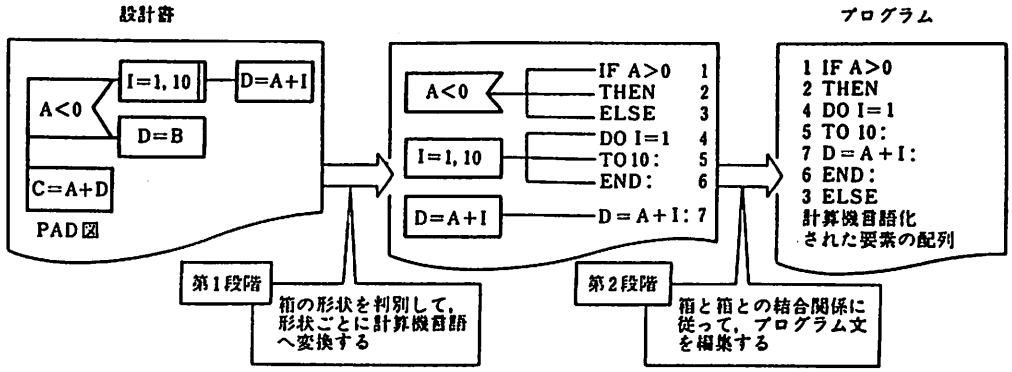

図5・8 変換系（詳細設計書からの変換）

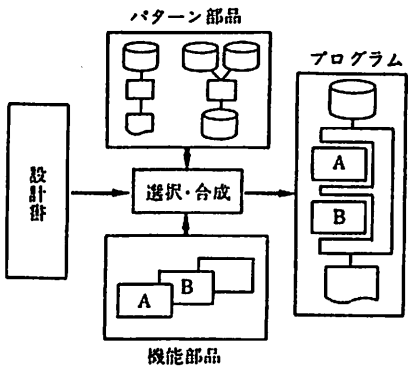

図5・9 変換系（概略仕様からの変換）

変換系には，概略仕様をもとにプログラムを生成する高度な機能を備えたものもある．この場合，生成方式には，部品を用いるものとそうでないものがある．部品を用いないものは，自動プログラミングと呼ばれているが，現時点では実用段階にはない．部品を用いた生成は，実用段階にある．図5・9は，部品を用いた生成系の標準的な構成を示したものである．部品には，個々の機能を実現するプログラム部品と，部品の組合せ方を標準化したパターン部品の2種が準備されている．変換系は，設計書に基づいてプログラム部品とパターン部品を選択し，自動的に組み合わせてプログラムを生成する．この種の変換系では，部品化の方法が課題である．現在，部品化は，経験豊かな設計者が行っているが，設計者の能力によって，部品の質が大きく変化する．部品の質が悪いと生成できるプログラムの範囲は限定される．経験に頼っていた部品化を定式化しようとする試みもなされているが，完成は今後である．

〔4〕 文 書 化 系

画面上で対話設計した図面を文書としてプリンタから出力するのが文書化系である．出力の書式は，あらかじめ定義されたものに従っており，頁付け，索引の作成なども自動的に行われる．

以上，〔1〕～〔4〕の四つの系のほかに，逆変換系や変更解析系を備えた設計CADシステムもある．逆変換系は，上述の変換系と対をなすもので，変換系が設計書からプログラムを生成するのに対し，逆変換系ではプログラムから設計書を生成する．この機能により，既存のソフトウェアの保守に設計CADが適用可能となる．変更解析系は設計書の一部が修正された場合，その波及範囲を解析してレポート出力するものである．変更時の修正もれを防ぐ効果がある．

設計CADシステムへの知識工学の適用は今後の課題である．設計技法がいまだに完成されておらず，実際の設計においては，ノウハウに頼ることが多い．この点で知識工学の適用可能性は高いが，現時点ではノウハウの定式化が困難である．今後，この問題を解決すべく種々のアプローチがなされるであろう．

（前澤裕行）

5・5 知的デバッギングとテスティング

プログラムテスト工程は，ソフトウェア開発費用の約半分を占め，生産性向上に重要であるばかりでなく，品質保証に不可欠の重要工程である．もちろん，高品質ソフトウェアの開発のためには，ソフトウェアの検証を要求定義や設計の各段階でも行うことが基本であ

るが，プログラムの作成を人手に頼る現在の開発方式では，最終的なソフトウェアの検証はプログラムの検証によって行わざるをえない．

しかしながら，コンピュータの処理手順を1ステップずつ正確に記述していくというプログラミングスタイルでは，プログラムの検証技術にも限界がある．単に検証手順の定式化とツールによる自動化を主体とした従来のソフトウェア工学的アプローチの限界を打破するためには知識工学の応用が有効と思われる．

〔1〕 プログラムの基本的な検証方法

プログラムの検証は，「プログラムが仕様どおり作られている」ことを確かめることが目的であり，次のような基本的な方法がある．

(1) 仕様の検証と仕様からのプログラム自動生成
(2) 仕様とプログラムの等価性証明
(3) 仕様から作成したアサーションによるプログラムの正当性証明
(4) テストデータを用いた動的検証

これらの現状の技術レベルに関して，(1)はおもに5・2節で述べている．(2)は(1)よりも難しく，実際的でない．(3)はプログラム実行の前置条件や後置条件あるいはループの不変式などのアサーションの選択が難しく，まだ実用レベルに達していない．現在，実用レベルのプログラムの検証は，もっぱら(4)の方式を用いて行われている．

〔2〕 動的検証の自動化

テストデータを用いた動的検証の自動化は，テスト項目/テストデータの作成支援とテスト，デバッグの作業支援が中心で，次のようなツールが実用になっている[中所 1983]．

(1) 機能仕様からのテスト項目自動生成
(2) プログラムからのテスト項目/テストデータの自動生成
(3) テスト手続き言語を用いた自動テスト
(4) 再テスト時の結果の自動照合

〔3〕 知的なプログラム検証への研究アプローチ

(a) 操作的アプローチによる仕様検証　従来のウォータフォールモデルに基づくソフトウェア開発方式では，仕様の検証が最終工程のシステムテストで行われるため，仕様誤りの修正費用が大きくなるという問題があった．この欠点を除くため，実行可能な仕様記述言語を用いて仕様を記述し，実際にそれを実行させて検証する方法が操作的アプローチである．

代表的なシステムとしては，プロセス間通信とプロセスの状態遷移を基本にしたPAISLey，オブジェクト間の関係と各オブジェクトの属性の記述を基本にしたGISTなどがある．PAISLeyの仕様は対話的に実行され，未定義関数の値をユーザが入力する代わりにデフォルト値を自動設定したり，乱数を用いることもできる．GISTの仕様実行では，実行結果が制約条件を満たしていないときにバックトラックする機能や複数の入力データに対して複数回実行する代わりに記号実行を行う機能がある[佐伯 1987]．

(b) 帰納推論による論理プログラムの検証　帰納推論は，与えられたいくつかの例を満たすような一般的な規則を求めることであるが，この一般的な規則をプログラムと考えると，帰納推論は，例によるプログラム作成と考えられる．

その一例として，まず例から関数を求める方法について述べる．今，求めるべき関数を$f(x)$とし，xおよび$f(x)$の具体的な値の対$(x_i, f(x_i))$の列が次のように与えられたとする．

$$(x_1, f(x_1)), (x_2, f(x_2)), \cdots$$

関数の集合を$F=\{f_1, f_2, \cdots\}$とし，次のような処理をする．

(1) Fからfの候補となる関数f_iを選ぶ．
(2) このf_iが与えられたすべての例を満たすか否かを調べる．
(3) 満たさないときは(1)に戻る．
(4) 満たしたときは，そのf_iをfの候補とする．

このように，関数の集合から順に一つずつ要素を取りだして調べていく方法を枚挙法という．この方式は，簡単であるが，実際には膨大な時間が必要になり，実用的でない．

そこで，上記ステップ(1)において，候補となるf_iを効率よく選ぶ方法として，モデル推論がある．これは，対象を論理プログラムに限定して，その特徴を利用する．今，求めるべき論理プログラムをLとし，このLに対し，trueとなる事実集合$T=\{t_1, t_2, \cdots, t_m\}$とfalseとなる事実集合$F=\{f_1, f_2, \cdots, f_n\}$が例として与えられたとする．節の集合を$C=\{c_1, c_2, \cdots\}$とし，その部分集合$L'$が$L$の候補として与えられたとき，次の処理をする．

(1) ある$t_i \in T$に対し，L'がfalseと判断したと

き，L'に新たにCの要素を一つ加える．

(2) ある$f_i \in F$に対し，L'がtrueと判断したとき，L'からCの要素を一つ除く．

(3) 上記(1)，(2)を繰り返し，すべての$t_i \in T$に対してtrue，すべての$f_i \in F$に対し，falseの判定をするL'が求まったとき，L'をLの候補とする[有川 1987]．

このモデル推論は，原理的には論理プログラムの作成であるが，最初に与えられたプログラムL'の検証とみなせる．このモデル推論は，Prologプログラムのデバッグアルゴリズムに応用できる．

(c) 知識工学アプローチによる検証　従来，テストデバッグツールをはじめとするプログラミングツールは，熟練プログラマに使用され，その作業の効率化を実現するのが主目的であった．ところが，最近では，非熟練プログラマがソフトウェアを開発するケースが急増している．この傾向は，今後のプログラマ不足とともに一層深刻なものになりつつある．しかも，ある調査では，ツールの善し悪しによるプログラムの生産性の差が1.5倍程度なのに対し，個人差（熟練度差）は4倍以上になっている．したがって，この個人差の解消が最大の課題である．

そのためには，熟練プログラマの知識をコンピュータに組み込んで非熟練者に利用させるようなプログラミング用エキスパートシステムが有効であろう．もともと，エキスパートシステムは，診断システムや相談システムが先行して実用化されており，プログラミングツールについても同じことがいえよう．

テストデバッグ用のエキスパートシステムとしてエラーメッセージをもとに周辺の状況から誤りの原因を特定するようなデバッグ支援システムやテストデバッグ機能の効果的な利用方法を誘導したり，テスト項目の選択やテストデータの作成をアドバイスする相談システムなどが考えられる[中所 1986]．

(中所武司)

参考文献

[有川 1987] 有川，石坂：帰納推論による自動プログラミング，情報処理，Vol. 28, No. 10, pp. 1312-1319 (1987)

[佐伯 1987] 佐伯元司：実行可能な仕様記述，情報処理，Vol. 28, No. 8, pp. 1346-1358 (1987)

[垂水 1986] 垂水浩幸，ほか：クラス再利用支援のためのオブジェクトモデル，コンピュータソフトウェア，Vol. 3, No. 3, pp. 61-70 (July 1986)

[千吉良 1987] 千吉良英毅，ほか：ソフトウェア再利用技術の動向，人工知能学会誌，Vol. 2, No. 3, pp. 316-323 (1987)

[辻井 1987] 辻井：知識の表現と利用，昭晃堂(1987)

[長尾 1983] 長尾，淵：論理と意味，岩波情報科学講座(1983)

[中所 1983] 中所武司：ソフトウェアのテスト技法，情報処理，Vol. 24, No. 7, pp. 842-852 (1983)

[中所 1986] 中所，増位：知的プログラミング，計測と制御，Vol. 25, No. 4, pp. 368-373 (1986)

[原田 1984] 原田　実：ソフトウェア生産性向上ツール(上)，日経コンピュータ，Vol. 3, No. 5, pp. 145-160 (1984)；Vol. 3, No. 19, pp. 175-199 (1984)

[淵 1986] 淵，黒川(編)：新世代プログラミング，共立出版(1986)

[Balzer 1978] Balzer, R., et al.: Informality in program specification, IEEE Trans. on Softw. Eng., Vol. SE-4, No. 2, pp. 94-103 (1978)

[Barstow 1987] Barstow, D.: Artificial Intelligence and Software Engineering, Proceedings of 9th International Conference on Software Engineering, March 30-April 2, 1987, Monterey, California, USA, pp. 200-211 (1987)

[Boyle 1984] Boyle, J. M., et al.: Program reusability through program transformation, IEEE Trans. on Softw. Eng., Vol. SE-10, No. 5 (1984)

[Brooks 1975] Brooks, Jr., F. P.: The Mythical man-month, Addison-Wesley (1975)

[Burstall 1977] Burstall, R. and Darlington, J.: A Transformation System for Developing Recursive Programs, JACM, Vol. 24, No. 1, pp. 44-67 (1977)

[Cheng 1984] Cheng, T. T., et al.: Use of very high level languages and program generation by management professionals, IEEE Trans. on Softw. Eng., Vol. SE-10, No. 5, pp. 552-563 (1984)

[Dijkstra 1976] Dijkstra, E. W.: A Discipline of Programming, Prentice-Hall (1976)

[Finzer 1984] Finzer, W. and Gould, L.: Programming By Rehearsal, BYTE June 1984, p. 187における"Dean Brown and Joan Lewis: The Process of Conceptualization, Educational Policy Center Resarch Note EPRC-6747-9. SRI Project 6747. December 1968"の引用

[Frenkel 1985] Frenkel, K. A.: Toward automating the software development cycle, CACM, Vol. 28, No. 6, pp. 578-589, ACM (1985)

[Glib 1985] Glib, T.: Evolutionary delivery versus the "Waterfall Model", ACM SIGSOFT SEN, Vol. 10, No. 3, pp. 49-61, ACM (1985)

[Green 1976] Green, C.: The Design of the PSI program synthesis system, Proc. of 2nd ICSE, pp. 4-18, IEEE (1976)

[Horowitz 1975] Horowitz, E., et al.: Practical Strategies

for Developing Large Software Systems. Addison-Wesly, Reading Mass (1975)

[Kant 1981] Kant, E., et al.: The Refinement Paradigm; The interaction of cording and efficiency knowledge in program synthesis, IEEE Trans. on Softw. Eng., Vol. SE-7, No. 5 (1981)

[Kay 1977] Kay, A. C.: Microelectronics and the Personal Computer, Scientific American, Vol. 237, No. 3, p. 244 (Sept. 1977)

[Kowalski 1979] Kowalski, R. A.: Logic for Problem Solving, North-Holland (1979)

[Licklider 1968] Licklider, J. C. R., Taylor, R. and Herbert, E.: The Computer as a Communication Device, International Sciense and Technology (Sciense & Technology for the Technical Men in Management) p. 31 (April 1968)

[Manna 1974] Manna, Z.: Mathematical Theory of Computation, McGraw-Hill (1974)

[Neighbors 1984] Neighbors, J. M.: The Draco approach to costracting software from reusable components, IEEE Trans. on Softw. Eng., Vol. SE-10, No. 5, pp. 564-574 (1984)

[Prywes 1977] Prywes, N. S.: Automatic Generation of Computer Programs, in Advance in Computers, Vol. 16, Rubinoff, M., et al., pp. 57-125, Academic Press (1977)

[Redwine 1985] Redwine, Jr., S. T. and Riddle, W. E.: Software Technology Maturation, Proceedings of 8th International Conference on Software Engineering, Aug. 28-30, 1985, London, UK, pp. 189-200 (1985)

[Rich 1978] Rich, C., et al.: Initialreport on a LISP programmer's apprentice, IEEE Trans. on Softw. Eng., Vol. SE-4, No. 6, pp. 456-467 (1978)

[Rich 1981] Rich, C.: A formal representation for plans in the programmer's apprentice, Proc. of IJCAI-7, Vancouver (1981)

[Shanker 1980] Shanker, K. S.: Data structures, Types, and Abstractions, IEEE Compu., Vol. 13, No. 4, pp. 67-77 (1980)

[Smith 1985] Smith, D. R., et al.: Research on knowledge-based software environments at Kestrel Institute, IEEE Trans. of Softw. Eng., Vol. SE - 11, No. 11, pp. 1278 - 1295 (1985)

[Smith 1986] Smith, R. G., Dinitz, R. and Barth, P.: IMPULSE-86, A Substrate for Object-Oriented Interface Design, OOPSLA '86, Proceedings pp. 167-176 (Sept. 1986)

[Wasserman 1985] Wasserman, A. I., et al.: Extending state transition diagrams for the specification of human-computer interaction, IEEE Trans. on Softw. Eng., Vol. SE-11, No. 8, pp. 699-713 (1985)

[Waters 1978] Waters, R. C.: The Programmer's Apprentice; A Session with KBEmacs, IEEE Trans. on Softw. Eng., Vol. SE-11, No. 11, pp. 1296-1320 (1978)