ソフトウェア開発への知識工学の応用

玉井 哲雄（三菱総合研究所）

1　はじめに

ソフトウェア工学という分野が誕生して早くも 20 年近くになるが（1968 年の NATO 会議を起点とすれば），このところやや行詰まりが見られることは，多数の認めるところであろう．この行詰まりを打破するのに，人工知能（AI）あるいはその応用分野としての知識工学との結びつきが期待されている．

20 年弱のソフトウェア工学の歴史に対して，知識工学という言葉は，生れてから約 10 年である．この語が最初に提唱されたのは，1977 年の国際人工知能会議 (IJCAI）における E. Feigenbaum の講演であった ([1]) ．一方 AI の歴史は，しばしば言及されるように 1956 年の Dartmouth における会議を起点として考えれば，30 年と見なせる．つまり，ソフトウェア工学は，AI と知識工学のちょうど中間時点で生れたことになる．

知識工学とソフトウェア工学との関係には，次のような双方向性がある．

1. ソフトウェア工学→知識工学

   AI ないし知識工学の成果は，ソフトウェアとして実現される．そのようなソフトウェアを開発するのに，ソフトウェア工学の考え方を利用できるし，またすべきであろう．

2. 知識工学→ソフトウェア工学

   知識工学の対象分野としてソフトウェア開発は，もっとも重要な分野の一つである．

ここでは，上記のうち 2 の視点を中心に考えることになるが，知識工学がソフトウェアによって実現されるという当り前の事実に，改めて注意を促しておきたい．

2　知識工学とは

2.1　知識の特徴

“ 知識 ”の取扱いは 1970 年代以降の AI 研究の一つの中心課題になったばかりでなく，応用面から改めて AI が注目されるきっかけとなった．そのことを概念的に明確に示したのが，知識工学という言葉である．

知識を核とするシステムを，知識ベース・システム (Knowledge Based System) という．とくにある特定の分野の専門家の知識を取り上げ，その知識に基づく判断をシステム化したものを，エキスパート・システムと呼ぶ．ここで知識とは何か，が問題となる．哲学論議に陥ることは避け，以下のような点に注意しておくことにしよう．

データと知識の比較

情報の蓄積形態として，データと知識との違いを考えることができる．データとは，それが蓄積されている形そのままで検索され利用されるものと考えられる．一方，知識は多くの場合，それがしまわれている形態そのままで利用価値があるのではなく，推論等の思考過程の中で利用されて初めて意味を持つ．

問題領域への依存性

かつてのAIでは，探索技法や定理の自動証明技法等，問題解決の一般原理の探求に精力が注がれた．しかし知識工学で強調する知識は，そのような一般原理よりむしろ対象とする問題領域に固有の個別的・経験的な知識を想定している．AIの歴史の中で，一般原理重視から個別知識重視へという大きなパラダイムの変化が70年代に生じた，とまでいう人もある．

AI分野の共通項としての知識

AIの主要分野には，自然言語処理，画像理解，音声理解，等の多様なものが含まれる．これらの分野に共通するものがあるとすれば,“ 知識 ”の取扱いをその筆頭に挙げることができよう．自然言語にせよ画像にせよ，それらが表現している対象の世界に対する知識を考慮することなしには，その表す意味を十分に，かつ効率よく理解することはできない．そして，その知識の表現や利用方法には，共通の枠組みが役立つ可能性が高い．

知識と手続き

知識は，それを利用して問題解決に導く手続きとは分離したものとして，一般にとらえられる．従来のソフトウェアも，ある意味では知識の塊であった．しかしそこでは，問題領域の知識とそれを利用するための制御的な手続きとが一体化している．それに対し，知識指向のシステムでは，知識を独立した対象として扱い，一定の形式で記述する．その表現は，宣言的な形を取ることが多い．このように知識が分離独立することにより，新たな知識の追加や修正が容易になり，システムの柔軟性が増す．

2.2 知識表現

知識工学の最も重要な課題は，知識をいかに表現するかにある．知識表現の方法により，その上で可能となる推論の形態も決まってくる．代表的な知識表現法には，次のようなものがある．

1. 論理表現
2. 意味ネットワーク
3. プロダクション・システム
4. フレーム
5. 決定木

これらについての説明は多くの文献にあるので，それに譲る（たとえば文献 [2]).

2.3 エキスパート・システム

このような知識工学の方法ををを特定の専門領域に適用し，その分野の専門家の知識を知識ベース化したシステムが，エキスパート・システムである．2.1であげた問題領域への依存性や知識と利用手続きの分離といった特徴は，そのままエキスパート・システムの特徴でもある．さらに，多くのエキスパート・システムは，次のような特徴を持つ．

- あいまいさを取扱うこと．あいまいな知識を取扱う技術は，実はまだ充分に発達した段階とはいい難いが，確信因子を用いる方法，ファジイ論理，などが用いられている．

- 説明能力を有すること．システムの行う判断に関し，システム自身がそこに至る過程や理由を説明する能力を持つ．

典型的なエキスパート・システムは，図 1 のような構成からなる．

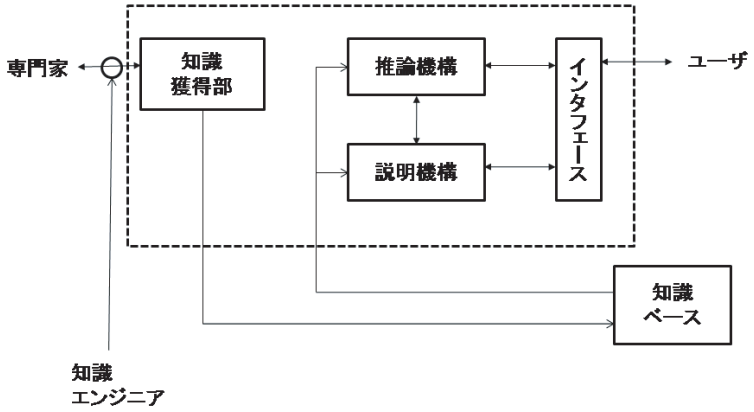

図 1: エキスパート・システムの構成

ここで，専門家が直接，知識獲得部を介して知識ベースに知識を入れることは，一般に難しい．これを媒介するのが，いわゆる知識エンジニアの役割である．なお，図中で枠に囲った部分は，個々の知識分野によらず汎用的に使えるソフトウェアである．これを通常，エキスパート・システム・シェルと呼ぶ．これは，エキスパート・システムからその中身（知識ベース）を取り去った殻（シェル）という意味である．

3 AI におけるソフトウェアの研究

AI の研究対象領域の一つとして，ソフトウェアは比較的古くから取上げられてきた．実際，AI は人間の知能を解明し，それをコンピュータ上に実現することを目的とするのだとすれば，ソフトウェアの開発はすぐれて知的な作業であるから，まさに AI の対象分野としてふさわしいといえる．

3.1 自動プログラミング

自動プログラミングは，AI の一つの重要な分野である．たとえば，人工知能ハンドブック ([2]) でもこれに独立した章が与えられ，かなりのページが割かれている．

自動プログラミングという言葉の起源は古く，当初は Fortran や Algol とそのコンパイラを指すものとして，使われたという．その後，この流れではコンパイラ・コンパイラや超高級言語などが，この名で呼ばれたこともある．

一方，1960 年代末以降，人工知能やソフトウェア基礎論をベースとするプログラムの自動合成技術が研究され始めた．以下で紹介するのは，この流れに含まれる自動プログラミング技法である．

(1) 定理証明に基づく方法

プログラムの仕様は，論理式を用いて形式的に次のように表すことができる．

入力 x の満たすべき条件： $P(x)$

出力 z の満たすべき条件： $R(x, z)$

このようなプログラムを作ることができるためには，$P(x)$ が成立するときに，$R(x,z)$ を満たすような z が少なくとも一つ存在しなければならない．すなわち，次の論理式が，真である必要がある．

$$\forall x(P(x) \supset (\exists z R(x,z)))$$

この定理の証明を構成的に行うこと，すなわち z の具体的な形を与えることにより定理を証明することと，この仕様を満たすプログラムを作成することは，ほぼ等価の作業とみなせる．この認識のもとに，定理の機械的証明図からプログラムを生成するという方法が，ここでいう定理証明に基づくプログラムの自動合成の考え方であり，R. Waldinger 等により，導出原理を用いた証明結果からプログラムを合成する方法が示されている ([3]) .

このような定理証明を応用したプログラム合成は，理論的にきわめて美しいことが，その特徴といえる．しかし，定理の自動証明が一般には実用的な効率レベルにないこと，プログラムの生成過程が人間にとって分りにくいこと，といった問題点がある．

なお，Prolog は，定理証明過程をそのまま計算機構として利用しようとするものである．すなわち，出力条件を満たす解を構成的に証明して求める過程から，別のプログラムを合成するのではなく，具体的な入力に対してその証明過程を個別に適用して，結果を得るものである．

(2) プログラム変換法

プログラム変換法は，与えられたプログラム仕様に対し，あらかじめ定められた変換規則の集合から適切な一連の規則を選んで，次々と変換を施し最終的に求める機能を果たすプログラムを得ようとするものである．この場合，初めに与えられた仕様も一種のプログラムとみなされる．ただし，それはプログラムの機能を明確には記述しているが，そのままでは実行不可能であるか，性能的にきわめて劣るものである．

変換法の代表的なものは，Darlington 等による方法である ([4],[5]) ．変換の対象となるプログラムは，関数型の言語によって書かれる．変換は，６個の限定された基本変換規則の組合わせによって行われる．変換規則の中でとくに重要なのは，関数の定義式による展開を意味するほどき (unfolding) という操作と，その逆操作としてのたたみ (folding) である．

別の研究には，Manna と Waldinger の仕事がある ([6]) ．合成システムとしては，DEDALUS が試作されている．DEDALUS は Darlington のシステムと比べると，変換規則を一定のセットに定めていない．それだけ柔軟性はあるが，システムとして形式性・論理性に欠け，その記述能力や正当性の検証，等の分析に問題があるように思われる．彼等のアプローチは，変換規則の使い方まである程度自動化するところに重点を置いている．

なお，Manna と Waldinger は，定理証明をベースとする方法においても多くの業績があり，最近はむしろこの方向に力を入れている（たとえば [7]).

ミュンヘン工科大学の Bauer 等は，CIP(Computer-Aided Intuition-Guided Programming) というプロジェクトで，仕様から実行レベルまで一括して記述できる広範囲言語を定め，その上の変換法を多角的に研究している ([8]) .

論理型プログラミング言語に対するプログラム変換の適用については，玉木と佐藤による研究がある ([9]) ．ほかには，南カリフォルニア大学 ISI の R. Balzer 等が，変換法を核とする自動プログラミング・システムの研究を行っている ([10]). これら変換法全体のよいサーベイに，文献 [11] がある．

(3) 知識ベースによる方法

知識工学に基づくことを明示した研究もいくつかある．Stanford 大学では，C. Green 等が 1970 年代の半ばから PSI という知識工学に基づく自動プログラミングを目指した大規模プロジェクトを，実施した ([12]). PSI の後，Green は Kestrel Institute という研究所を起こして，PSI のアプローチを引継いだ CHI というプロ

ジェクトを始めた ([13]). さらに，Reasoning Systems という会社を起こして REFINE というシステムを作り，これらの研究成果を企業における実際的なソフトウェア開発に応用することを，目指している．

PSI の主要メンバーで，その中で PECOS というプログラム合成の核となるシステム ([14]) を開発した D.Barstow は，その後 Schlumberger の研究所で，Φ_0 という限定された分野におけるプログラム合成システムを開発した ([15]).

これらのシステムで用いられているプログラム合成の手法は，広い意味でプログラム変換法に属する．すなわち，知識は変換規則という形式で整理されていることが，一般的である．定理証明による方法にしても，論理による表現に従い，証明に必要な知識が整理され用いられているわけであるから，これを知識ベースによるアプローチの一種とみなすことも可能である．

ここにあげたものが，(1) や (2) にあげたものと比べて特徴があるとすれば，以下の点であろう．

1. 人間がソフトウェアを作る過程で用いる知識に近いものを，なるべく使おうとしていること．
2. 自然言語による仕様作成など，形式化しにくい部分もシステムにとりこもうとしていること．

ほかには，プログラムの仕様ではなく，実行の例題を与えて，それからプログラムを帰納的に生成しようとする方法もある ([16]).

3.2 学習

AI では，学習は非常に大きなテーマである．古くから様々な研究があるが，いまだ応用が可能なほどの成果は得られていない．機械学習の技術が進めば，ソフトウェア開発における利用の可能性は高い．

実際，ソフトウェアの作り方自体を，コンピュータが自動的に学習することを利用したものに変えてしまうことが，考えられる．H. Simon は，ソフトウェアが大規模化，複雑化し，その保守が多大な負荷となっている状況に対し，機械学習によりソフトウェアが自ら保守し成長するようにし，人間の目からみればブラックボックス化していくことが，解決の道であることを示唆している ([17]). 機械学習技術がそのレベルに達するまでの道程は長いものと思われるが，将来的には，注目するにたる考え方であろう．

別の見方をすれば，機械学習は自動プログラミングに新しい方法を提供するともいえよう．たとえば D.Lenat は，自ら開発した数学上の理論を自動的に展開するシステム AM に関連して，プログラムをランダムに生成して，意味のあるものを評価・選択するような方法に対し，生成に合理的な方向性を与えられれば有効でありうると示唆している ([17]).

3.3 プログラムの検証

プログラムとその仕様が与えられて，その両者が論理的に合致しているか否かを数学的に証明するというプログラムの検証技術は，やはり AI に関係の深い技術である．1970 年代の後半には，Stanford 大学の D. Luckam 等による検証系を初め，多くの試みがあった．十分に実用的な段階にまでは至らなかったが，プログラミング方法論やプログラミング言語の設計にかなりの影響を与えた．

最近の ACM/SIGPLAN の Verkshop 特集号 ([18]) に見られるように，この方面の研究は現在でも続けられている．

4 ソフトウェア開発における知識の利用

前章では，AI の直接的な研究対象としてソフトウェアの分野をとりあげている例を概観した．ここでは，AI の中でも知識の処理技術に絞り，それをソフトウェア開発の様々な場面で適用するという立場から，どんな可能性があるのかにつき，みることにしよう．

4.1 一つのパラダイム

R. Balzer, T. Cheatham, C. Green の３人は，1990 年代のソフトウェア開発方法論として，知識ベースを核とした仕様作成，設計，開発というパラダイムを提唱している ([19])．そのパラダイムは，図 2 のように描かれる．

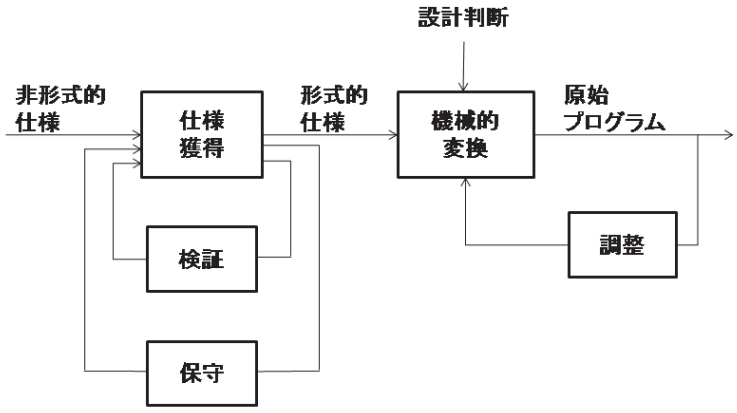

図 2: 新しいソフトウェア開発のパラダイム

この提案は，一つの概念を示したに過ぎず，これを実現するまでに解決しなければならない問題は山積しているが，将来への指針としては，参考になろう．

4.2 知識ベースの適用可能領域

ソフトウェア開発では，様々な知識が用いられる．知識を保有する人で見れば，マネージャ，SE，プログラマ，テスタ，オペレータ，教育担当者，等がある．知識の対象としては，ソフトウェア開発手法，ソフトウェア開発環境やツール，ハードウェア，OS，言語，アプリケーションの対象領域，開発要員，プロジェクト管理手法，ユーザ，等を挙げることができる．

ここでは，知識工学手法が適用可能なソフトウェア開発上の領域としてを整理するのに，二つの軸を考えてみよう．

一つは，一般のエキスパート・システムの機能分類としてよくとりあげられるもので，解釈，診断，教育，ガイダンス（コンサルテーション），設計，制御，予測，計画といった分け方である ([23])．

もう一つは，ソフトウェア・ライフサイクル上の各工程に対応するもので，要求定義，設計，プログラミング，テスト，運用・保守，プロジェクト管理，といった分け方である．

この両者の組合わせの上で，考えられる知識ベース・システムを例示してみたのが，表 1 である．

4.3 システム例

表１にあげたようなシステムの実例をいくつか，簡単に紹介しよう．

Programmer's Apprentice ([20],[21]) MIT の C. Rich，H. Shrobe 等が取組んでいるプロジェクトである．PA では，大規模ソフトウェアの開発に役立つようなプログラマの徒弟を勤めるシステムを，研究開発することを目指している．たとえば，プログラムを構成する要素の様々な形をプランと呼び，そのライブラリを利用して徐々にソフトウェアを作成することを助ける機能が，実現されている．また，この徒弟を使うのに，制限された英語表現を許す等，インタフェースを工夫している．

表 1: 知識ベース・システムの適用可能分野

	要求定義	設計	プログラミング	テスト	運用・保守	プロジェクト管理
解釈				テスト結果の解釈システム	運用負荷状況解釈システム	要員の適正解釈システム
診断				デバッグ支援システム	故障原因診断システム	プロジェクト進捗診断システム
教育	業務知識ベース	設計技法教育システム	言語・ＯＳ・ツール等教育システム			プロジェクト管理シミュレータ
相談	問題分析・業務分析ガイドシスンテム	設計ガイド／ソフトウェア部品使用ガイド	ツール使用ガイドシステム	テスト・ツール使用ガイドシステム	運用ガイドシステム	
設計	要求仕様記述システム	自動プログラミング・システム		テストケース／テストデータ自動生成システム		
制御				自動テスト・システム	自動運用システム	
予測						見積りシステム
計画				テスト計画支援システム	運用計画支援システム	プロジェクト計画システム

このシステムは，設計・プログラミング工程でのコンサルテーション・システムとみることができよう．

PROUST ([22]) Yale 大学の E. Soloway 等によるプログラム理解システムである．学生への Pascal プログラム教育を目的として開発されている．これは，設計・プログラミング工程における教育システムといえる．

UNIX Consultant ([23]) California 大学 Berkley で R. Wilensky を中心に進められているプロジェクトである．システムは，UNIX のコマンドに慣れないユーザが，英語で質問するのに対し，適切なコマンドとその使い方を教えるものである．ソフトウェアの運用工程での，教育システムである．知識ベース・システムとしては，エキスパートの知識の利用という面より，自然言語の処理に力点がある．

CMS ([24]) ソフトウェア開発のプロジェクト管理を，知識ベースに基づくアプローチでシステム化しようとするプロジェクトで，Kestrel Institute において，B. Kedzierski により研究開発が進められた．その後，Kedzierski は Carnegie Group に移り，新たに Crystal というプロジェクト名で，より実際的な場への適用を想定した，ソフトウェア開発の管理支援知識ベース・システムの開発に取り組んでいる．

YES/MVS ([25]) IBM の Watson 研究センタで開発されたエキスパート・システムで，OS のオペレーションを助けるもの．運用コンサルテーション・システムの一例である．

4.4 問題点

このようにソフトウェア関連の知識ベース・システムが成立しうる領域は多様であり，開発の試みも多いが，現状ではまだ実際的なシステムの例はほとんどない．問題点として，以下のようなものが挙げられよう．

知識獲得の難しさ

一般にエキスパート・システム開発のボトルネックは知識獲得部分にあるといわれるが，ソフトウェア開発でとくに新しい技術の求められる要求定義やシステム設計などの分野では，知識獲得がきわめて難しい．ソフトウェアを作成する作業で使われる知識は，その量が膨大かつ複雑で，おそらく単純なルール表現などにはなじまないであろう．

他のツールとの関係

ソフトウェア開発のために，すでに多くのツールが作成され使われてきている．知識ベース・システムも同じソフトウェア・ツールとして，これら既存のツールと整合性を持ち，できれば統合化された環境を提供することが望ましい．しかし，従来のツールと知識ベース型のソフトウェアは性格が異なり，それらの間で整合性を保つことが難しい場合がある．この点に関し，Programmer's Apprentice では，プログラマとともにシステムである徒弟も最新のプログラミング環境についての知識を持ち，ツールを使ったりツールについてアドバイスしたりするという概念が提出されている．

4.5 実現性のある知識ベース・システム

表 1 には，実現可能性のあるソフトウェアの関連の知識ベース・システムを例示した．これらのいくつかは，筆者も関与したソフトウェア開発の実態調査 ([26]) の中で，何人かのソフトウェア技術者により示唆されたものである．一部につき簡単に，説明を加える．

業務知識ベース・システム

アプリケーション・システムの開発には，対象とする業務分野の知識が，ある意味で最も重要である．これを知識ベース化すれば，要求分析・定義や設計の場面で有効なものとなりえよう．

作業標準の順守支援システム

ソフトウェアの開発作業工程は，ほとんどの開発現場でなんらかの形で標準化されている．そのような標準手続きに準拠して作業するようにガイドし，作成された文書やプログラムが基準を満たしているかどうか評価するエキスパート・システムが考えられる．

ソフトウェア部品使用ガイダンス・システム

ソフトウェアの部品化や再利用という考え方は，一部で実行に移されている．問題は部品の検索とその合成であり，それを知識ベースで支援することには，意義が認められる．また，その実現性も，部品化がうまく行われていれば，決して低くはなかろう．

5 AIから生れたソフトウェア技術の活用

初めに注意したように，AI を実現するのもソフトウェアである．したがって，AI の研究の中で工夫されてきたソフトウェア技術は豊富にあり，一般のソフトウェア開発に有効なものも多い．

5.1 知識表現とモデル論

AI で研究されてきたフレームや意味ネットワーク等の知識表現モデルと，ソフトウェア工学で要求分析やシステム設計用の技法として提案されてきた ISDOS, SADT, JSD, 構造化分析，等，およびデータベース設計論で取上げられる ER モデル等のデータ・モデル技法は，それぞれ独立に研究されてきたが，互いに共通する部分がある．このことから，AI の知識表現技法が，ソフトウェアの対象となる問題領域を記述する場合や，それをシステム上に反映させる際の，モデル化の方法を提供するものとして，期待される．

しかし，これは 4.4 であげた問題点と関連し，決してやさしい課題ではない．たとえば A. Borgida 等が，知識表現を要求仕様の記述に用いる試みについて発表しているが ([27])，実用的に意味のある成果がでるの

は，まだこれからであろう．いずれにせよ，知識工学，ソフトウェア工学，データベース技術の三者が互いの成果を意識的に参考にすることは，大いに意味がある．

5.2 種々のプログラミング方法論

プログラミングの主流を続けている手続き型の方法に対し，新しいプログラミングのパラダイムが種々提案されているが，それらの多くはAIに根を発するものである．また，これらのプログラミング方法論に則したプログラミング言語も，たとえば次のようにさまざまなものがある．

- ルール指向：OPS5
- オブジェクト指向：Smalltalk
- 関数型：ML, HOPE, SASL
- 論理型：Prolog
- 混合型：LOOPS, KEE, ART

このような言語も通常のワークステーションやコンピュータで使えるようになりつつあり，その方法論の浸透ともあいまって，今後ソフトウェア工学に大いに影響を与えよう．（これらの動向の一部は，本特集の二木厚吉” 新しい形態のプログラミング” で詳述されている．）

知識とその制御を分離するという知識ベース・システムの考え方自体が，一つのプログラミング方法論を示したものとみることもできる．知識を独立した存在として扱うことで，その追加・修正といった保守の柔軟性が増し，開発の効率も上がるとされる．その仕組みとして，ルール型やオブジェクト型の表現が使われる訳であるが，それだけでソフトウェア危機が解消されるというような期待は，楽観的に過ぎよう．実際，たとえばルールで表現された知識ベースの保守管理は，大変難しい問題であり，下手をすればソフトウェア危機を知識危機に置き換えただけになる危険もある．

要するに，新しいプログラミングに対しても，それにあった開発手法が工夫されねばならないという当然のことに，注意が必要であろう．その際，逆に従来のソフトウェア工学が，知識工学を支援するという構図が成立しよう．

5.3 ソフトウェア開発環境

AIから生れたプログラミング環境としては，古くは時分割システム (TSS) があり，近くはリスプマシンのような高度のインタフェースを持ったワークステーションがある．これらはAIの直接の成果ではないが，コンピュータの研究者や産業界に大きなインパクトを与えた．

これは，ソフトウェアの開発環境における人間と機械のインタフェースの向上に寄与したが，また最終ユーザのためのシステムのインタフェース向上にも応用されてきている．

このようなインタフェースの向上には，さらにAIの直接的な成果も利用されうる．たとえば自然言語処理およびそれに基づく対話処理技術である．すでにあげた例ではUNIX Consultantがあるが，ほかにもデータベースの検索に自然言語を用いることを可能とするシステム例がある．さらに，エディタ，デバッガなどのあらゆるツールをもっと気のきいたものにするのに，AI技術が活用される可能性は高い．

5.4 動く仕様とプロトタイピング

ソフトウェア工学では，80年代に入ってプロトタイピング技法が注目されてきた．その中で，形式的な仕様をそのまま実行することができれば，それをそのままプロトタイプと見なせ，プロトタイプの完成が仕様の決定と一致するから，一石二鳥であるという考え方がでてきている．

実行可能な仕様は，動く仕様または歩く仕様などと呼ばれる．その仕様記述システムとして例に出されるのは，J. Goguen 等の OBJ，S. Gerhart 等の AFFIRM，R. Balzer 等の GIST などである．これらをすべて AI の成果というわけにはいかないが，これらの基盤となっている論理や関数などの形式表現とその評価は，AI の重要な一分野であることは確かである．

プロトタイピングと知識工学は，別の意味で接点を有する．典型的なエキスパート・システムの開発は，シェルと呼ばれる開発ツールを用いる場合には，知識の獲得と整理を行った後，比較的短い期間でプロトタイプを作り，それから時間をかけて知識の改善と拡張を計り，さらにフィールド・テストを十分に行うという，プロトタイピング・アプローチを取るのが普通である．これはまさに，ソフトウェア工学の方法が知識工学に活かされるべき一例といえよう．

6　おわりに

米国のある AI ベンチャー企業が,「知識工学はソフトウェア工学より高度です」という宣伝文句を使っていたが，それを最近「知識工学はソフトウェア工学より高度です，少しだけ」というように変えたという．しかしこれまで見たように，ソフトウェア開発という分野では，知識工学の成果が直接実用的に役立っている例は，今のところきわめて少ない．

しかし，ソフトウェア開発がすぐれて知識集約的な作業であることは確かであり，それだけに知識工学アプローチへの期待は高い．一方，知識工学からみれば，ソフトウェア開発は多様で挑戦的な課題を豊富に提供してくれる．さらに，既に強調したように AI や知識工学を実現する手段がソフトウェアであることを考えると，今後，ソフトウェア工学と知識工学の結びつきが，いよいよ重要性を増すと予想される．

参考文献

[1] Feigenbaum, E.: The Art of Artificial Intelligence—Themes and Case Studies of Knowledge Engineering, 5th IJCAI(1977), pp.1014–1029.

[2] Barr, A. and Feigenbaum, E. eds.: The Handbook of Artificial Intelligence, Pitman, 1981. (翻訳：田中，淵監訳：人工知能ハンドブック，共立出版，1983.)

[3] Waldinger, R. J. and Lee, R. C. T.: PROW: A Step towards Automatic Program Writing, 1st IJCAI(1969), pp.241–252.

[4] Darlington, J. and Burstall, R. M.: A System which Automatically Improves Programs, 3rd IJCAI(1973), pp.479–485.

[5] Darlington, J.: An Experimental Program Transformation and Synthesis System, Artif. Intell., Vol.16(1981), pp.1–46.

[6] Manna, Z. and Waldinger, R.: Synthesis: Dreams→Programs, IEEE Trans. Softw. Eng., SE-5(1979), pp.294–328.

[7] Manna, Z. and Waldinger, R.: A Deductive Approach to Program Synthesis, ACM Trans. Prog. Lang. Syst., Vol.2(1980), pp.90–121.

[8] Bauer, F. and Wossner,H. eds.: Algorithmic Language and Program Development, Springer-Verlag, 1982.

[9] Sato, T. and Tamaki, H.: Transformational Logic Program Synthesis, FGCS-84(1984), Tokyo, pp.195–201.

[10] Balzer, R.: A 15 Year Perspective on Automatic Programming, IEEE Trans. Softw. Eng., SE-11(1985), pp.1257–1268.

[11] Partsch, H. and Steinbruggen, R.: Program Transformation Systems, Comput. Surv., Vol.15(1983), pp.199–236.

[12] Green, C.: The Design of the PSI Program Synthesis System, 2nd IJCAI(1976), pp.4–18.

[13] Smith, D., Kotik, G. and Westfold, S.: Research on Knowledge-Based Software Environments at Kestrel Institute, IEEE Trans. Softw. Eng., SE-11, pp.1278–1295 (1985).
[14] Barstow, D.: An Experiment in Knowledge-Based Automatic Programming, Artif. Intell., Vol.12(1979), pp.73–119.
[15] Barstow, D., Duffey, R., Smoliar, S., and Vestal, S.: An Automatic Programming System to Support an Experimental Science, 6th ICSE(1982), pp.360–366.
[16] Biermann,A.W. and Krishnaswamy, R.: Constructing Programs from Example Computations, IEEE Trans. Softw. Eng., SE-2(1976), pp.141–153.
[17] Michalski, R. S., Carbonell, J. G., and Mitchell, T.M. eds.: Machine Learning, Springer-Verlag, 1984.
[18] Verkshop III—A Formal Verification Workshop, ACM Softw. Eng. Notes, Vol.10(1985), No.4.
[19] Balzer, R., Cheatham, T., and Green, C.: Software Technology in the 1990's: Using a New Paradigm, IEEE Computer, Vol.16(1983), No.11, pp.39–45.
[20] Rich, C. and Shrobe, H.: Initial Report on a Lisp Programmer's Apprentice, IEEE Trans. Softw. Eng.,SE-4(1978), pp.456–467.
[21] Waters, R.: The Programmers's Apprentice:a Session with KBEmacs, IEEE Trans. Softw. Eng., SE-11(1985), pp.1296–1329.
[22] Johnson, W. and Soloway,E.: PROUST: Knowledge-Based Program Understanding, IEEE Trans. Softw. Eng., SE-11(1985), pp.267–275.
[23] Wilensky, R., Arens, Y., Chin, D.: Talking to Unix in English: An Overview of UC, Comm. ACM, Vol.27 (1984), pp.574–593.
[24] Kedzierski, B. K.: Knowledge-Based Project Management and Communication Support in a System Development Environment, Proc. 4th Jerusalem Conference on Information Technology (1984).
[25] Griesmer, J. H. et al.: YES/MVS: A Continuous Real Time Expert System, AAAI-84(1984), pp.130–136.
[26] 情報処理振興事業協会技術センター： ソフトウェア開発に関する実態調査報告書，上方処理振興事業協会 1986 年 4 月).
[27] Borgida, A., Greenspan, S., and Mylopoulos,J.: Knowledge Representation as the Basis for Requirements Specifications, IEEE Computer, Vol.18(1985), No.4, pp.82–91.
[28] ソフトウェアエンジニアリングに関する調査ー知識処理技術の動向とソフトウェア生産への応用，日本電子工業振興協会，1986.