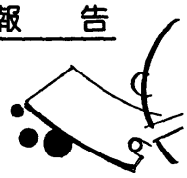
報　告



パネル討論会

人工知能とプログラミングの接点†

パネリスト
岩元　莞二[1]，杉本　正勝[2]，奥乃　博[3]
米澤　明憲[4]，國藤　進[5]
司会　諏訪　基[6]

諏訪　このパネル討論のねらいは，最近大きく取り上げられるようになってきた「ソフトウェア危機」という問題に対処するために，1950年代後半から行われている人工知能研究が培ってきた種々の技法とかそこで整理された概念に期待できるとすればそれは何なのか，また，これから人工知能のどういう分野がそのために重要となってくるのか，という点についてディスカッションしようというものです．独断と偏見があってもかまわないのではないかと思います．

ソフトウェア生産の活動を大まかなステップで捉えると**図-1**のようになるのではないでしょうか．つまり何か問題が与えられたとして，その問題を分析・理解して記述する部分と，それを計算機で実行可能な形式にするためにプログラムを書くという部分があります．このソフトウェアの生産性を扱うのがソフトウェア工学なのでしょうが，それは一方ではソフトウェア生産のための人的資源をどう管理すべきかという問題意識があり，他方ではプログラムの信頼性，効率，保守のし易さなどの問題意識も当然あって，守備範囲が実に広くなってしまっています．

人工知能では，問題解決，推論，知識表現論などの研究が盛んに行われてきており，それぞれ種々なパラダイムが提案されてきています．最近は知識ベースシステムの研究から知識プログラミングというものもあり，人工知能向き言語の開発がこれらを支えてきています．私の感じでは，プログラミングに問題解決機構が直接導入されるのはまだ先のことと思われますが，人間が行う問題の分析・理解・記述といった作業を支援するのに，人工知能の研究成果を徹底的に活用するのが第一ステップではないかと思います．そのための糸口は，人間の思考活動を理解してモデル化を試みる認知科学の研究の中にも見いだすことができるようになりつつあります．

ソフトウェア危機に関連して今後ますます比重を増してくるのは，このようなソフトウェア生産活動の前段に当たる部分，つまり問題を分析して記述するあたりではないでしょうか．私は，プログラミングと人工知能との間の接点で大切なのはその辺だと考えています．

まずパネリストの方に順にお話していただきます．ソフトウェアの現場に近い所におられる方からもっぱら人工知能と取り組んでおられる方へという順序でお願いしたいと思います．

岩元　私は10年来ソフトウェアの生産性や品質をあげようということをやってきました．現在もソフトウェアのCAD/CAMということで，人間に分り易い図形をマンマシンインタフェースに使い，ソフトの設計・製造の過程をサポートすることを進めています．振り返ってみると，従来からソフトウェア工学ではいろいろやってきているが，主にプログラムに近い所が中心になっていました．しかし生産性に対する要求は3～5倍にして欲しいということだが，実質はそれほど上らず，あるアメリカの報告では年率4％の生産性向上となっています．これは，要求定義から設計，製造，テストの各作業をダイレクトにサポートしないと生産性は上らないということで従来と違うアプローチが必要だと感じています．つまり，これまで主な対象になっていた製造工程以下では表現されるものがフォーマルな形式，多くはプログラミング言語であり，ここでサポートされるのはフォーマルなシンタックスに基づくツール，言わばシンタックスオリエンティドツールということができます．設計工程についても設計言語というフォーマルな世界に持って行こうとして

---

† 日時　昭和59年11月22日の「知識工学と人工知能研究会」（場所　機械振興会館）で行われた内容を各パネリストが編集したものである．

1) 日本電気ソフトウェア生産技術研究所，2) 富士通研究所
3) 電電公社武蔵野電気通信研究所，4) 東京工業大学
5) ICOT，6) 電子技術総合研究所

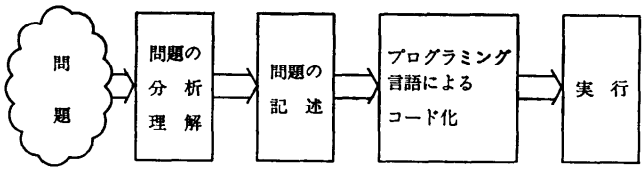


図-1 ソフトウェア生産の活動

いるのだが，設計に必要なセマンティクスを十分には表現しきれません．そこで高度な支援のためにはもっとセマンティクスを考える必要があります．

ソフトウェア開発は雲のような漠然とした要求から始まるわけで，インフォーマルな世界からだんだんとフォーマルな世界に変換していくのがソフトウェアの開発だと思うのです．したがってインフォーマルな世界のサポートのために，もっとセマンティクスオリエンティドなツールを開発する必要があります．私たちは，手始めにプログラムに近いレベルからこの方向を目指して検討しています．図-2 は銀行プログラムの仕様の一部ですが，よく読むとあいまいな表現であり，不足している情報もあります．この関係のプログラマはこのインフォーマルな表現をプログラムに落とすのですが，私たちはこの作業を何とか機械でやれないかと考えていまして，まだ実験レベルですが COBOL のプログラムを出力できるようになりました．

これまでの経験から考えて「人工知能とプログラミングの接点」で欲しいものは，知識表現とその操作メカニズム，自然言語理解の手法です．ソフトウェアの開発は人間が中心で，その人間は自分の言葉で考えるので，そこを理解するのが必要です．そこをフォーマルな言語で書こうとすると非常に長いものになってしまいます．したがって，省略の補完を含めた高度の自然言語理解が必要で，そのバックグラウンドの技術として知識表現とその操作メカニズムも必要になるわけです．また，人間は文章のほかに図（特に線図形）や表も使うので，文章と関連づけてそれらの理解も必要になります．特に図は他人にわからすための例を示すことが多いので，図の真の活用には例題からの一般化を行う学習メカニズムの導入が望まれます．

人工知能の実用化という観点からは，メタコントロールということが重要です．私の勝手な解釈ではメタコントロールに二つの意味があります．第一は学習と関係しており，システムに新しい知識を追加するとき，従来の体系と融合するようにシステムの内部をコントロールする．第二に，一つのツールでサポートできるのは非常に限られたソフトウェアだけで，ソフトウェアの種類ごとに一つずつツールを作るわけにいかないので，ツール類もその利用環境に合ったツールを生成するメタツールにしたい，そのためのコントロールという意味があります．

実用化への課題をまとめると，人工知能はだいたいが発見的な方法であって完全な解を与えてくれず，不完全な解をそのままシステムに組込むわけにはいかないということです．完全な解にするためには問題をうまくサブセット化する，その問題の中では完全になるようにするということ，そしてシステムが分らない部分は会話形のインタラクションで補うことができるということです．また，環境に適応するしかけが必要です．

これらの問題は，AI を使おうと思っている側の仕事だと思いますが，AI の側でも関心をもってもらうと助かります．いずれにしても，ソフトウェア工学の人たちは AI に非常な期待感をもっています．

杉本 現在のソフトの作り方と，今後の新世代ソフト開発技術との間には，図-3 に示すような関係があると考えます．例えば要求仕様，詳細設計仕様については，5年ほど前までは手書きで書くしかなかった状態から，現在はコンピュータの中で仕様書を編集できるという状態になってきています．これからはコンピュータの助けを借りて，何か出てくると考えられ，それが AI 研究の進展という流れの中でうまく展開されれば，もっと加速されると思います．図-4 にはいくつかの解決策を示します．今日のパネル討論の検討点は工数の推定とかプロジェクトマネジメントではないわけですが，これらも AI 的な流れの中で解決されていくだろうと考えています．

ソフトの開発・保守全般では自動合成，自動検証，ラピッド・プロトタイピング，性能測定，品質推定などで新しい展開が期待されます．AI の代表的なアプリ

普通預金のとき 表-1 の組合せ以外はエラー "403" でリターン．

表-1

処理区分	操作媒体
有帳	G，K
無帳，特別訂正	C
	O

図-2 プログラム仕様の例

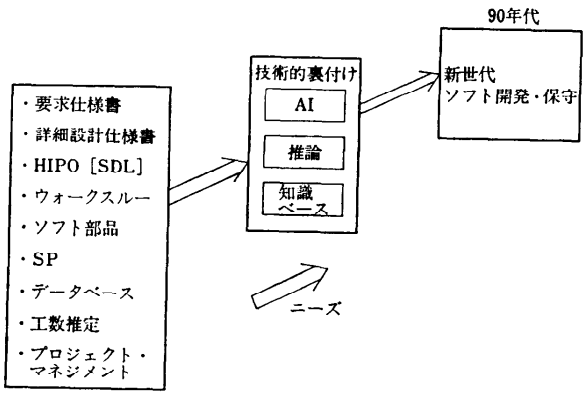


図-3 人工知能技術とソフト開発・保守技術

- ラピッド・プロトタイピング
- 実行可能ドキュメント (executable specifications)
- オペレーショナル・アプローチ
- 高機能検証機能
- 最適化技術

図-4 新世代ソフト開発・保守
—ラピッド・プロトタイピングからのビュー—

ケーションに，機械翻訳，エキスパートシステム，CAI などありますが，エキスパートシステムという考え方から「ソフト開発・保守用エキスパートシステム」という考え方が出てきます．

AI 研究の成果をコンポーネントとして使って，新たに実現できそうな機能を探すと，ソフトウェア工学の分野ではラピッド・プロトタイピングがあります．最近は**実行可能ドキュメント**という言葉もあるように，Prolog のような言語でプログラムを書くと，それがそのままドキュメンテーションにもなることが考えられています．ラピッド・プロトタイピングで実現できるソフトウェアとか実行可能ドキュメントは実行スピードは当然遅い，ただし，いったん仕様がそれで良いとなれば，最適化技術にまかせることができるようになると思います．

自動合成や半自動合成は分野によってはすでに行われているところもあります．例として通信ソフトを考えてみます．**図-5** の中央，下側がハードウェアで，アセンブラがあって，CHILL という PL/1 のような言語があるわけです．更にドキュメントにいくつかの標準があります．状態遷移図を書くところでは SDL という国際標準の書き方があります．このようなポイント，ポイントがあると，自動的に変換しようという話が出てきて，現実に研究が行われています．つまりドキュメントの記述レベルや用語が固まっていることが重要なのです．こう考えてみると，自動変換については分野を限り，言葉を定義していけば，相当なことができると思えます．

これからはドキュメント体系，ここではプラットフォームと名前をつけているのを気軽に作れるツールが必要ではないかと思います．これは，要するに一つの言語を作るみたいなもので，今までは大変なことだったのですが，AI の技術を用いて手軽にできるようになってくると思います（ユーザとのインタフェースで自然言語も重要です）．プラットフォームを手軽に構築するには，AI の成果の一つと言える DCG や BUP が応用できると思います．プラットフォーム間の変換機能に例えばモンタギュー文法による変換規則が使えるとか，論理プログラミング言語がパターン形の変換には強いといった話があります．現在のソフト開発では部品化が切札のようになっているのですが，部品化は AI 技術を用いて発展すると思います．一つの部品が多くの用途に使えるジェネリック機能とか，与えられた状況のもとで最も性能のよい部品を選んでくる技術とかと結びついてくる気がします．ソフトウェアで WHAT と HOW という話がありますが，WHAT と HOW をどう結びつけるのかというのは，今後 AI に

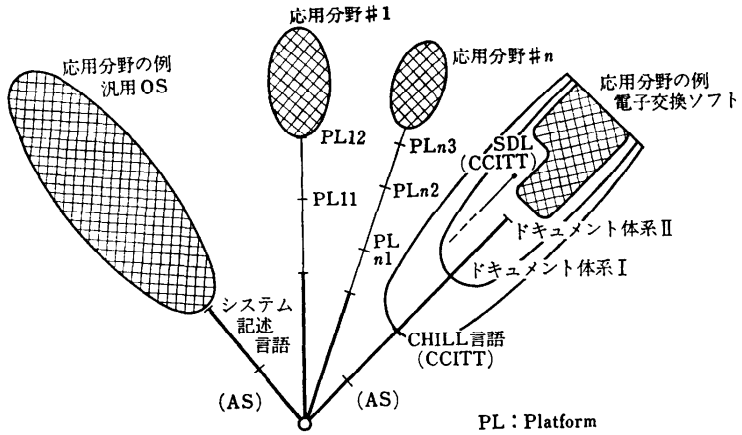


図-5 ドキュメント体系（プラットフォーム）とソフトウェアの自動合成

大いに期待する課題だと思います.

奥乃 私は「人工知能とソフトウェア工学」という面から問題提起を行いたいと思います. ソフトウェア工学で問題となっているソフトウェア危機とは, well-structured な対象に対して well-structured なアプローチで取組むのがうまくいっていたのが, 対象が大規模化, 複雑化し well-structured でなくなり始めると今までの well-structured なアプローチではうまくいかなくなった情況であると認識しています. 一方, 人工知能研究では, ill-structured なアプローチを積極的に容認しています. 人工知能研究の対象である人間の心は完全な記述が不可能であり, また, アルゴリズムも明確ではありません. そのために, 人間の認知モデルや情報処理モデルを立てて, 記号処理のプログラムを用いてシミュレーションを行うわけです. シミュレーションの結果を見てモデルやプログラムの修正を行うというフィード・バックを繰り返したり, あるいは, うまくいった場合には, 対象領域を広げるために同じことを繰り返します. 人工知能研究でのプログラミングの位置付けは物理学や化学での実験に相当していると言うことができます. 人工知能の対象分野が条件分岐が非常に多く構造が明確でない問題とか碁のプログラムのように非常に難しい問題であるので, 得られた解が inconsistent でも incomplete でもあるいは, near-optimal であってもかまわない. 極言すると, 何かわからないけれどもうまくいっている, つまり, automagically ということが許されているわけです. 先ほど, 岩元さんが「incomplete な解を AI によって complete な解として見つけたい」とおっしゃいましたが, 私が見ている限り AI とは automagically, near-optimal でよいという居直りがあるのではないでしょうか. この居直りを容認することがソフトウェア工学と人工知能との接点の第一歩だと考えております.

実際に応用されつつある人工知能のアプローチとしてヒューリスティックスをデータあるいは知識として独立して扱おうと手法があります. いわゆる知識工学と呼ばれる分野ですが, 知識を顕在化されることによってシステムの開発や保守が容易になるというメリットが出てきます. 人工知能研究ではプログラミングが実験であると申しましたが, 実験の対象が知識の組立てにあるわけです. 知識工学の応用としてさまざまな分野のエキスパートシステムや最適化システムが開発されています.

最後に人工知能とソフトウェア工学とが協調して生まれた二つの重要な概念について指摘しておきます. 一つはオブジェクト指向プログラミングであり, もう一つは論理型プログラミングです. オブジェクト指向という考え方はいろいろな分野でほぼ同時に生まれ, お互いの問題意識を持ち寄って一つの概念となったものです. プログラミング言語では Simula から Smalltalk-80 が, 人工知能では, Minsky のフレーム理論からさまざまなフレーム記述システムが, プログラム意味論からは並列計算モデルのアクタが発表されました. 論理型プログラミングについては Prolog がよく知られておりますが, その他にプログラムの自動合成や自動検証もその目標の中に含まれると思います.

人工知能からソフトウェア工学へのインパクトは, ここ10年くらいはそれほど画期的なことはないと思われる. その中で注目されるのはプログラミングによる実験を支援するプログラミング環境とデータベースに対する知的アクセスではないでしょうか. いずれにしても, ソフトウェア工学との接点は, complete なものから near-optimal への発想の転換から生まれてくるものだと考えます.

米澤 東工大の米澤です. 前に話された人々と趣きを変えるためにキーワードだけをまず掲げます (**図-6** 参照). CS は Computer Science, そして AI, KE は Knowledge Engineering, DB は Data-Base, MS は Management Science です. これらは, レベルがそれぞれ異なりますが, (1)知識の**表現形式**と(2)知識／機能を表現した単体／構成要素／部品／モジュール等の**管理**, という観点からある意味で共通の問題にアタックしているのではないかという気がします. 大変, 荒っぽいですが…….

SE (Software Engineering) というのは相対的には, かなり well-structured あるいは well-defined な問題を扱っていて, AI はその先, すなわち, われわれがものを考えるメカニズムとか, われわれの知識や記憶の構造のモデルを考案して, それを計算機プログラムによって実験しているわけです. AI と SE の中

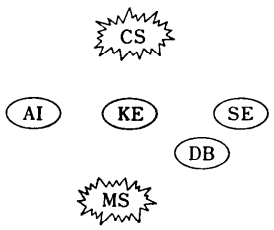


図-6

AI
☆知識の表現形式・表現言語
- frame, semantic net 等

☆最も複雑な構造をもつプログラムシステム
- management of programs
- society of programs
- society of minds

DB
☆世界をモデル化する
- semantic data modelling
- ER-model

SE
☆プログラム／プログラマの管理
☆設計論・方法論
- modelling
- prototyping

図-7

間にある(問題のレベルや複雑さの面でも)のがKEでしょうか. AIで基本的なことは，人間が持つさまざまな機能とか知識とかをどうプログラムとして，すなわち記号の体系としてどう表現するかで，その表現言語とか形式が常に問題になります．AI の歴史の中で，さまざまな形式が提案され，フレームとかセマンティックネットとかがいたるところで使われるようになってきました(図-7 参照)．もう1つ重要な点は，AI では，プログラミングの世界の中で一番複雑な構造をもったプログラムを扱わざるを得ないという点です．ですから，AI の世界ではプログラムの管理（マネージメント）が基本的に重要な要因であり，プログラミングのための良い“環境”などという考えも，エディタとか TSS とかなど，AI の世界からまっ先に出てきたものです．また，比較的小さなエキスパートの集まりが互いに協力し仕事をするという知識や問題解決の形態というものが AI の中でこれから，どんどん議論されるようになると思いますが，このときエキスパート集団の管理や，それを管理するエキスパートなどという概念も登場してくると思われます．

DB の本質は，世界のモデル化だといわれているわけで，それが関係データベースのような，ある種の表の集まりとして表現する考えや，さらに，最近の ER-モデルのように，AI を知っている人から見ると，AI ではかなり昔から議論されているような概念が登場してくる場合もあり，いずれにせよ，かなりセマンティックなモデル化に向けて DB が進んでゆくのではないかという気がします．ですから，これも知識の表現形式と，その表現の管理という問題が中心となっていると見られます．

一方，SE ですが，これはプログラム（あるいはプログラマ？）の管理・マネージメントという印象が強いですが，その段階に至る前に，システムをどう設計するかという点で世界のモデル化・記述という問題に基本的には最も重要な要素があると思います．システムの設計に関して，実行可能仕様とか，プロトタイピングとかいろいろ話が出てきていますが，これらも，世界のより簡易なモデル化とそのシミュレーションということになります．また，ソフトウェア一般でいえば，AI 的な手法や，その考え方がソフトウェア作りにだんだん浸透してきて，ソフトウェアと人間の間のインタフェースが，今までは人間がシステムに合せて使っていたのが，“ソフトな”というか，より知的なインタフェースをもったソフトでなくては売れなくなるという方向にいくんではないかと思います．この点は余談ですが重要なことだと思っています．

以上のように AI, DB, SE は，基本的には，知識とか機能とか，それを体現している世界のモデル化という点と，それらがどんどん複雑になっていったとき，複雑さの限界を越えて，先に進むためには，それらを，どのように表現し，管理・マネージするかという点にかかってくるのではないかと思います．

ここでやっと，私の立場が出てくるのですが，AI, DB, SE などに共通する，知識の表現と，その管理という問題は，“オブジェクト”とか“オブジェクト指向”という概念で考え直してみると，もっと先に進めるんじゃないかと思います．私のように昔から“オブジェクト”という考え方の研究をしてきた人間にとっては，世の中がだいぶ開けてきたなという感じが最近してきました．時間(と紙面)の関係で，“オブジェクト指向”とは何かなどという話はできませんが，宣伝で恐縮ですが，私が，ソフトウェア科学会の機関誌『コンピュータソフトウェア』の創刊号に書いた論文を読んでいただければと思います．また，現在の私の研究室の研究テーマは，オブジェクト指向型のプログラミング言語を設計して，それで並列システムの記述を試みたりソフトウェアのプロトタイピングの方式を考えたり自然言語理解システムを作ったり，文章の理解モデルを提案したりしているところです．

國藤　パネリストの中で一番 AI よりの人間として，AI の世界からプログラミングの世界へ寄与したことをご紹介したいと思います．AI 研究の歴史を振り返ってみますと，60年代前半まではゲームとパズルの世界，60年代後半はロボットの時代，70年代は言語と知識の時代，80年代は知識工学と認知科学の時代と言われています．80年代は知識工学的ブラックボックス・アプローチと認知科学的ホワイトボック

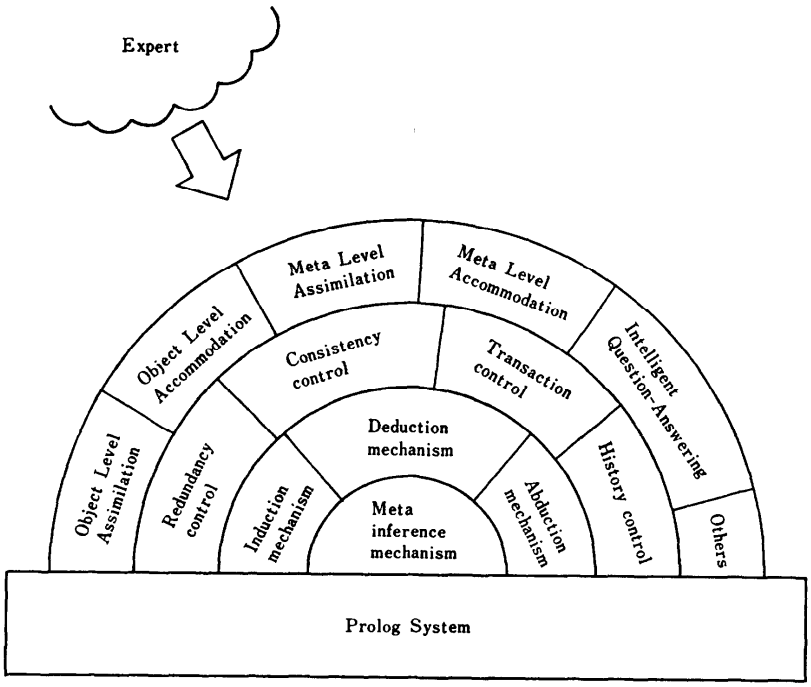


図-8 Kaiser における知識獲得機能

ス・アプローチとに分化しつつあります．前者はある入出力仕様を満足するものならばどんな形でもよいからその機能を工学的に実現しようとする立場で，後者は入出力仕様のみならずその機能の処理過程そのものも人間の思考過程を忠実にシミュレートしようとする立場です．前者の本質は記号処理の本質を追求する立場で，後者の本質は認知過程の本質を追求する立場です．両アプローチの共通課題は知識の表現技術，知識の利用技術，知識の獲得技術の確立にあります．私自身の所属しているグループは知識ベースの利用や知識獲得に焦点をあてた Kaiser (Knowledge Acquisition-oriented Information SuppliER) という知識獲得支援システムを研究試作中ですので，その経験を踏まえ，AI の世界からプログラミングの世界へ寄与したことを述べたいと思います．

第一の寄与は論理型言語による AI 技術の見直しにより，論理的な知識獲得方式が確立しつつあることです．人間の問題解決過程の本質的処理を記号処理過程とみなすと，発想・演繹・帰納の過程があります．未知の驚くべき事実に遭遇した人間がその事実を説明する仮説を生成する過程が発想です．生成された仮説に基づき導出される結論を推論する過程が演繹です．ついで得られた結論に基づき，与えられた仮説を検証するために帰納という過程が実施されます．論理型言語は演繹処理が得意なのは周知の事実ですが，制限された論理的世界を前提とする限り，それは帰納処理も比較的相性よくサポートしうることが最近になって実証されました．その結果は知識の同化，調節，均衡化という論理的知識獲得方式としてまとめられ，11月に行われました第5世代コンピュータ国際会議後の ICOT オープンハウスでもデモンストレーションされました．

第二の寄与は処理対象とする知識の拡大です．ホーン論理で扱える知識ひとつをとってみても，データベースで利用可能なファクト型知識，知識ベースで利用可能なルール型知識，あるいはファクトやルールの使い方に関するノウハウ型のメタ知識があります．メタ知識とは知識の使い方に関する知識であり，そのための方法論をメタプログラミングとして提案してきました．通常の推論エンジンの上にメタ推論エンジンを構築する方法論を開拓し，先ほど述べた知識の同化や調節といった知識ベースの更新をサポートできることを実証しました．

第三の寄与はコンピュータ・エイディド・プログラミングからヒューマン・エイディド・プログラミングへの発展という方向性を，AI で生まれた言語が加速したことです．プログラミング言語は最初，機械とのインタフェースをとる言語として2進処理の機械語やアセンブラ言語が生まれ，ついで数値処理向きの Fortran，データ処理向けの Cobol，リスト処理向けの Lisp とか，知識処理向けの Prolog へと発展してきました．この流れは基本的には，より人間に近いインタフェースをもつプログラミング言語（その究極は自然言語かもしれませんが，）への機械の側からの接近で

す．AI 向けの言語やツールの開発の途上で得られた種々のノウハウが，これらの言語に埋込まれていきました．

第四の寄与はプログラミングの世界への新しい考え方の導入です．先ほど，米澤先生がおっしゃられたので簡単に述べますが，エディタとか TSS の考え方は AI の流れの中で生まれたものです．最近のホットな話題としてはオブジェクト指向プログラミングがあります．ICOT では ESP や Mandala というユーザ言語を提案していますが，それらは論理型言語上でのオブジェクト指向プログラミングをサポートするものです．AI 向きのプログラミング言語はオブジェクト指向プログラミング，ルール指向プログラミング，データ指向プログラミングといったプログラミング・パラダイムを融合することの重要性を常に喚起し続けてきました．

最後に今後期待されるべき第五の寄与として，高級言語の使用によるソフトウェアの生産性向上が挙げられます．私の周辺のハッカーの経験則によれば，どんな優れた人間も一人の人間がすみずみまで熟知しプログラムできるコード量は，言語に依存せずたかだか一万ステップ以内だそうです．しかもこのことは，人間が瞬時にアクセスし管理できる知識の総量として成立するそうです．この経験則は，人間は年を取れば取るほど，ソフトウェアの生産性を保つためにも，より使い易く考え易い高級な機能をもつプログラミング言語を使用せざるを得ないという必然性を示しています．

諏訪 一通りパネリストの方からの話を伺いましたので，ディスカッションに入ろうと思います．まずパネリストの方から口火を切って頂けますか．

奥乃 米澤先生は「知識をどう表現するか複雑度をどう乗り越えるかという問題に対して，人工知能の世界の蓄積を外に出せばよい」とおっしゃったのですが，私は人工知能の世界の蓄積は ill-structured なものをそのまま ill-structured なものとして認めようという居直りの上に成り立っているものと考えています．ですから，人工知能の成果はそう簡単には外に出ていかないのではないでしょうか．また，どちらかと言うとコンピュータサイエンスの成果を取り入れることによって人工知能の方が更に発展していくのではないかという気がしています．

米澤 AI から出てきたものは，例えば，フレーム，プロダクション・システム，オブジェクト，インヘリタンス，セマンティックネットなど，いろいろあるではありませんか．またプログラミング環境の新しい概念はほとんどすべて，AI のハッカーたちから生れたのではないでしょうか．関連しますが，最近 AI をやりたいという学生が研究室にたくさん来ますが，彼等はプログラムのまともな作り方を知らない場合が多いようです．これはむしろ逆で，ソフトウェアを作るセンスというか，方法論的な考えがしっかりしていないと，AI や KE は本当はダメだということが，あまり認識されていないのではないかという気がします．

國藤 ICOT が招へいしたエジンバラ大学の Alan Bundy 教授は ICOT の研究開発活動をつぶさに観察して，PSI や Delta 開発グループを総称して第4世代グループと報告書に書いていました．ICOT で現在行われていることは第4世代の成果，すなわち AI の世界の外で成功したプログラミング環境——例えば DEC 2060 の OS である TOPS-20 上のエディタ，デバッガ，とかその他のコーティリティ——を，まず AI ツールの上に吸収していく．第4世代の成果を結晶化したプログラミング環境を提供し，そこで本当の AI に興味をもっている人たちが生んだ独創的な知識情報処理技術の卵をかえし，それを知識情報処理システム構築に必要なひなへと育てていこうというのが ICOT の基本戦略です．

奥乃 人工知能の成果の大部分はプログラミング環境を充実するツールのレベルを越えてはいない．例えば，Lisp マシンや知識表現言語などは出てきているが，人工知能の具体的応用は特に少ない．そういう意味で外に出せる成果は少ないと言ったわけです．

会場より 日本タイム社の高田と申します．ソフトウェアハウスという現場におり，ソフトウェアを作る立場から聞かせていただきましたが，ピンとこなかったことが多いので，具体的な問題で考えてもらいたいということで申し上げます．ソフトウェアを設計する場合，最初ドキュメントを書きプログラムするわけですけれど，プログラムする直前では必要なすべての情報はドキュメントに書かれていなければならない——エラリー・クイーンのように直前ですべての手がかりが与えられているので，あとは犯人が当るはずだ——ということだったと思います．あらゆるテストツール，たとえば自動検証でも出来上ったプログラムについてテストするわけですが，使う以前に条件が足りないことがドキュメントの段階でもあるのが現状です．そのあたりを AI，知識工学にやってもらいたい．

また建築ではまず構造的な検証をして，その後の間

取りなどはどうやっても間違いないのですが，ソフトウェアではここまでやっておけばあとは細かいので安心だという基準がない．知識工学で支援してくれればありがたい．そして小さい建物の場合は構造設計しなくても間取りから始められるわけですが，ソフトウェアの場合もこれは元から設計しなければならない，これは途中からやればよいという基準とかルールができてくると非常に役立つと思う．

諏訪 最初の問題は，岩元さんの「フォーマルな世界では Syntax-oriented ツールがあるが，その上のインフォーマルな世界では Semantic-oriented ツールが必要だ」という話に関連すると思いますので岩元さんどうですか．2番目の問題はプログラムの自動合成にも関心をお持ちの杉本さんにお願いします．

岩元 言われた要求は，まさにソフトウェア工学に関係している人の考え方ですが，そのところはまだ未開拓なわけです．製造から上はインフォーマルワールドと言いましたが，それと大きな問題はオープンワールドだということです．私の紹介したシステムもデモはできるのですが，現場で本当に使ってもらうには辞書が十分でありません．ソフトをなぜ作るかというと今までそういうソフトがないから作るわけで，そのためには新しい知識が必要になります．知識を共通に使える知識と個別に必要な知識にうまく分ける，そして対象としている世界について複数のモデルを設定して，モデル対応に知識を集め，問題が違えばこのモデルを変えるということでうまく対応できれば良いと思っています．

杉本 最近では要求仕様について，論理的な矛盾の検出をする研究が始まっています．また設計仕様についても SRI などで興味深い研究が進行中です．これらは形式的なアプローチなので，比較的に早い時期に実用されるかと思います．

諏訪 この問題は，一般的にいえば，問題がだんだん複雑になってきた時にそれをどう扱ったらよいのか，プログラムとしてどう表現するのかという問題になりますが，先ほどオブジェクト指向のパラダイムが有望だと話された米澤先生，いかがですか．

米澤 有効だと思います．そう思っているので，私たちも研究してるわけで，かなりよい結果も出ています．今，杉本さんから設計言語のお話がありましたが，最初，頭の中にあるモデルがすっきりしていれば，それがそのまま素直に表現できるオブジェクト指向言語を直接使われてもよいのではないかと思います．

奥乃 ソフトウェアの部品化については，Unix では豊富なライブラリとして提供されており，それをパイプでつないで使うわけですし，Smalltalk-80 では2000以上のオブジェクトが提供されており，やりたいことはこれらを使えばできるようになっているわけです．問題は，どのライブラリ，どのオブジェクトを使えばよいかが使いこなしていない人にはわかりにくいことです．人工知能の対話技法を使い，より快適なオブジェクト指向システムを提供することが必要だと思います．

会場より 安立電気の内藤です．一つの方法論として仕様を日本語とか形式言語で表わし，マシーンで走る形式に変換する方法を採られている方もいるのですが，私は疑問をもっています．というのは仕様自体が完全でなく，プログラミング中に仕様の細部が明らかになることが多い．そして形式言語で書いたとしても結局 How しか記述できず，保守を考えた場合には Why とか What が必要になります．それと形式言語が出来た場合でも仕様の質が問題になり，一つ上のレベルで仕様の質とか方法論が議論されてくるだけであり，繰り返しになると思う．したがって，もっとプログラミング過程に注目し，プログラムを書くことが仕様を書くことになるようにしないと，レベルが上っただけで本質的な解決にならないような気がします．

岩元 お話のようなものができれば良いのですが，一つの言語であらゆること一仕様を書いてプログラムにも落とせるもの一は無理なようで，実は人間がどうやっているかを考えてみても，もやもやしたところからプログラムまで速くないわけです．何回も何回も議論して細かく詰めていくので，設計の中にも実は何百という作業単位があると思います．そのあたりをできるだけ多く自動化するのが良いと思っています．

杉本 自然言語，形式言語からプログラミング言語という話ですが，自然言語は機械で扱うようになるには，長い時間かかるという気がします．形式言語的な設計言語の研究とともに，プログラミング言語にオブジェクト指向機能やプロダクション機能，フレーム的データベース等を取り込む研究も重要だと考えます．

諏訪 最後に今日のパネル討論のまとめをお一人ずつにお願いします．

米澤 会場の方から保守の場合，Why までゆかなければダメだというお話がありましたが，まったく賛成です．そのためには，問題領域のモデルがシステムの中にちゃんと表現されていなければなりません．そ

のモデル化とそれを機械上で実現し実行させるのに，オブジェクトの考え方が有効なのではないかと思っているわけです．

杉本　あいまいなところ，不完全に記述されているところを取り扱おうという立場と，きちんとは記述はできるが今までのソフト作りでは大変だったところを扱うという二つの立場がAIとどう関係するかという議論だったと思います．前者のサポートという点ではエキスパートシステムを応用する立場が独立してあるでしょうし，後者に対してはLispとかPrologで開発したツール的なものがプログラム生成で使われると思います．きっちりしたところはツールがよく定義されていれば使えるということで，比較的近い時期と思います．

今日の話を通じて，AIとソフトウェア工学とが結びつくことは確実のように思います．そして，関連する知識を顕在化させたり，整理する技術の研究が重要だと思います．ちょうど，機械翻訳が辞書ができ，構文が整理されればできてくるように，知識の整理や学習機能が重要であると思います．

國藤　ICOTオープンハウスで小学校3年の国語の文章題をもとに談話理解実験システムをデモしました．そのプログラム開発を担当していた人たちが何度も発していた言葉に「計算機正しい．うそつかない．」というのがあります．論理的なプログラミング言語の使用は人間のあいまいな矛盾した知識をあいまいでない無矛盾な知識に変換する助言を与えます．また不完全な知識を完全にするにはどうしたらよいかの手がかりを与えます．そのような用途でもって論理プログラミングを大いに使っていこうと思っています．論理の世界は前提が誤っていると，どんな結論も真となる，つまり発狂するわけです．論理プログラミングを修得してくると，次第に自分自身発狂しない範囲を意識しつつ，高度な機能をもつソフトウェアの開発が容易に行えるようになります．

本日のパネルでは時間の関係で十分に議論できませんでしたが，パネルで取り上げられましたAIとプログラミングの間に派生する種々のギャップはツールの整備と方法論が知識ベースの蓄積という二つの方向で，次第に解消されていくと思います．前者についてはICOTの活動がトリガとなってどんどん整備されていくでしょう．後者については実際的なアプリケーションをかかえている人たちが，整備されたAIプログラミング・ツールで，そのアプリケーション向きの方法論や知識ベースを蓄積する努力が，今後ますます必要になると思います．

奥乃　AIとソフトウェア工学の接点はIA (Intelligence Amplifier) にあると考えます．すなわち，プログラマの知力を増幅するためにAIの成果を活用することです．ソフトウェアの生産性を向上するためにモデルを記述する，あるいは形式言語で記述するということが提案されておりますが，これらのことはプログラムを書くことよりずっと難しいことです．いくつかのプロトタイプを用意しておき，それをIAによって洗練化することでプログラムが完成するというアプローチが重要であると思います．

岩元　AIはシステムの表に出ないで陰に隠れているべきと思います．人間とのインタフェースは自然なもの—自然言語，図，表—で，そこから意味を取り出したりするのにAIが働くべきで，特定の言語を押しつけるとうまくいかないと思う．ソフトウェア工学でもいろいろな言語が出てきましたが，ちょっとしたことをやるにも厚いマニュアルを読まなければいけないという問題があります．自然な表現の理解に関して，私は現在プログラムに近いレベルをやっていますが，もう少し上のレベルでも，例えばマニュアルを読んでテスト項目作成用の原因と結果の対応リストを生成するなど，自然言語理解のテクニックを使えるのではと期待しています．ソフトウェア工学の方でも，大勢の人々がAIの手法を取り入れようとしていますので，これから接点もますます密になると思います．

諏訪　いろいろなことが浮彫りになったと思います．私は，計算機の応用範囲が物凄い勢いで広がりつつあって，扱いたい問題をどう計算機に取り込んだら良いかという問いに応える技術が追いついていない——というのがソフトウェア危機ではないかと感じました．最終的な研究の対象は人間ではないかという気がします．

コンピュータサイエンスは実験科学だという主張がありましたが，私もそう考えています．印象としては日本ではそういう認識がまだまだ足りないのではないでしょうか．したがって実験を行うためのインフラストラクチャとしてのコンピュータネットワークのようなファシリティの整備も遅れているのでしょう．

人工知能とプログラミング技術とがその接点においてお互いに補完し，刺激し合いながら一層発展することを念願しつつ，パネル討論を終了したいと思います．

（昭和60年3月1日受付）