最近のソフトウェア技術の展望と将来†

クラーク・ワイスマン††

1. はしがき

ソフトウェア技術で我々が現在注目して研究している課題は，データ・マネジメント分野での今日的な最新システムの作成と未来システムの検討である．おそらく，将来，システムにおけるハードウェアとソフトウェアの区別は解消され，トータル・システム設計の方向に向うものと考えられる．その場合のデータ構造は，ハードウェア及びソフトウェア双方を含む単一のものと考えられ，設計目的は，機種に対してではなく，システムを使用する人間に便利なものに向けられねばならない．我々が現在研究しているものに，英語による鍵盤入力，音声または手書き文字等の入力によるコンピュータとの対話がある．現在では，グラフィック・ディスプレイに文字型を入力すれば，同じ画面から出力が得られるものになっている．コンピュータとの対話には4段階のレベルが考えられる．第1レベルは簡単で，グラフィック装置とミニ・コンピュータとの連絡・制御の手段，第2レベルは，コンピュータに文字を認識させる問題で，手書き文字型を文字として判別させることである．第3レベルは，文字の一部として考えられる数式等を，数学的文脈としてコンピュータに認識させる問題である．これらの文字は文脈に従い意味が変化するので，上添字 (superscript)，下添字 (subscript)，積分・極限及び他の数式内容を演算させる場合，その相違をコンピュータに認識させることである．第4レベルの問題として，数学的意味をコンパイラ言語に翻訳し，かたわら多数のプログラム言語を統合，編集して実行させる問題があり，更に，これらコンパイルしたプログラムでその解答を直ちに出力することである．

† 情報処理月例会特別講演（昭和48年3月9日）
†† Clark Weissman, Chief Technologist, System Development Corp.

2. TAM

我々が先年，NASA（アメリカ航空宇宙局）向に開発した対話グラフィック・システムである TAM (The Assisstant Mathematician) について概要を述べてみよう．コンピュータに数字を認識させるには，グラフィック・タブレット上に数字を書き，この数字をライト・ペンで丸で囲めばよい．これにより，簡単に手書き文字をコンピュータに認識させることができる．アルファベットについても同様である．また，コンピュータに流れ図を認識させることもでき，円，ボックス，矢印の基本形があり，ボックスに2本の矢印をつければ判断ボックスになる．それらを消去したい場合にはライト・ペンで消したいものを抹消すればよい．積分，関数を含むすべての数式計算ができ，その答はグラフィック・ディスプレイ上に出力される．これを使えば，自由に手書きの式をコンピュータに入力できるので，もはや FORTRAN は必要なくなるものと考えられる (Fig. 1).

3. 新概念のコンピュータ

我々は，現在，新しい概念で設計したコンピュータを使い，データ・マネジメント・システムを研究している．これはハードウェアとソフトウェアの将来の方向を示すものと考えられ，経営情報システムの諸問題を解決してくれるものと期待している．

3.1 連想処理

我々は，リアルタイム・マネジメントに応用される連想処理（アソシアティブ・プロセッシング）の価値について研究を進めてきたが，以下はその概要である．

順次処理機械（シリアル・プロセッサ）と連想処理機械とを比較してみると，前者はビット毎にパラレル，ワード毎にシリアルであり，ワードのロケーションに対してアドレスできる機構であるが，後者はビット毎にシリアル，ワード毎にパラレルであり，ワード

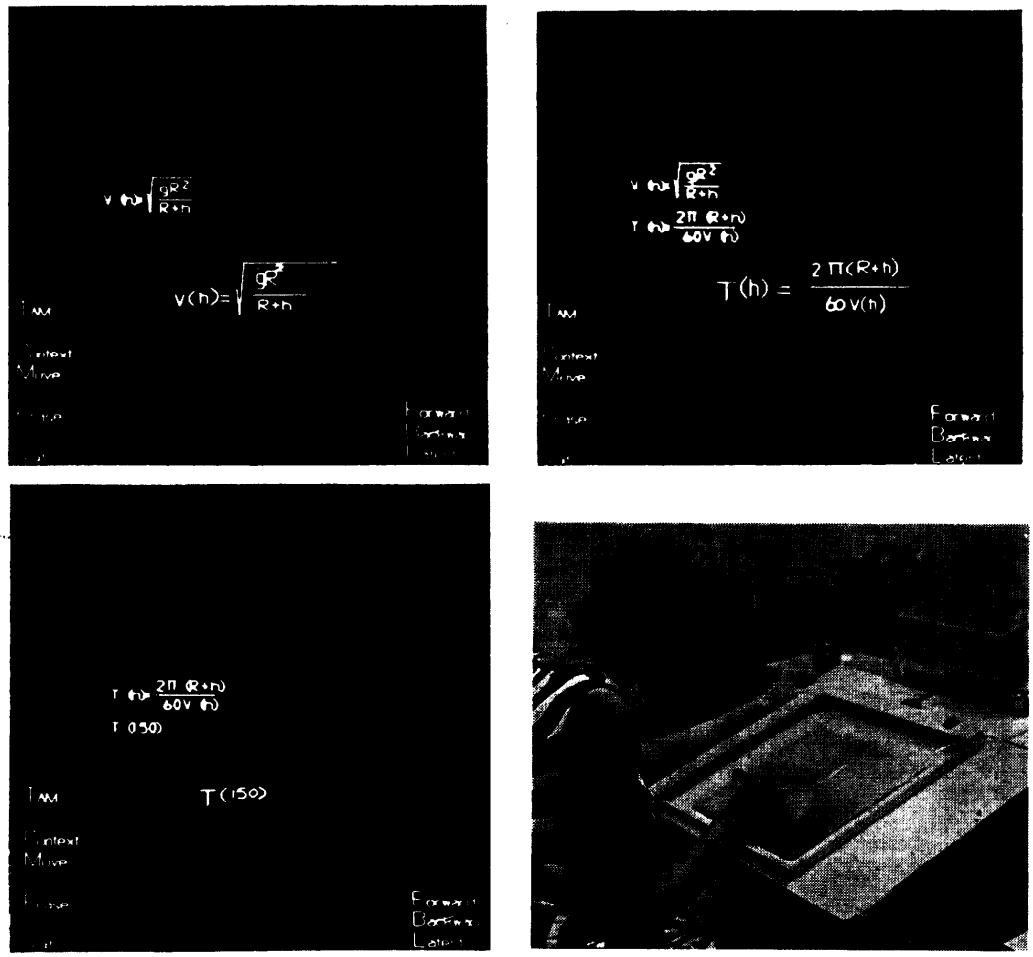

Fig. 1 TAM (The Assistant Mathematician)

の内容に対してアドレスできるようになっている．前者は通常，単一の演算装置をもつが，後者は各メモリ・ワードに対して，同時に演算，論理演算が可能である．このため，連想処理装置でパラレルに多くの演算を行えるようにさせるには，その補助手段として多くの I/O チャンネルが必要になる．

連想処理装置の論理構造について調べてみると，そのメモリは数多くのフィールドで構成され，マスク・レジスタを使えば，マスクされていないフィールドが調べられる．レスポンス・ビットは，それに対応するメモリ・コンペランドが検索条件に合えば，セットされる．このように，連想処理装置のメモリは，ロケーションではなく，内容によりサーチされる．我々はこの装置を経営情報システムの性能向上のためにテスト研究してきた．我々が関係したデータ・マネジメントのアプリケーションには，1) 多量のデータ，2) 少量の計算，3) レコードの迅速なアクセス，4) レポート作成用コンピュータ・タイムが小さいことの4項目が必要であった．その特長として，プログラム・メモリ，データ・メモリ，2次記憶装置，従来の周辺装置及び各種通信装置を組み合せ，中央装置には，I/O プロセッサがつけられ，連想機能をつかさどるコントロール・メモリと同時に，順次処理装置も装備された．

2台の連想処理装置は，シリアル・データ・パスで処理でき，データ転送用に512ビットパラレルに処理する．各装置とも，ワード当り256ビットで，2k ワードのデータ・メモリをもつ．データ取扱の命令語には，情報をパラレル・ブロック毎に移すものと，シリアル・ブロックに移すものとあり，読出し，書込み命令，各種フィールドのシフト命令はもちろん，それに

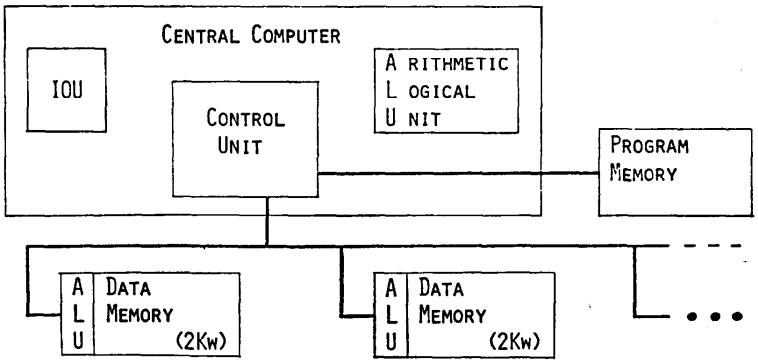

Fig. 2 Associative Processors System

加えて，検索開始命令，連想サーチ用タグ・ビットのテスト命令，各種演算・論理演算命令，検査，判断命令等がある (Fig. 2).

この研究は，データ・ベースにして 12 万件以上，1日約2,000件の項目をオンラインで取扱う必要のある軍事システムで行なったが，応答時間は 1/10 秒以内とした．データ・ベースは185の非階層データ・フィールドからなる 55 のファイルで構成され，検索には 50 個のキーを使用した．いま，この連想データ・ベースと従来のデータ・ベースとを比較してみよう．データ蓄積・検索用にはこれまで使用されてきた数多くの技法があるが，ここでは，インバーテッド及びスレッド・リスト構造の辞書検索型構造技法 (dictionary look up schemes structure technique) をとりあげ，プログラマのコーディングのしやすさ，データ記憶構造の各種要件，検索プログラムの大きさ，I/O 装置及び検索処理中の全計算時間等について比較した．このテストでは，今日のデータ・マネジメント・システムでの最も性能条件のよいバイナリ・サーチを選び，一般によく使われている中型電子計算機と連想処理装置とを比較したものであるが，その結果は，連想処理装置に 50 万バイトのパラレル I/O 装置をつけた場合，その検索時間は 7～110 倍速く，従来の中型電子計算機に 1.5M バイトのシリアル I/O 装置をつけると，連想処理装置のコスト効率はあまりよくなかった．さらに我々のテストで判明した注目すべきことは，データ・ベースが大規模階層構造のものであれば，従来のコンピュータの方が優れ，逆に，非階層構造のデータ・ベースで，パラレルに検索できるキーが多数ある場合には連想処理装置の方がよい．パラレル高速チャンネルがこれに装備されれば，更新時間は 15～4,000 倍ほど速くなる．すなわち，階層的なデータ・ベース構造のものであれば，従来の中型電子計算機の方が性能が高いが，現在あるデータ・ベースに更に多くのキーをつけ，パラレル演算がよくできるように並べ替えれば，連想処理装置全体の性能向上が 30 倍，ある場合には 60 倍になりうる．

3.2 情報処理システムの機密保護

近年，アメリカで重要な問題となってきているものに，システムを損傷や誤動作から如何に安全に守り，信頼性を維持するかという問題がある．この問題を法律問題としてではなく，技術的な観点から取上げてみたい．我々が今日，顧客に提供するシステムは，単なるコンピュータでも，ソフトウェアでもなく，1)物理的環境——建物，設備品，メディアおよび消耗品等，2)人間——製造，運営管理，保守担当者，3)通信連絡手段，4)施設管理方針，5)ハードウェア，6)コンピュータ用ソフトウェア等を含めたトータル・システムと考えられる．これまで前半 1)～3) については，よく保全対策がとられてきたといえるが，後半の 4)～6) は，今後解決していかねばならぬ問題領域である．アメリカのプロジェクトでシステム開発に要するマンパワーが数百人・年であり，その OS だけでも数十万の命令語が含まれる場合がある．そのようなシステムの各種要件は，しばしば，はっきり明記されないことが多いので，我々がシステムの保護を行う場合，すでに完成したシステムの極く一部分にしか手を回せないことになり，容易に破壊されやすい．これに対する手掛りは往々にして得られない．その理由として考えられるのは，そのシステムを作る時点から欠陥があり，しかも，システム要素を保護するには不完全な施設管理が行なわれていることである．この問題の最も適切な

Table 1 Secuirty property determination matrix[1)]

Object \ Property	Authority A	Category C	Franchise F
User, u	Given Constant	Given Constant	u
Terminal, t	Given Constant	Given Constant	u_t^i
Job, j	min (A_u, A_t)	$C_u \frown C_t$	u_j^i
File, f	*Existing file* Given Constant	*Existing file* Given Constant	u_f^i
	New file $\max(A_h(e-1), \rho(A_f{}^e))$, $e>0$	*New file* $C_h(e-1) \cup C_f{}^e$, $e>0$	u_j^i

解決法は，そのシステム，各要件，仕様を明確に記述することから始めることにある．我々が行なったアプローチで最も成功したものにアクセス管理モデルを使用する方法がある．このモデルは ADEPT-50 というタイム・シェアリング・システム設計のベースとなっており，機密保持の対象を定式化し，それらを数学的に取扱うものである．システム記述で明確にされた重要な対象は，利用者，端末，ジョブ及びファイルである．ADEPT-50 では，これらが使用権限（オーソリティ），種類（カテゴリー），専有権（フランチャイズ）という三重鎖のキーないしはロックによってアクセス管理されている．このようなモデル化によって，我々はロックの作動情況が把握でき，また三重のキーによって私有権（プライバシー）の侵害が未然に防止されるように適切な安全管理を行なっている．システムの保護で残された大きな問題は，ハードウェアとソフトウェアを如何にしたら守れるかという問題であるが，現在これについては研究中であり，新しいシステム製造技術の開発過程で解決がみられるであろう．

4. ソフトウェア・エンジニアリングと生産性

数百のプログラムで数十万行にもなる大規模システムの製造技術の面では，かなり遅れており，しかも非常に高価なものになっている．あるシステムの例で，最近 10 年間の生産性を前の 10 年間と比較してみると，ハードウェアの価格性能の面では，約 50 倍にも上昇している．ソフトウェアの面でみれば，1960 年代の一般法則として 40—20—40 の比率が支配的であった．すなわち，ソフトウェア製造費の 40% がシステム分析・設計に，20% はプログラムの作成，残り 40% がシステム検査という割合であった．更に興味深い点は，過去 10 年間の基礎研究は主に，その 20% を解決するためのコンパイラ及び言語の開発に注がれてきたことである．ある大規模プロジェクトでの典型的プログラマの平均的生産性を見てみると，プログラマはアセンブラ言語或は高次言語のどちらかで，年間約 1 万行をプログラミングしている．理由は，はっきりしないが，人間のプログラミング作成能力はその使用言語にあまり関係ない．そのため，アセンブラ言語を使えば生産性は落ち，FORTRAN 或は COBOL 等の高次言語を使えば，生産性は上る．ある人間工学の学者の調査によれば，その原因は人間が鉛筆を使い手で書くことそのものに起因しているということである．我々が 10 万行にのぼるコードを含む複雑なシステムを設計する場合，500 人・月のマンパワーが必要で，1 人・月約 2,000 ドルかかるとすれば，全体で 100 万ドルの費用がかかる．別の調査によれば，多数の命令語の翻訳には，時間当り約 3,000 ドルで，2,000 時間必要であり全体の経費は 160 万ドルかかる．すなわち，1 命令当り約 16 ドルかかることになる．過去 10 年間の生産性を試算してみると，1962 年にアセンブラ言語の命令をプログラム化したと同じ労力で，1972 年

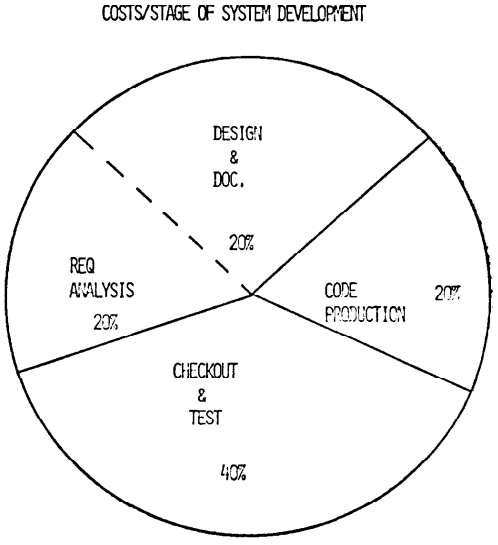

Fig. 3 General Rule of 40-20-40 in the 1960 s'

には高次言語を使い，4個のアセンブラ命令を作成できる．そのコード作成に1962年で1人・月当りCドルかかったとすれば，現在ではインフレのため，その1.7倍余計にかかり，1962年には高次言語によるプログラムはきわめて少なかったが，1972年には全システムの約80%が高次言語でプログラミングされたものと思われる．1962年の生産性を$1/C$とすれば，1972年の生産性は簡単に計算でき，式は次の通りである．

$$P_{62\sim72}=\frac{1\times0.2}{(\text{アセンブラ言語})}+\frac{4\times0.8}{(\text{高次言語})}\Big/1.7\times C$$

試算して驚いた点は，過去10年間のソフトウェア生産性は数字の上で2倍しか上昇していないのに反し，ハードウェアの生産性は約50倍近くも上昇していることである．この著しい違いのため，システムの価格に占めるハードウェアの割合が減少し，逆にソフトウェアの割合が上昇している．この点から，今後はソフトウェア経費を削減して，システム全体の経費を減らす方向に向うものと思われる．次に，将来の生産性を試算してみよう．1982年に，プログラム・モジュールに基くプログラミングができるようになれば，それぞれが約100行からなる高次言語プログラム・ライブラリが期待でき，100行のプログラムからは，少くも400行のアセンブラ言語に相当するものが得られる．今後10年間の労働経費は，同じ割合で上昇するとすれば，2倍になるものと思われ，ソフトウェア全経費の50%はモジュラ・ライブラリに，残り40%が高次言語，10%がタイミングやハードウェアの特別の機能のためにアセンブラで組まれる．いろいろ組合せて予測してみると，1982年の生産性は，1962年のそれの50倍～60倍，1972年の30倍近いものになる．この数値を達成させるためには，ソフトウェア，ハードウェア双方の人間がこの問題を深く検討してみることが大切である．何故なら，上に述べたモジュールはソフトウェア・ライブラリである必要はなく，特定の目的の関数計算といったハードウェア・ライブラリによって構成されることがあるからである．この問題に対するアプローチには，いろいろ考えられ，SDCが実施した方法はストラクチャード・プログラミングに関する形式的プログラミング理論である．このプログラミングにはプログラムの形式論的理論と形式論的管理技法の2つが関係する．この理論は，「プログラム・ブロック構造」，「if-then-else構造」及び「do-while, do-loop構造」という3つの基本部分を使用して展開される (Fig. 4)．

イタリアの学者Corrad Bohmと Giuseppe Jacopini[2]両氏は，この3つの構造がストラクチャード・プログラミングに必要十分な条件であることを証明している．重要な点は，その3つの構造には一つの入力と出力があり，そのいずれもが明瞭かつ正確に規定され，何ら波及効果はないことである．これはAND, OR, NOTの回路構成とよく似ている．このためシステムを設計する場合，プログラムを更に小さなプログラムに分解したり，その逆に，小さなものから大きなものにプログラムを組合せて行くことができる．これには，システムの上部から下部に見て，上から下に，プログラムして行く必要がある．

最後に，与えられたプログラムの正確さを表す数学の展開が必要であり，研究領域としては注目される分

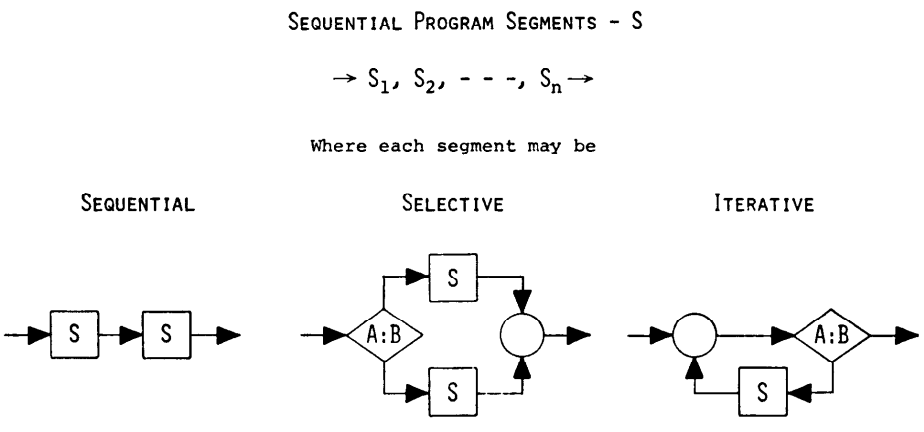

Fig. 4 Structured program design

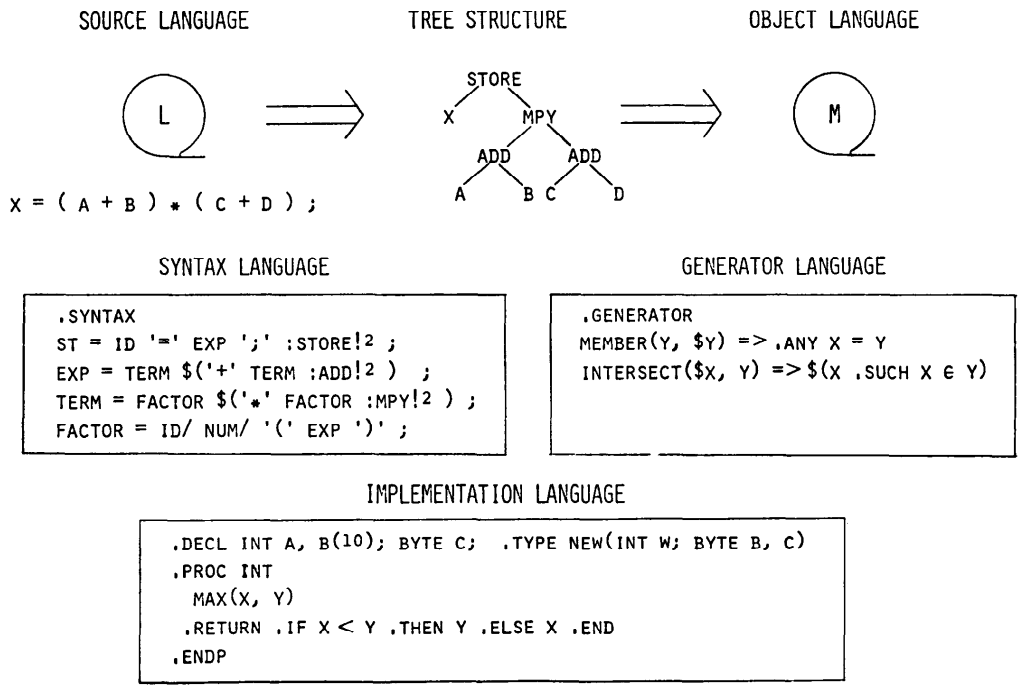

Fig. 5 CWIC schema

野である．その結果は，プログラムを数学的に証明できるものでなければならない．その方向の技術の例を挙げてみよう．E. W. Dijkstra[3][4]は小規模のタイム・シェアリング用OSを開発するのにこれらの技術を応用した．B. H. Liskov[5]は彼の設計したOSに類似の技術を用い，C. W. Roseは，完全なグラフィックI/Oが設備されている未来のコンピュータ用の「logos」データ・マネジメント・システムの中で，この方法の応用を発表している．F. T. Baker[6]はニューヨーク・タイムス情報バンク開発の際この技法を使い成功した．私は，以前に「LISP」に関する仕事で，この技法を具体的に使用した．あるシステムでは，83,000ステップかかり，そのうち77,000ステップを高次言語でプログラム化した．この技法を使用した結果，生産性は2～3倍上昇し，しかもコーディング・エラーはほとんどなくなった．このシステムは，全マンパワーが100人・月を越える大規模なものであった．最近3～5年の間に，Dijkstra, H. D. Mills[7][8], London, King, B. Wegbreit等の学者によって，その手法がきわめて有効なことが証明され，産業界に大きな影響を及ぼしつつある．また，Strachey, Scott, Perlisやカーネギー・メロン大学のグループ等によっても，研究が行なわれており，大規模システムの形式的仕様やその内容についての言語が開発されつつある．いずれ，ソフトウェアの自動作成が1982年までには開始されるであろう．

5. CWIC

ストラクチャード・プログラミング手法を用いて最近SDCが開発したツールにCWIC (Compiler for Writing and Implementing Compilers) がある．このCWICには多くの適用例があり，CWICそれ自身の製造だけではなく，FORTRANコンパイラにおける最適化，JOVIALあるいは宇宙プログラミング言語向のクロス・コンパイラ等の製造に有効である．PROGRESSというPL/Iより大型の航空管制システム向リアル・タイム・プログラミング言語の開発にも応用された．シンボリック・データ・ベースの変換用トランスレータの製造，ミニ・コンピュータ用クロス・コンパイラの作成にも利用できる．各種コンピュータ用コンパイラの製造だけではなく，数種の計算機に同一言語のコンパイラを製造する場合にも有効である．コンパイラは，チェック済のモジュラ・ライブラリを利用して作られ，また，システムそのものがモジュー

ル構成になっている．このため，プログラムをチェックする時間を，最適化作業に振りむけられるため，能率のよいオブジェクト・プログラムを作るコンパイラを経済的に開発できる．また，チェック済のモジュールでシステムが構成されるため，そのシステムの信頼性も著しく高くなる．CWIC のようなシステムの出現により，ソフトウェア・エンジニアリングが確立され，モジュラ・プログラミングの開発が促進されれば，更により生産性の高いソフトウェアが誕生するであろう．

6. むすび

私の研究している問題を中心にして述べてきたが，今後，我々のソフトウェア技術はますます広く実用化されて行くものと信じている．

参考文献

1) C. Weissman: Security Controls in the ADEPT-50 Time-Sharing System, FJCC, 1969, pp. 119–133.
2) C. Bohm and G. Jacopini: Flow Diagrams and Languages with Only Two Formulation Rules, *Communications of the ACM*, May 1966.
3) E. W. Dijkstra: Notes on Structured Programming, Technische Hogeschool, Eindhoven, The Netherlands, August 1969.
4) E. W. Dijkstra: The Structure of "THE" Multiprogramming System, CACM 11 (1968), 341–346.
5) B. H. Liskov: Guidelines for the Design and Implementation of Reliable Software Systems, The MITRE Corporation, MTR 2345, Bedford, Mass., 14 April 1972.
6) F. T. Baker: Chief Programming Team Management of Production, *IBM System Journal*, No. 1, 1972.
7) H. D. Mills: Mathematical Foundations for Structured Programming, Federal Systems Division, IBM, FSC 72-6012, Gaithersburg, Maryland, February 1972.
8) H. D. Mills: Top down programming in large systems, Debugging techniques in large systems, Prentice-Hall.

(翻訳・日本エス・ディー・シー株式会社)