館大ホール
1) 日立，2) 日本ユニバック，3) 京大，4) 東芝

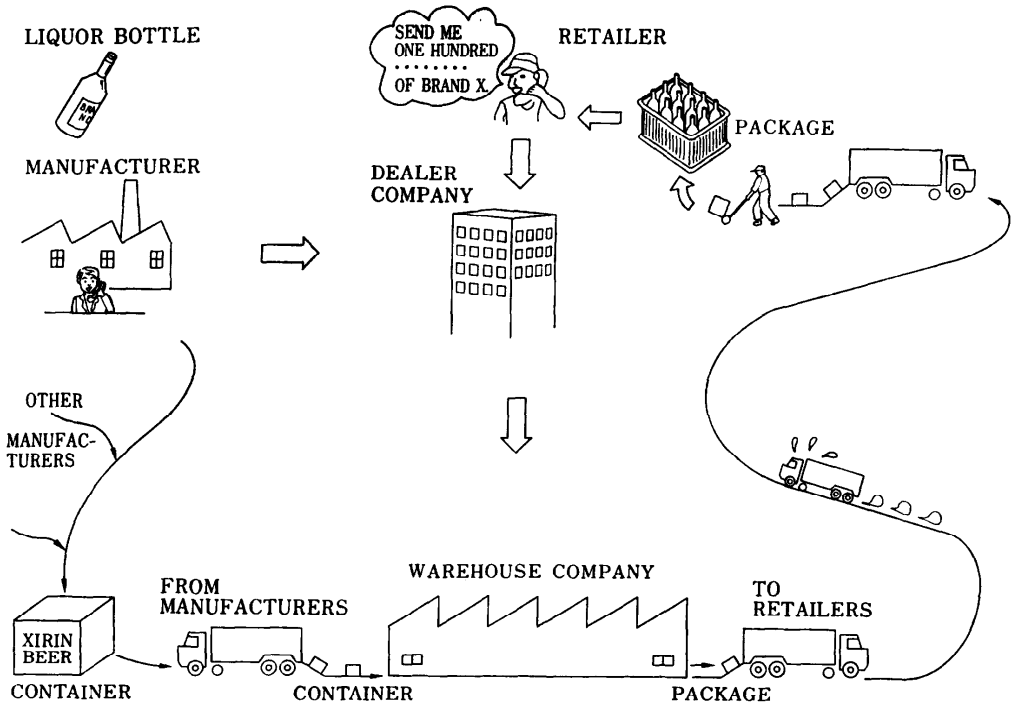


図-1 形態的ビュー (morphological view)

一つの商品であるという社会になっているわけです．これからわれわれが国際的になるためには，定義や記述に関する標準をもう一度見直すようなタイミングが日本にも必要となるのではないかと考えます．これに関連して，構成管理プラン (configuration management plan) というものがあります．これはライフサイクルでのあと戻りを含めて，構成及びその要素をキチンと管理する技術です．ライフサイクルでいう waterfall とは，そのまま日本語に訳すと滝ですが，そうじゃなくて水を上下に導く樋を張りめぐらして水を流すという概念を含みます．あまり上下に行ったり来たりしますと，ポテンシャルがなくなって水が止まってしまいます．構成管理については，IEEE 828-1983 の規格があります．

要求は，工程の上を上下に行きかう水樋のような通路に沿って往復しながら成長するものであるというのが第一の論点でございます．

第二の論点は要求仕様書 vs. 基本設計仕様書ということです．日本の場合は，あまり要求仕様をきちんと書かなくても，物事が進むということで，設計者はできるだけ工程を短絡したい，生産性を上げるためには少しでも工程を減らしたいという気持が強いわけです．したがって客先の要求をそのまま書くのでなく，それを実現するための設計モデルを頭の中で作ってから，その記述をまず行う．その中に客先の要求を合わせて記述し，要求仕様書を代行するような仕様書を作成する場合が現実には多いと考えます．このような記述を基本設計仕様書，またはそれに相当する呼称でよびます．

第三の論点は形式的仕様記述 vs. 非(半)形式的(以下インフォーマルと称す) 仕様記述です．

実務では，形式的仕様記述を採用することは，ほぼ難しいといえます．しかし全くのインフォーマルということもあり得ない，様式 (form) に代表されるなんらかのルーズな形式性はどうしても必要になってくる．上流工程での形式性をどのように考えるかというのがこの論点でございます．オペレーショナル・アプローチという手法があります．要求を形式化しにくいときには，オペレーショナル・モデルというのを作って，これを動かしながら客先のニーズとの整合性を高めていく．最終的なオペレーショナルな仕様によって要求仕様を代行させようというのがこの手法です．この手

法にも問題があります．ここで作られるオペレーショナル・モデルが基づいている構成上のセマンティックスが，実計算機上での実現におけるセマンティックスとかけ離れておりますとコードへの変換効率が悪くなります．したがってできるだけこのセマンティックスの差異を小さくしたいという基本的な要求があるわけです．どういうところを注意しないといけないかといいますと，タスク並列性，記憶データアクセス，資源アクセス，外界との入出力，データ通信などにおけるセマンティックスです．あまりにも抽象的なオペレーショナル・モデルでは，これらのセマンティックスはほとんど反映されませんから，コードへの変換工数が従来と同じようにかかって，結局ウォータフォールモデルとあまり違いがなくなってくることになります．

第四の論点は，実世界モデル vs. 実現モデルです．プロトタイピングという概念は，従来のシミュレーションという概念と近い．この二つの概念は，どこが違うかというと，プロトタイピングの多くは，記号で表される概念の検証を主として行います．ヒューマン・ワールドとのインタフェースの検証は非常に楽に行えます．しかし，機械などフィジカル・ワールドとのインタフェースを検証するためには定量的なデータ表現や数値演算が必要になります．そうなると従来のシミュレーションに頼らなくてはいけないということが起きます．

では，まず野木さんからお願いいたします．

野木　野木でございます．今，松本さんのほうから論点が出されたわけですけれども，その話をする前にまず私の考え方の基本になっております操作的アプローチについて簡単に説明をしておきたいと思います．

操作的アプローチ（オペレーショナル・アプローチ）というのは，70年代のソフトウェア工学で研究された要求定義技術の反省という形で，特にシステムの機能的動作を簡潔に記述することができないという問題を克服しようとして提案された新しいアプローチですが，80年代の要求定義技術の主流になっているといっていいかと思います．

その特徴ですけれども，一つは要求定義と設計の違いということです．従来のように何をというのは外部仕様，いかにというのは内部仕様，そういうふうに考えるのではなくて，何をというのは問題領域におけるシステム仕様，いかにというのは計算機領域におけるシステム仕様，こういうふうにとらえ直したというのが一つの大きな特徴じゃないかと思います．

二番目の特徴は，トップ・ダウン機能分割法というのがよいと従来されていたわけですが，それは要求定義ではあまりうまくいかないという批判がありまして，むしろ要求定義の場合はボトム・アップに実世界というものをモデル化していく方法，ボトム・アップあるいはアウトサイド・イン・モデリング法と言っていますが，そういう実世界のモデル化というのを非常に重視しているということです．

三番目の特徴は，よく話に出ます実行可能性ということで，実行可能な要求仕様によってプロトタイピングをやろうということです．こういう操作的アプローチという考え方をベースにして，先ほどの松本さんから出された論点について考えてみたいと思います．

まず要求分析と要求定義ということですが，この問題を考える観点として，私は要求定義と設計の類似性を考えたらいいんじゃないかと思います．というのは，もちろん要求定義と設計というのは質的に違うものですが，広い意味の設計というのが，設計とプログラミングに分かれてるということと対応して，広い意味の要求定義というのが要求分析と要求定義に分かれてる．ここに何か類似性があるんじゃないか，こういう観点でこの問題を考えると次のようなことがいえると思います．

一つはプログラミングを設計から分離するというのが不自然であるように，要求分析を要求定義から分離するのは不自然である．これはどういうことかといいますと，広い意味の設計は計算機の世界で考えるものでありまして，要するに設計とプログラミングとは同じ質の作業なわけですね．それと同じように，要求分析と要求定義も質的に同じものである．だからどうしても前のほうの作業は，後のほうの作業の準備作業のような感じになってしまって，要するに閉じないわけですね．それで設計者とプログラマとの引き継ぎに問題が出てくるとか，そういうことになるわけですが，それと同じことが要求分析と要求定義でもいえるんじゃないか．

ところが実際にこの二つの段階は分離されているわけです．なぜ分離されているかという理由を考えてみますと，一つはやはり目標とするところが違う．前のほうの工程ではそのプロセスが重要であるのに対して，後のほうは結果の品質が優先されるということ

で，目標が違うために分けざるを得ない，そういうことがあると思います．それからもう一つは大規模システムではモジュール化というのが必須でありまして，前のほうでざっとした概略イメージを作っておいて，後のほうできちんと書く，そういうふうにしないとできない．人間の能力の限界に関連しているわけで，どうしても一気に書き下せないので，こういうふうに分かれてるということもあるわけです．したがいまして一つにすべきだけれど，分けざるを得ない，こういう矛盾が出てるんじゃないかと思います．

ではどうするかということになるわけですが，結論は平凡なところに落ち着くんですけれども，分けざるを得ないにしても，この同質性を保証するために，一貫した言語，方法論，ツール，こういうものを開発することがこれから重要になってくると思います．

時間がないので全部の論点について述べられませんが，最後の実世界モデルと実現モデルについて，ちょっと話したいと思います．この場合の観点としては，実世界モデルと実現モデルの違いを考えてみました．それを整理しますと，要するに実世界モデルというのは，環境と対象システムを両方含んだ形でモデル化するものであり，それに対して実現モデルは，一応対象システムだけについて，それを計算機上でどういうふうに実現するかということを考えるモデルである，といえるわけですが，両者の対象システムのモデルのどこが違うかといいますと書かれている言語が違う．実世界のほうは問題領域の言葉で書いてあるし，実現のほうは計算機領域の言葉で書いてある．そういう違いがあると考えられるわけです．そうすると操作的アプローチなどでも強調してることですが，実世界のモデルというのは，環境の構造を対象システムの構造に反映させることができるわけで，それによって利用者にとってわかりやすく，保守もしやすい構造が作れるということになるし，一方実現モデルのほうはプログラムへの変換が比較的容易であるということになって，それぞれ長所があるわけです．

しかし二つ作るのは大変ですから，どっちかにしたいわけですが，実現モデルで対象システムというものをモデル化しても，そこには環境の分析が入っていないわけですから，原理的に実世界モデルの代替手段にはならないわけです．したがってどっちかを作るとすれば，実世界モデルを作りたいわけですけれども，実世界モデルのほうはプログラムへの変換がむずかしいという問題がある，それを解決しなければ実世界モデルだけというわけにはなかなかいかないという話かと思います．

それでどう考えたらいいかということになるわけですが，対象システムのモデルの違いは要するに言語だけなわけですから，その違いというのは，どうやって解消できるかというと，抽象度を上げることによって解消できるんじゃないかと思うわけです．要するに利用者向けの言語とか，開発者向けの言語といいますが，具体的に考えると，いろいろ違いが出てくるわけですけれども，抽象化していくとだんだん特有の性質がなくなってきて，一番抽象度の高いレベルでは違いがなくなってしまうんじゃないかということです．そういう抽象的な実世界モデルを考えれば，それは抽象的な実現モデルを含んでいて，しかも範囲的にも環境と対象システムの両方のモデル化になっているわけなので，結局そういう抽象的な実世界モデルを作るというのがコスト的にも一番いいということになるわけです．もちろん抽象的なモデルを作るのは非常にむずかしいという点もありますが，それはやはりノウハウの蓄積とか，そういうことで解決していくしかないんじゃないかというふうに考えます．

阿草　京都大学の阿草です．次の森澤さんが控えておられますので２～３分で遅れを取り戻したいと思います．

まず松本さんからのテーマについて，われわれの立場を少し考えてみました．ただ，われわれ大学の立場ですので，物を作ったことがない．たぶんつるし上げる材料として出されてるんだと思います．要求の仕様の分析というのは，結局相手の言葉を理解するとか，立場の違いを認識することである．要求の記述というのは，それを自分の言葉で書くことである．ということは，エンド・ユーザが書くのと，システム・エンジニアが書くのとは本質的に違う．ただし，相手の言葉の理解という意味で同じコンセプトがとられているはずだということを期待したい．そのときに理解が言葉で本当に左右されるのだろうか．われわれが持っている概念というのは，どうにかすればエンド・ユーザと，たとえばSEレベルの人たちとの間で，同じ理解がとれてるとすればうまくできるんじゃないか．その理解をするためにどうするかというと，そういうコンセプトが頭の中に入ってるだけではだめですから，どうにかして外に具現化したい．そのためにはプロトタイプが有効である．そうい

うロジックで要求仕様化，すなわち分析を正しく行って記述するためにはプロトタイプが重要であると，そういう立場だと思います．

次に要求仕様の役割としては，玉井さん(三菱総研)のお話にもあったように合宿があるとか，インタビュがあるとか，会議があるというのは，すべてコンセンサスを得るためである．さっきいったその理解を得るためである．理解した結果を契約書か何かの形で加えるわけですが，われわれが興味を持ってる要求をフォーマルに扱いたい．そうするとその要求が最小性を満たしているかというと，たとえば要求側でいう最小性と，もちろん受入れ側でいう最小性というのは本質的に違う可能性がある．すなわちその最小性や一貫性というのも立場によって違うわけで，要求仕様書を，そういうフォーマリズムの検証の対象にしようとするといろいろ問題があると思います．

今日インフォーマルなアプローチについて話されましたが，私はインフォーマルというのは全然存在し得ない．いわゆるインフォーマルという意味はフォーマリティが少し下がるという意味はあっても，フォーマライゼーションがゼロということは絶対あり得ないわけですね．われわれが使ってる自然言語も，ある種のロジックで組まれているわけで，それは解釈系がちゃんと用意されてるという意味で形式性をとらえるんなら，形式性は必ずある．そのときの形式性をもたらしてるのは，意味が定義されてるというか，その解釈がちゃんと用意されてるという意味でフォーマルである．それが検証を目的とするときには，数学的モデルですと扱いやすいが，理解を目的とする場合には具体的にわかりやすいという意味で要求仕様の場合には極端にいえば，最終システムを渡すのが一番いい．だからラピッドプロトタイピングよりも，ラピッドインプリメーションをやってしまうほうが早いということですね．

そこでのモデルは，実現システムのモデルとか，そういう立場でとらえられてますが，モデリングというときには，モデル化の過程を重要視するときと，でき上がったモデルを重要視するときと二つあると思います．今からは方法論のモデル化ということも，もう少し頑張らないといけない．そのあとのソフトウェアを作る全体のプログラム変換モデルをもう少し考えてもおもしろいんではないかと思っています．

いろんな人のいろんなモデルがありますが計算機が内部状態を変えながら実行するものだとして認めればFSM モデル，それから実際の社会環境をうまく表わそうとするとエンティティリレーションとか，あと分散システムとか，今後のアプローチとしてはコミュニケーションというか，CSP のような形のモデルというのが重要ではないかと思っています．

それから要求仕様化でのモデルを考えると，さっきと同じことがいえまして，対象システムをモデル化するというのは，対象システムの空間を定義することで，それが正しく行われたかどうか，品質検査などをやるためには，やはり数学的モデルがほしい．ただしそのコミュニケーションがうまくいったかどうかという意味での検証をするためには，最終システムというか，ある意味でイグゼキュータブルな意味定義がされていないといけない．そういうふうに考えています．以上です．

森澤 ユニバックの森澤です．こういう場に出るのは好きじゃないもので，逃げて回ったんですが，逃げられず，今回出ることになりました．紹介にありましたように，シンポジウムの実行委員３人の末席を汚しています．

諸先生方に整った話をしていただいたので，私のようなおちこぼれプログラマが何をほしがっているかの私見を述べてみたいと思います．話し方として利用者の立場と，物を作らなければいけないという立場があると思います．プログラマの端くれですから，物を作る立場から話をしたいと思います．

まず1970年代だと思いますが，孫引きになりますがエルショフの講演の中に気になる言葉があります．

プログラミングに必要な才能はどのような才能か．第一級の数学者の論理性が必要です．ちゃんとした工学の才能も必要です．それから物事をきちんと理詰めでやる，手続きに従ってやる才能も必要です．それから何か突拍子もない発想法，この突拍子もない発想法の一番良い例はたぶん，詐欺だと思います．詐欺師は人が思い付かない発想をしてくれます．そういう豊かな発想法が必要です．それからちゃんと真面目に仕事する．それから大規模なソフトになると１人では作れません．たくさんの人と共同で仕事をする必要があります．したがって同僚とうまく酒を飲みながら仕事ができる．こうやって振り返るたびに私はおちこぼれ，どれもあてはまらない．どうすればよいか，いつも悩んでます．

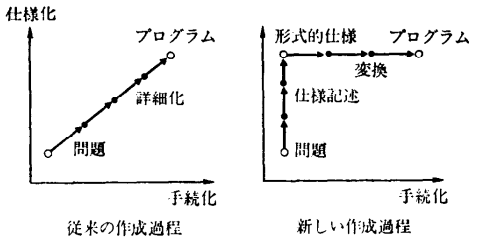


図-2　ソフトウェア作成過程
注）岡田ほか：プロトタイピング技法の形式化の試み，情報処理学会「プロトタイピングと要求定義シンポジウム」予稿集，pp. 29-37(昭61年4月)より引用

要求仕様とは何か，皆さんの話を聞いてても思うんですが，要求という言葉には，それぞれ皆さん方思い込みがあるようです．失礼と思いますが発表者も発表に合せた定義をなされてるような感じがします．たとえばある問題を解くソフトウェアを開発するとき，問題を定義する人の要求もありますし，ソフトウェアを利用する人の要求もあります．全部違います．このような要求を記述できるのは自然言語しかないだろう．でも自然言語ではやっぱり計算機に乗らないので，だめなプログラマにとって乗るような，もう少し楽な道具だてがほしいという希望を持ってます．たとえば事務処理のある分野にしぼりますと割と形式的なことでできるだろう，そこら辺の発展をまず願ってます．そうするとプログラマの命がもう少し延びるんじゃないかなと思ってます．

説明に図-2を使用させていただきます．従来のプログラム作成過程，たとえば問題が与えられると問題から要求定義，仕様記述，概要設計，詳細設計へと仕様化の度合及び手続き化の度合を上げています．今までプログラム作成過程で計算機支援をやってるのは，この従来のプログラム過程に対する計算機支援だと思います．しかし，われわれがほしいのはこれではないのではと思います．もう少しソフトウェア作成過程を楽にするための方法が別にあるんじゃないかと思います．

ほしいのは，ほしいというのはこれ私の希望するということですが，問題を手続き化度合を変えずに，仕様化の度合を上げる道具です．まず，インフォーマルというのは自然言語だとしますと，自然言語に少し制限を加えて，フォーマルな仕様にする．もう少し何かを追加すると runnable までいかないが walkable よちよち歩きできる仕様になるだろう．そうなるとその後に少し実行を早くするための手続きを加えれば，runnable 仕様ぐらいになるんじゃないか，そこまでいきますと機械的なプログラム変換で，実用的なプログラムが作れるのではと思っています．こういうプログラム作成過程をサポートしてくれる道具がほしいなと思ってます．

ここではそれに Intelligent Amplifier という言葉を付けました．これは私が作った言葉じゃなくて，NTT の奥乃さんの使われた言葉です，最近はやりの AI じゃなくて，IA のほうがほしいと思っています．IA によって runnable 仕様から実用的なプログラムへはたぶん半自動的で，すなわちわからないところがあれば利用者に対話で問い合わせることによって実用的なプログラムを作ることができるようになると思っています．こういうものがあればいいんじゃないかなというふうに思ってます．

参加されている皆さんは，要求仕様の研究または，その実用化の立場に立たれる方が多いと思いますので，少しお願いをいたします．

だめなプログラマが理解できる方法で，割と現場に近い観点から，たとえば前回のシンポジウムの酒倉庫問題のような問題，それも一つじゃなくて複数の問題を各種の要求仕様方法で解いて，評価するなんてことをやっていただけると，私みたいなだめなプログラマにとっては有益だなと思ってます．以上です．

松本(司会)　どうもありがとうございました．

これで第1部のパネラの話を終りたいと思いますが，どうぞ活発な問題提起，ご意見を伺いたいと思います．

松谷(日鉄コンピュータ・システム)　今いろいろお話がありましたが，実際にシステムを作ったりしてきたことから若干意見を述べさせていただきたい．要求はソフトウェア工学の一つですから，エンジニアリングということで現場に役立つ必要があります．そうすると理論的なアプローチと現実的な問題とをある程度整理しながら，議論しなければいけない．たとえばライフサイクル・モデルに関する議論ですが，現実の大規模なプロジェクトになった場合は，ウォータフォール型のモデルで管理しなければならないところのが現実であります．現実は認めた形で出発しないと，議論が進まない．その中ではあと戻りも現実にあり得るので，チェック・ポイントをどうするかというような具体的問題をとらえていかないと，ウォータフォール型はだめだとか，現実に合わないんだという形の議論だけでは，現場で作業してる人たちの混乱を招くと思います．

第二は，要求定義についてですが現実的にいえば，何がしかの仕様書が各段階において必ず必要だと思います．これは関係者間の連絡のためにも，分業体制においては不可欠と考えます．それぞれの工程でいかに関係者にわかりやすく表現されてるかということが大事なことじゃないかと思います．この観点から要求手法なり設計手法へのアプローチがあると考えます．要求定義や設計手法は，実際，使ってみると，いろんな意味で参考になります．いろんな観点から物事を考えることができるようになる．実務では極端にいえば全部の観点が必要になるというのが，現実じゃないかと思います．

第三に，阿草先生のお話から思い付いたんですが，システムを作るのは家を建てるとか，ビルを作るとか，土木建築という仕事と一番よく似てるんじゃないか．つまり完全にはマスプロ化できない．しかし部品があり，ツーバイフォーのような早く作れる工法がある，しかし一方，創造性も必要となる．そうすると土木建築でよく見られるのは，模型を作ってお客さんに見せる，全体の模型を作る場合もあれば，部分だけ，たとえば装置だけの模型を作る場合もある．これには二つの意味があって，お客さんに見せることと，自分たち自身が，何かそれで不都合がないかを見つけることということがある．以上感じたことを述べました．

松本(司会) 今のご意見に対して，パネラ，会場の皆さんからお話はないでしょうか．

今のお話を聞きながら感じたことを申します．建築物，ハードウェアの製造とのアナロジについて言及されましたが，ハードウェア，あるいは建物と基本的に違うところは，ソフトウェアは作り替え，書き替えができるというところです．何千人もいるソフトウェア工場を抱えて生産管理をやらなければいけない．ソフトウェアだけじゃなくて，インタフェースのハードウェアも一緒に作っておりますから，ハードウェアと同じ生産管理過程の中においてソフトウェアを管理しなければならない．このような場合全く同じ生産管理ができるかというと，最も違うところはソフトウェアが作り替えができるというところです．したがってソフトウェアをハードウェアと同様に管理するために，一つのドキュメントができますと，それを凍結するというか，ある過程を通って承認されますとアンタッチャブルにしてしまうとかいう規則が必要になるわけです．構成管理というのは，実はそういう手法でありまして，あるベースラインを越えると構成要素をアンタッチャブルにする．上流工程に戻って，再度直すときには，アンタッチャブルのものを，タッチャブルにしてからでなければ変えられない．タッチャブルにするには再び承認を得なければならない．これが構成管理の基本的な思想です．おっしゃるとおり建物やハードウェアのようにソフトウェアも作ろうとしています．ただソフトウェア特有の問題を含んだ管理をしていかなくちゃいけないという点が難しいと考えております．

松谷 アンタッチャブルにしなければならないというのは，これは現実問題としてそうしなければならないというご認識なんですね．ソフトウェアは変えられるからということでずるずるいってるかあるいはそれを容認してるというところに問題があるわけです．たとえば大きな設備の建設みたいなところですと，極端なことといえば工期の問題からいっていついつまでに設備の大きさを絶対に決めないと困る，ここで建築を建てないとあとの工程に間に合わないというようなことがある．ソフトウェアでも同じようなことがある．今いわれたようなハードウェアの製造とリンクした上でソフトウェアを凍結するということと通じる．なんらか，外部要因で変えちゃいけないという領域が出てくると思います．ソフトウェア自身からくるのでなくて，むしろ工期など，別の要素からアンタッチャブルに対するニーズが生れる．このニーズに基づいて，アンタッチャブルの問題は扱うべきと考えます．

森澤 ソフトウェアの開発を建築にたとえられているんですけれども，要求仕様という観点からみると建築の場合，要求仕様は作る前にきちんと書いた設計図だと思います．設計図というものをとおして，完全に分業がなされてます．設計図の上でああでもないこうでもない，だから設計図は書き変えられる，彼らはそういう設計図という共通の言語を持っているんですね．それで書くとあとどうやって作ればいいかができてきます．

ソフトウェアの場合においても，たとえばこういう要求仕様で書けばあと自動化できる．またはそれを書けば誰もが同じように作れる．そういうような要求仕様定義の方法，または言語あたりができてこないかぎり，建築と同じようにやることは，私は無理だと思います．建築全般とソフトウェアの開発工程全般と同じに議論するとわけがわからなくなると思いますが．

柴田(慶応大学) 野木先生に伺いたいんですけれども，オペレーショナル・アプローチということで，従

来の what と how の代りとして，問題指向と計算機指向の分離をはかろうというのがオペレーショナル指向の一つの大きな特徴というお話をなさったと思いますが，ぼくも非常にそれに興味を持っているんです．しかし，一つ気になることがあります．

JSD はオペレーショナル・アプローチの代表的なものの一つと考えられるんですが，エンティティとアクションという二つでもって現実世界をとらえようということだと私は思います．実際エンティティとアクションでとらえられる問題領域も確かにあると思うが，とらえられない世界もある．ほかの見方でとらえたほうがもっとよい問題領域もある．いろいろな見方があると思いますが JSD では，エンティティとアクションという二つのものだけでとらえようとしている．本当に問題指向的な見方をするんでしたら，問題領域によって見方は，それぞれ異なってくると思うんです．ですから見方ごとに一つのモデルというものが存在して，それによって設計の方法が変わってくると思います．この点について野木さんのご意見を伺いたいと思います．

野木 操作的アプローチの考え方なんですけれど，JSD だけじゃなくて操作的アプローチで実世界をモデル化するという場合には，問題領域ごとに言葉とか概念が違うわけですね．それで概念とか言葉を定義する場合には，どういうふうにすればいいかということになると，ある抽象的な対象と，それに対する操作ですね，預金通帳だったら払出しとか預入れとか，そういうものによって概念を定義するわけです．その定義される概念はもちろん問題領域ごとに違ってくるわけですが，どんな分野でも概念を定義する仕方は同じじゃないか，それを基本にしてその分野固有の言葉をきちんと決めて，それを使って仕様をきちんと書いていくということをすればいいんじゃないかというのが基本的な考えなわけです．だからある程度そういう考え方は汎用性があるので，どんな分野でも通用するということが一つと，もう一つは見方がいろいろあるとおっしゃいましたが，もちろん見方はいろいろありまして，分野ごとに私も違うと思います．しかし，見方というのとモデルというのは別物でありまして概念的なモデルの作り方は同じだけれども，一つのモデルをどういうふうに見るか，どっちの断面で切って見るかが分野ごとに違うのだろうと思います．

まずモデルを作って，それを適当にその分野に合った断面で見てみるというやり方がいいのかもしれないし，あるいはその分野特有の見方を統合してある一個の統一的なモデルを作るというやり方がいいのかもしれませんけれども，モデルと見方はやはりそういう補完的な関係にあるのであって，一緒にして考えないほうがいいんじゃないかというのが私の意見です．

松本(司会) それでは時間的制約もありますので，後半の第2部に移りたいと思います．では森澤さんからどうぞ．

森澤 先の質問にもからむと思います．

要求仕様の方向，なんて大袈裟に申すよりは，私の興味の方向というほうが正解です．問題は一つのパラダイムでは対応できないと思います．たとえば数値計算と事務処理と同じパラダイムで解けるかとか，ジャクソン法は入力データ構造があって，出力データ構造があって，それからプログラム構造を導き出します．たとえばパラメータが一つ与えられて，それで一気に計算してぽんと答えを出す．これがジャクソン法のパラダイムで解けるかというと，まずだめでしょう．私は，問題においては，ある類似した問題ごとにパラダイムがあると思ってます．また類似した問題ごとにパラダイムを作るべきだと思ってます．現在興味を持ってますのは，さっきも質問がありましたが，問題指向型構造をとる方法です．これに興味を持ってるのは，多くの事務処理がこれで対応できるんじゃないかと思っているからです．私の割と近いところで，JSD の仕様の実行系が作られています．横から首を突込みながら茶々を入れています．

これがなぜかというと，たぶんご承知のことと思うんですけど，問題領域の構造と仕様の構造を同じようにする．仕様の構造から具体的構造へは規則的変換をやればいい．仕様記述用の言語と仕様から効率的に具体化への変換する記述言語を二つ作れば割とうまくいくんじゃないかなというアイデアがジャクソンから提示されてます．これが面白いと思ってます．

つぎに，runnable 仕様あたりのプロセス性指向型の言語と，あとそれから実際のプログラムに移るための何か変換の言語を作れば割といいスピードで変換でき，ある程度のパフォーマンスで動くものが作れるんじゃないか，そうするとプログラムを書くという立場からいいますと少し楽になるんじゃないかなと，そこら辺をやってもらいたいなという，希望を持ってます．

何をいいたいかということをまとめますと，まず，ものを作るというのは人，それからその人がどういう

道具を使うかができ上がったものに大きく影響を与えます．したがいまして人を育てるということと，よい道具を作るということをやってほしい．

10年ぐらい前ですと，日本ではパンチカードだと思います．それからちょっとして端末が入ってき，ワークステーションとかいうものが，はやり始めて，最近ですと，それを統合化して大型機の上でなんでもやってしまうという話があるようです．最近 LISP マシンを見る機会が多いのですが，あれはハードよりソフトのほうがずいぶん示唆を含んでいます．使い勝手を横から見てますとずいぶん参考になるものが多いようです．たとえばプロセス指向型の言語を作って，LISP マシンの上で動かしてみると，そのためにどういう環境がいるかというあたりがずいぶん参考になるんじゃないかと思います．したがって今までの単なるワークステーションじゃなくて，その上でたとえばrunnable 仕様がある程度実行できる．それから runnable 仕様をたとえば Cobol に変換するトランスレータがある．それででき上がったプログラムを大型機の上で高速で動かせばいいだろうと．そういうような Intelligent Amplifier Workbench，適当な言葉を付けましたけれども，そういうものを是非研究してほしいなと思ってます．そのためには単なるハードだけでなく，標準的な要求仕様言語のパラダイム，お手本も作ってほしいです．何かを学習するには，よいお手本をまねるのが最良です．

物事を抽象化できるということは，たとえばここから裏まで行くのに一つの道しか見えない人はたぶん物事を抽象化できません．いろんな道があってどういう道を通って行ったらいいか，そしてその道を一般化するとどうなるかという複数の道が見えない限り，たぶん物事を抽象化することはできません．それにはやはり訓練が必要ですし，いろんな方法を示してもらう必要があります．そういう規範の蓄積がほしい．当然最近はやりですが，AI 技術というのは一つのパラダイムだと思っています．そのパラダイムが一つ追加されたんで，使えばいいんじゃないか．やはり英語はしんどいですから日本語がほしい．

それから教育も，数学でたとえれば中学生の数学と高校の数学と大学の数学があるようにそれぞれの利用者のレベルに合わせていろんな教え方及びその教科書があります．そのような教科書を是非大学のほうで作ってほしいと思っています．

それからもう一つ，言語を話すには，仲間が必要です．だからどのような要求仕様言語でもコミュニティが必要です．そこで切磋琢磨する必要があるんじゃないかと思ってます．

最後におちこぼれプログラマに愛 (AI) の手を，というのが今回の締めくくりです．愛は AI にもかけてますし，IA にもかけてます．よろしく皆さん方のご研究及びご精進をお願いいたします．

野木 要求定義技術というのが生まれてきた経過を見てみますと，設計というのがあって，どうもそれだけではうまくいかないということで，それより前に何か仕様を決める段階がいるんじゃないかということでできてきたわけです．設計とは何か質的に違うものであるから分けたほうがいいんだということだと思います．そこでどういうふうに違うのかということをここでちょっと整理してみたんですが，設計と対比した場合の要求定義技術の特性ということで５つほどあげてあります．だいたい明らかなことが多いんで，それほど詳しく説明しなくてもいいと思いますが，一つは問題の世界と計算機の世界ということですね．それぞれの世界で使う言語が違う．それから利用者か開発者かということで当事者の関心が違う．利用者の関心というのは機能にあるわけですが，開発者の関心は計算機の上でどうやって効率よく動かすかにある．それから抽象化対具体化というのがありますが，問題の世界から計算機の世界に直接移すというのは非常に難しいわけで，何か抽象化してから具体化する，そういうステップをたどらないとなかなかうまくいかないという話になるわけです．

その前のほうのステップである抽象化の過程というのが要求定義にあたっているんだろうと思います．それから非形式的対形式的というのがありまして，これは先ほど出ていた仕様の形式性ではなくて過程の形式性ですね．要求定義というのは何もないところから仕様を作るわけですから形式的にできるはずがないということで非形式的．これに対して設計というのは仕様に対して妥当性の検証ができるような形式的過程である．それから最後の点ですが，モデルの生成と変換ということで何もないところから作り出すという生成の段階と，いったんできたものを最適化していく変換の段階に分かれるだろうということです．

要求定義技術の将来のめざすべき方向を考える場合，この特性を生かすような方向で，それぞれの特性についてどんなふうに進めていったらいいかと，そういうふうに考えたいわけです．全部はできませんの

で，興味のあるところだけやりますが，まず一つは利用者対開発者ということで，今回のシンポジウムのテーマでありますプロトタイピングはやはり利用者と開発者のコミュニケーション・ギャップを解消するには非常に有効な方法だと思います．したがってこれをやはりなんとか将来は生かしていきたいと思うわけですが，現在プロトタイピング方式というのが三つ考えられております．

一つは逐次改良方式といいまして，プロトタイプを作って，それをどんどん改良拡張していって最終プログラムにしてしまう，まあ要求仕様みたいなものは作らないという感じのやり方ですが，これはやはり場当り的なやり方でありまして，小さいシステムではいいんですが，大きいシステムではとても保守ができないということで問題があるわけです．この逐次改良方式というのは，従来のライフサイクルを否定するという立場ですね．

それに対して二番目の使い捨て方式は，従来のライフサイクルとプロトタイピングという柔軟な考え方を併用させていこうという方式で，プロトタイプによって，仕様を固めていくというやり方ですね．

それから三番目の仕様実行方式というのは，使い捨て方式のようにわざわざ別にプロトタイプを作るのは開発コスト上大変だから，実行可能な仕様でプロトタイプを兼ねてしまおうという方式です．

これはうまくライフサイクルの中にプロトタイピングというものを取り込んでしまおうという発想のわけで，コスト的にもできれば一番いいわけですね．本当にこれができるかということが問題になるわけですが．

仕様実行方式の課題として，どんなことがあるかといいますと，一つは解釈実行系の高速化と書いてありますが，どうもよい仕様であるほど実行効率が悪いという傾向があるような感じがします．なぜかというと，普通プログラムの実行効率がよいというのは，アルゴリズムが入っているからなわけです．ところが，アルゴリズムというのは発見的に見つけ出すものですからなかなか難しいわけで，仕様を考えるときに，いちいちアルゴリズムを考えていたんでは大変なわけです．

それでアルゴリズムが入っていない仕様というものをまず簡単に作ってしまってそれを最適化していきたいという話になるわけですが，当然仕様というのは，そうすると非常に実行効率が悪い．ちょっとしたものでも，実行するのに何時間もかかってしまうという問題が起こるんじゃないかと思われます．だから解釈実行系の高速化をもう少し本気で考えないと仕様実行はうまくいかないんじゃないかという懸念があります．

それからもう一つは，解釈実行系で実行しても，直接利用者に見えるような形になるわけではないので，プロトタイピングのツールとしては不十分だということです．今日の発表でもありましたけれど，インタフェースを生成するようなツールとか，そういう周辺ツールの開発も非常に重要だろうと思います．

次にモデルの生成と変換について話したいと思いますけれども，モデルを効率的に作るための有効な方法として，どんなものがあるかといいますと，概念の階層化というのがあるんじゃないか．階層化といいましてもいろんな種類の階層化がありますが，ここで有効だと思われるのは一般化，特殊化という階層化ですね．一般的な概念を作っておいてそれを特殊化して使うというメカニズムがあると，かなりモデルの生成というのは楽になるんじゃないか，モデルの生成を0から始めるというのは非常に効率が悪いんで，なんとか既存の概念を利用して効率的にモデルを作っていきたいわけです．そのための現在の技術としてはオブジェクト指向言語なんかでスーパ・クラス，サブ・クラスというものが提案されておりまして，技術的にはできていると思うんですが，むしろ問題はそういう技術を使ってこれから各問題領域における基本概念をいかに体系化して標準ライブラリとして整備していくか，誰かシニア・プログラマがそういうところはきちんと作っておいてくれて，利用者はそれを簡単に使えるような形にしていかないと，とても実用的な技術にはならないわけです．各分野におけるそういうライブラリの整備というのがどれだけできているかということによって実用性が決まってくるんじゃないかと思います．そういうふうにライブラリを作った場合に，ある概念のより一般的な概念は何かとか，類似概念は何かとか，そういうものをうまく捜し出せて再利用できるということが非常に重要になってくるんで，その辺の技術の確立ということもこれからの課題だと思います．以上です．

阿草 要求仕様の今後ということでは，やはり仕様化というのが，ドキュメンテーションを書くわりには効果がないということをよくいわれますので，やっぱりこの過程の効率化ということが必要だと思います．そのためには要求仕様化の再利用化というのは分析

過程も記述も含めての再利用です．これは今日の発表の中にもあったと思いますが，同じようなシステムを作るときには前のが使える．また私は別の意味が少しありまして，正しくいった仕様じゃなくて失敗の仕様のほうが再利用できるのではないか．正しい仕様は同じようなシステムにはぴったり合いますけど，違ってますと，あわない．失敗の歴史が人間を作ってるのであって，成功の歴史は同じ人をフォロしても同じだけしか成功できないわけですから，失敗の部分の再利用もできたらおもしろいなと思います．私はあまり AI をよく知らないんで，AI で失敗の部分ばかりを使って，ワールドがクローズしてるかどうかということにも関係するかと思いますが，そういうところがおもしろいのではないかと思います．

工程の自動化といって，よく仕様書を作れば，プログラミングは自動的にいくとか，設計が自動的にいくとかということをよく議論するんですけど，そもそも要求仕様書というのは，現実社会とのインタフェースをとったはずである．そういう意味では効率化をどこでやるかといえば，保守作業の効率化，自動化というのが一番ある意味で最初に改めやすいんではないか．設計とかいうのはやはり計算機と密接に関連してますから，世界が違いますので，そこらを少し考えたいと．

それからさっき質問にありましたけど，確かに建築などは非常にいい例ですが，松本さんが以前に発表されてたとき，建築を例にとっておられました．建築のそれじゃ模型を作ったらなぜ，同じものだとわれわれはコミュニケーションできるかというのが問題で，それはたぶん3次元のわれわれが住んでる世界とうまくイメージがよく合う，すなわち家に一度も住んだことがない人が，家の模型を見せられて，これでいいですかといわれても全然わからないと思います．われわれの生活の中で家というものはどういうものかというのを知ってるんで，その模型がうまく評価できる．そういう意味ではわれわれは，runnable であるとか，executable とかよくいってますけど，たとえば Prolog で書いて1＋2が0のサクセサ，サクセサのサクセサであると答えが出たときに，これを実行したとわれわれが解釈するのかどうかということが非常に難しい話です．もともとセマンティックスの議論をやってるときに，オペレーショナル・セマンティックスが出てきて，その計算機の中のメモリの全部の値を一つの状態と見なして，状態から状態への変化ということで何か意味を付けようとしたら，そのメモリ・セルがあまりにも多すぎて議論ができなかった．それからディノテーショナル・セマンティックスになって，ファンクショナル・セマンティックスしか与えられていなかった．今，executable であるとか，runnable であるとか，実行可能であるとか，いろいろいわれてますけれども，われわれのコミュニケーションというか，スペシフィケーションを与えるときのセマンティックスにそういうものがそのまま使えるとは思えない．これは東北大学の伊藤先生がおっしゃってるんですが，たとえばファイナイトステート・マシンであるとか，ペトリネットであるとか，そんなアプローチにいくら頑張ったって，大規模なソフトのセマンティックスを定義できない．もっと別のセマンティックスを今からわれわれは定義する必要があるのではないか．じゃあそれは何かといわれると非常に困るんですけど，まあ，いってみれば模型を，どういう模型を作ったらみんながわかってくれるかというのを，もう少し考えましょうと，そういう意味で，このセマンティックスの定義をやりたい．そのセマンティックスの定義をやれば，そのセマンティックスの上で議論することが形式性があるということであると．いわゆる解釈実行系が定義できてるということです．

それから要求仕様の今後ですけど，自動化がどんどん進むと，結局要求仕様をやるということが，今度は次のプログラムのフェーズになるわけですね．昔機械語で書くことから，アセンブリという部分のリンクが取れるようになってアセンブリという世界ができて，それからフォートランのコンパイラができて，あれは自動プログラミングといわれたわけですから，今仕様化ができてしまったら，今度は仕様をどう書くかというのが今度はプログラミングに代るわけですね．そういう意味ではプログラムの intelligent programming environment という研究を一生懸命やってるんですけれども，intelligent requirements specification environment というようなことをやっていかないといけないんじゃないか．そのためには結局データベースを入れたという話になるのかもわかりませんが，まあそういうようなもの．それから各種の管理，レポート機能．それともう一つ，ここでいった run とか，execute, runnable であるとかいうのを，どう定義するかという，じゃあその runnable というのを使うことによって，テストができる．今 requirements specification を与えたのがいいかどうかというテスト

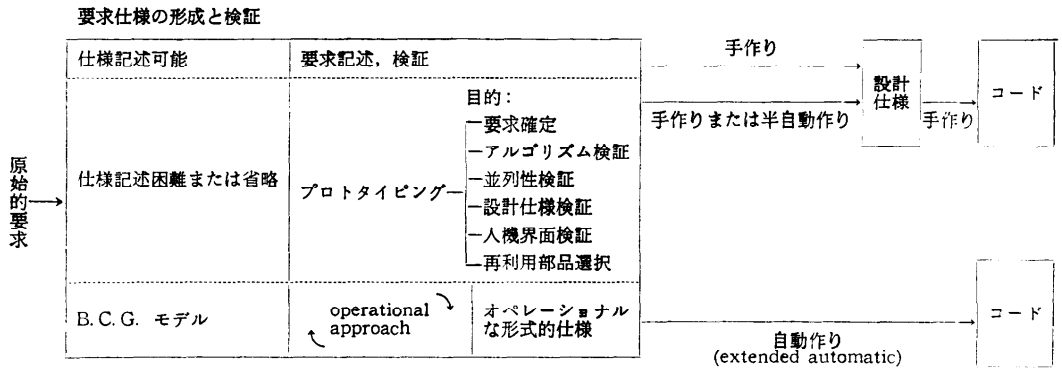


図-3　要求仕様形成法の分類

ができる．そういう実行系というのを定義していきたい．結論は今からコミュニケーションというのが，どうされてるかということに対するセマンティックスを与えるような何かを，今から考えていく必要があると思います．

松本(司会)　それではこのあと皆さんに討論をお願いするわけですが，その前に討論内容を整理して紹介したいと思います（図-3 参照）．

左端に示すものは，ユーザの原始的要求といいますか，全く形式化も何もされてない生の要求があり，それをなんらかのフォーマリズムによってまとめた記述が要求仕様です．要求仕様の形成と検証がこの大きな箱です．箱の中を分類してみます．なんらかの形式で記述できると考えられる部分については仕様が作成されています．あるものについては，検証までしている．IEEE std. 830-1984 規格は一つの形式を与えています．実務では，厳密な意味のフォーマルではありませんけれども，なんらかの意味での仕様記述が行われています．要求にヒューマン・ファクタが入ってきますと，文章で書きにくいような要求というのがクローズアップされてきます．仕様記述困難，仕様を書くのは大変，工程的にも非常につらいというときには，それを省略して，とにかく何か動くものを作って見せたい．こういう概念をプロトタイピングといってるわけですけれども，発表されているプロトタイピングを整理してみますと，要求を確定するという目的は共通ですが，それ以外に，アルゴリズムを検証，タスクがたくさんあるようなリアルタイム・プログラムにおける並列性の検証，設計仕様の検証，人間と機械とのインタフェースの検証，再利用部品の選択などの目的があります．それぞれの目的に基づいたプロトタイピングがなされています．以上のようにして作られた要求仕様は手作り，あるいは人間がきちんと介在した半自動で設計仕様に落され，それからコードが作られます．

最後の BCG モデルというのは，Balzer, Cheatham Green という 3 人の提案によるオペレーショナルアプローチという方法です．これは操作的な仕様を作りながらでプロトタイピングをやってユーザに見せる．最終的にできるものはこの形式的仕様でありまして，これができますとコードを，extended automatic という方法で生成するというものです．

以上のことをベースにして，討論をお願いいたします．

松本(日本電気)　私どもは日常仕事をしておりまして，いつも悩まされ，またシビアな立場に立たされてるのは，プロトタイピングと要求分析の定義を目的オリエンテッドな視点でとらえていないためだと考えてます．解法オリエンテッドにあまりにも目がいきすぎてはいないだろうか．要求仕様がなぜ必要かには大きな理由が二つある．一つはよりまともな仕様を得たい，要求の確定をしたい，いろんな検証もしたい，検証性を高めたい．もう一つはヒューマンとヒューマン，ヒューマンとマシンというよりはヒューマンとヒューマンの場においては意志疎通をきちんとやりたい．Computer 誌で Hasse という人がそういうことを書いていますが，ヒューマンとヒューマンの意志疎通をもっともっとやりやすくするために，要求仕様というものを改良したい．さらにマン・ツー・マンや，マシンを含めた世界において要求定義技術のアプローチが見出せるのか，是非お知恵を拝借したい．

第二の質問は，分析ならびに定義に関して，作業をやるための枠組とかモデルがある．たとえば環境とか，ツールとか，支援環境，こういう枠組が動くとき

には多くの知識が利用されます．このような知識を今日のソフトウェア工学の実力で満足に用意していけるというふうに見ておられるのか，どのくらいの時間の長さで分析とか定義とか，その先のコード形成の自動化をやるために必要なもろもろの知識をちゃんと用意していけるだけの技術がソフトウェア工学によって見出していけるのか，どのくらいの時間で期待し得るのかです．知識がないことには枠組だけでは，あるいは方法だけでは，あるいはモデルだけでは，われわれは飯が食えない立場です．

落水(静岡大学)　私もその辺をお聞きしたいと思ってたところでして，過去の5年くらい前の失敗例を申しますと，実体関連モデルという枠組を想定してデータベース・センタ環境を作り，構成管理をサポートすることによって，ライフサイクルの行き戻りを支援することを考えました．1年ぐらいたってやめたんですけど，そのときに一番問題だったのは，今のご質問の後半と関係する話なんですが，データベースの中にどういう情報を入れるかということが問題になりました．通常いわれてる方法論とか，モデルとか，言語とかいわれてるものを入れてもあまり細かいところに行き届くような，ポイントになる情報が入らないんです．それで，必要な知識を得るために，作ったシステムを理解して変更することに関するものに限定して調査を行いました．通常の生産現場で使われてるドキュメントを媒体にしまして，そのドキュメントがどうやって作られたか，メンテナンスのときにどう利用されてるかというようなプロセスのモニタリングをやって，今知識を集積してるところです．すでに3年か4年やりました．優秀な院生3人か4人を使って，だいたいわかったことはたとえば事務処理分野の50万ステップぐらいのソフトウェアです．まず最初にわかったことは開発時に作られる文章というのが，かなり冗長であるということです．メンテナンスに使われてるところというのは，2点ぐらいしかない．ソースにかなり近いレベルの情報と，それからユーザの世界に近いレベルの情報．そのユーザに近いレベルの情報はドキュメントすら残ってない．これは想像で整理してわかったことは，実はユーザがメンテナンス時に要求してくる機能と，ソースの変更場所との対応関係には実にシンプルな構造があるんだということ，これはメンテナンスの環境を作っていくとき使えるだろうこと，たったこれだけがわかりました．通常のライフサイクルの工程を踏んで作っていくとき，作業の短縮などのために，このような対応関係が見出せるのかを今後やりたいと思っています．このようなことが非常に大事なんじゃないかと思います．このような知識を集めるためにはコストがかかるということもわかりましたが，これをやらねば話が進まないというふうに思ってます．

表-1 Software Design Description の構成とユーザとの関係

Entity attributes	Design users						
	PROJ MGR	CONF MGR	SOFT DES	PROGRAMR	UNIT TST	INTG TST	MAINTNR
Identification	×	×	×	×	×	×	×
Designer		×		×			×
Source	×						
Type	×	×				×	×
Purpose	×	×					×
Function	×		×			×	
Subordinates	×						
Dependencies		×				×	×
Resources	×	×				×	×
Interfaces			×	×	×	×	
Processing				×	×		
Data				×	×		
Tradeoffs							×
Assumptions							×
References				×	×		

IEEE Project 1016, Recommended Practice for Software Design Descriptions (Draft), (May 1985) より引用

松本(司会)　ヒューマンファクタの問題と知識の問題が出ましたが，先のご質問のなかの前半についてご意見がないので，コメントいたします．

アメリカの IEEE では software design description (以下 SDD と略称) の標準を作ろうとしています．これが前に公布された reguirements specification の標準よりも1歩前進してると思われるのは，SDD のユーザを**表-1**のとおり7つにクラス分けしていることです．表でわかるように，各ユーザに対して，どういうアトリビュートが必要かが定義されています．要するにそのドキュメントを使用者の視点に従って使い分けられるように書かせるのです．ヒューマンファクタ改善の一例といえます．

花田(NTT)　阿草先生にご質問いたしますが，まず大学では要求定義ということをどういうふうにして

教えておられるか，あるいはもし教えてなければ，これから教えようとしてるのは，どういうことかをお聞きしたい．

阿草 私，大学院で情報システム工学というのを大野先生と一緒に教えてるんですが，そこで情報システムというのは，システム工学と全然変わらずにシステムの分析ツールとかいろいろ出てますので relevant tree であるとか，それからＫＪ法まで含めて，そういうインタビュ技術というのが非常に重要だよという話をやります．そのあと各種の仕様書作成のテクニックでたとえば SADT なんかですと，あれでたぶん一番むずかしいのは何をコントロールとしてとらえて，何をデータとしてとらえるのが全体のバランスがいいかということが一番むずかしいと思います．勝手に思い付くままやりますと上からくるコントロールが足りないから適当なものをコントロールに入れたりするので，そういうものがどういうものが，その中でメインのフローであるかというのをどうとらえるかというのを教えます．そのあとでこういうものはやってみないとわからないからということで PSL を用いて，実際にいろんなシステムを書かせます．卒業論文でやったシステムであったり，去年は阪神が優勝するためのシステムでしたか，何かそういうシステム，いわゆる情報システム全体じゃなくて，何か身近なもののシステムをあるドキュメント体系で書いてみるという授業をやります．ただしそれが，さっきいわれた意味で，それを書けばうまくソフトウェアにつながるかどうかという点から見て書けたかどうか，いわゆるシステムエンジニアとして書けたかどうかというのは問題ですけれども，一応自然言語以外の記述言語で，ある社会のシステムを書こうとする努力をさせてみると，そういう意味での教育はやっております．

玉井(三菱総研) 今のお話とちょっと関係して，松本さんが最初に実世界モデルと，実現モデルという問題提起をされて，野木さんがその間を埋めるというか一致させる見通しとして，モデルを抽象化すると一致するのじゃないかと述べられました．私もそういうふうに期待したいんですが，ちょっと無理なんじゃないかと思います．野木さんに伺いたいんですけれども，実世界をなるべく素直に記述するモデルというのが大事だというのは非常によくわかるんですけれども，しかしそれで実動するか，素直にインプリメンテーションにつながるかどうかというと，やはり多くはつながらないし，物による．シミュレーションみたいなものが一番単純で，割とつながるんだろうと思いますが，多くはそうはいかない．私は，昔数理計画モデルをやってたんですが，そこでは非常に解法という意味では強力なモデルがある，ともいえますが，なんでも実際のモデルを強引にそこへ引き込んでしまうというところがありました．それは悪いんですが，非常に便利は便利ですね．そこに持ってくるととにかく解けるわけです．モデルには計算メカニズムとか，しっかりしたアルゴリズムをもつモデルと，そうじゃなくて現実世界を素直に写すのに向くモデルがある．両者間には絶対ギャップがあって，そこの間をうまく変換できればいいんですが，やっぱり難しい．人間がそれぞれに向いたモデルを用いて，現実世界を写したモデルは役に立たずじまいになるという気がするんです．

野木 モデルの変換ではなかなかうまくいかないんじゃないかという話ですが，私は原理的には大部分のものがうまくいくんじゃないかと思っています．実用的にはいちいち実世界から抽象化して作っていくというのは面倒くさい作業で，前にやった経験があれば，アルゴリズムがわかってるわけですから，いちいちそんなところからやらなくてもできちゃうわけですね．だからそういうときにはもちろん実世界のモデル化なんていうところはとばしてやってしまうほうがずっといいと思います．実世界のモデル化が重要だというのは，やっぱりよくわからないときの話だと思うんですね．原理に立ち戻って考えれば，そういうアルゴリズムがわかってるものも，最初やったときには実世界の分析からやって，いろいろそういうノウハウを蓄積していったんじゃないかと思っているわけです．

松谷(日鉄コンピュータ) こういう要求定義の技術の実用化の見通しはどうでしょうか，プロトタイピングについてもいかがでしょうか．

松本(司会) 玉井さんの調査によりますと，10年かかるということであります．今は種々のトライアルが行われている段階ではないでしょうか．現実にコマーシャルのソフトウェアで要求定義技術が有効に作用しているという例はきわめて少ないんじゃないかと思います．プロトタイピングについては 図-3 の仕様困難のところ，人間とのインタフェースのところが大部分を占めているようなソフトウェアあたりからだんだん使われてくるんじゃないかというように考えております．

座長 最後に感想を２点述べて終りたいと思います．一つは松本さんが漫画が非常にうまいとびっくり

しました．もしかしたら要求定義をやる人は漫画がうまくないとだめなんじゃないか，なぜかといいますと，漫画というのは非常に抽象化してる．それからもう一つは，非常に論点といいますか，主張点をポイントをはっきりさせているというふうなことから，どうも要求定義以前に私は漫画をかく勉強をしたほうがいいんじゃないかなと思います．

もう一つは，私は全然要求定義がわかりませんので，素人として申しますと，これは営業用のツールじゃないかと思います．つまりお客様の要求じゃなくて願望を EDP 化でやればこうなりますよというのを，いち早くお客様に持って行って，だからわが社と手を組んでシステムを作りましょうというふうな，これは営業用のツールだなという感じがいたしました．これら二つを学ばせていただきまして，これでパネル討論を終りたいと思います．松本さん初め3人のパネリストに対して感謝の意をこめまして拍手をもって終りたいと思います．どうもありがとうございました．

（拍手）

参考文献

1) 山崎利治：共通問題によるプログラム設計技法解説，情報処理，Vol. 25, No. 9, p. 934 (Sep. 1984).