U.D.C. 681.32.06.002

ソフトウェア生産技術の展望

Perspectives of Software Engineering Technology

前澤裕行* *Hiroyuki Maezawa*
葉木洋一** *Yôichi Hagi*
津田道夫*** *Michio Tsuda*
印東　功*** *Isao Indô*

ソフトウェア生産技術は，約20年前に生まれた若い技術であるが，その変遷は急速である。高級言語とフローチャートを道具としていた1960年代の後半に，良いプログラムを開発するための技法として開発された構造化プログラミングがその出発点である。以来，プログラミング工程だけでなく，計画，設計，テスト，保守を加えた開発の全工程を対象として生産技術を確立しようとするソフトウェアエンジニアリングの誕生を経て，現在では工程ごとの開発支援ツールの開発と，それらの統合化，図形表記を用いたビジュアル化開発技法，分散形開発支援方式などの新しい技術が開発されている。本論文では，このような生産技術の動向を展望し，日立製作所での技術開発事例としてSEWBについて紹介する。

1 緒　　言

マイクロコンピュータの普及や大形計算機，オフィスプロセッサの高性能化によって新しい計算機適用分野が生まれている。オフィスをはじめ，工場，店舗，家庭でのオートメーション化が急激に進みつつある。これに伴い，ソフトウェアの大規模化，多様化が生じている。大形機のソフトウェアでは数千キロステップのものが珍らしくない。マイクロコンピュータソフトウェアは，機器制御や事務処理用のものと多岐にわたり，大きなものは数百キロステップのサイズとなっている。またネットワークによるコンピュータ間の結合も進み，ソフトウェアはますます複雑となっている。このような状況の下で，ソフトウェア技術者の深刻な不足が生じている。

過去，ソフトウェア開発は，経験が豊かで高度な技術を身につけた専門家が行ってきた。急激なソフトウェア需要の拡大は，高度技術を備えた専門家だけを集めて開発チームを構成することを困難にした。このため現在では，経験があまり豊かでない初・中級の技術者を支援する生産技術が必要となっている。

本論文では，構造化プログラミング技法から出発した生産技術の発展過程と，開発支援ツールの統合化，図形表現を用いたソフトウェア構造のビジュアル化開発方式，プロトタイピング技法，ワークステーションを用いた分散形開発方式など新しい技術の動向を展望するとともに，これらの技術の日立製作所での事例としてSEWB[1](Software Engineering Workbench)について紹介する。

2 ソフトウェア生産技術の発展過程

ソフトウェアの生産技術は，おおむね1960年代のプログラミング技法時代，1970年代半ばからのソフトウェアエンジニアリング時代の2期に大別できる。

2.1 プログラミング技法時代

2.1.1 高級言語とフローチャート

コンピュータが誕生してしばらくの間，ソフトウェアは機械語やアセンブラ言語を用いて開発されていた。1950年代の後半になると，ソフトウェアを人間の言葉に近づけたCOBOLやFORTRANなど高級言語の開発が行われた。以後約10年間，これらの高級言語と設計用のフローチャート記法がソフトウェア技術者の利用する主な道具であった。

この時代は高度な技術を身につけた名人的なプログラマが尊敬を受けた。彼らは小さなメモリ空間上で高速に動作する複雑な構造のソフトウェアを作りあげる人々であった。一人一人が個性的なプログラムを作り，それを模倣することは容易ではなかった。

やがて，コンピュータの大形化，高性能化に伴い，適用分野が拡大し，ソフトウェア開発量が増加するとプログラマの不足が生じた。高い能力を持つプログラマと不慣れな初心者が協同で開発に当たらざるを得なくなった。精密機械のようなソフトウェアでは，初心者では組み立てることも保守することも極めて困難であった。

ここでソフトウェア構造の良しあしについての考え方に大きな変革が生じた。これまでは，巧妙で精ち(緻)な構造を持つソフトウェアが良いとされていたのが，だれにでも扱える簡潔な構造を持つソフトウェアが良いとされるようになった。この価値観の転換に対応して種々のプログラミング技法が提案された。

* 日立製作所システム開発研究所　** 日立製作所ソフトウェア工場　*** 日立製作所大森ソフトウェア工場

2.1.2 プログラミング技法

1960年代の後半から，構造化プログラミング技法，ワーニエ法，ジャクソン法などのプログラミング技法が開発された。

最も著名な技法は，Dijkstraの構造化プログラミング技法であろう。この技法は，ソフトウェアの構造を複雑にする“GOTO”命令を用いることなくソフトウェア構造を設計する方法として，連接，反復，分岐だけを用いることを定め(**図1**)，これが可能であることを数学的に証明した。この技法は，ほかに段階的詳細化をはじめとする種々の概念を含んだ実用的な技法として普及し，現在世界中で幅広く活用されている。この技法を高級言語に取り込んだPASCALが開発され，FORTRAN，COBOLなどの言語も構造化プログラミング用に改造された。問題分析図(PAD：Problem Analysis Diagram)をはじめとするフローチャートに代わる構造化プログラミング用の設計図式も開発され，対話設計のためのツールも充実してきている[2),3)]。

2.2 ソフトウェアエンジニアリング時代

1973年，Böehmはその論文[4)]の中で，ソフトウェアの不良件数とそれが発生した工程，修正に要するコストを分析した。その結果，プログラミング工程で発生した不良よりも，それ以前のシステム設計，モジュール設計などで発生した不良のほうが修正コストが高いことを示した。これを契機として，それまでのプログラミング工程に焦点を当てた技法だけでなく，計画から設計，プログラミング，テスト，保守に至るソフトウェア開発工程全体を対象とした生産技術の開発が行われるようになり，ソフトウェアエンジニアリングと呼ばれるようになった。

ソフトウェアエンジニアリングの基本となった概念は，ウォータフォール モデルと構造化技法である。

(1) ウォータフォール モデル

このモデルは，上から下に並べた器の間を水が流れ落ちる様子にたとえて，ソフトウェア開発工程全体をモデル化したものである。器に相当するのは，計画，設計などの小工程であり，水に相当するのは仕様書やプログラムである。開発作業を小工程に区切り，各工程での開発結果をドキュメントとしてまとめて次の工程に渡すこと(フェイズド アプローチ)を基本とする。

(2) 構造化技法

ウォータフォール モデルに基づき，各工程を効率よく進めるための多くの技法が提案されたが，これらの間には共通した概念があった。すなわち「構造化」である。「構造化」の基本は分割と統治である。「構造化」は，大きくて複雑な対象を小さな要素に分割し，要素間の関係を定義することによって対象全体を取り扱う手法である。この手法に基づき，構造化分析[4)]，構造化設計，系統的テスト技法などが開発された。これらの技法は，仕様の記述形式と設計手順で必ずしも厳密ではないが，反面，だれにでも使えることが評価され，現在，実用面で主流となっている。

連　接	反　復	選　択
処理1 処理2	処理 条件 F T	T 条件 F 処理1　処理2

注：略語説明　F(Fault)，T(True)

図1　構造化プログラミング技法　連接，反復，分岐の三つのパターンの制御構造を組み合わせることによって，GOTO命令のないプログラムを設計する。

3 ソフトウェア生産技術における最近の動向

ソフトウェアエンジニアリングは，既に開発された技法を，より実用的な方向に改良する方向と，新しい技法を開発する二つの方向に展開している。以下では，この二つの方向について展望し，本号での特集テーマとなっているSEWBでの具体化例を示す。

3.1 シームレスシステム化

ソフトウェア開発工程の機械化は，計画から設計を経て保守に至る全体を，一貫して支援することが不可欠である。一方，個々の工程については，そこで行われる様々な開発活動を総合的に支援するツール群が必要である。例えば，プログラミング工程では，従来あるエディタ，コンパイラ，デバッガに加えてレビュー用ツール，ライブラリ管理ツール，変更管理ツールが必要となる。これらのツール群を総称してプログラミング環境と呼ぶ。

ソフトウェア開発支援システムは，この「一貫」と「環境」の二側面から複数ツールを準備した統合化システムへと進化しつつある。この統合化を実現するシステムは，開発作業全体を支援するツール群を備え，しかもツール間が密接に結合していることからシームレスシステム(Seamless System)と呼ばれる。

図2に示す大形コンピュータを用いたシステム開発支援ソフトウェア(EAGLE2：Effective Approach to Achieving High Level Software Productivity 2)や，**図3**に示すワークステーションを用いたSEWBはシームレスシステムの実現例である。この二つのシステムは機能が互いに補完するように構築されており，組み合わせて活用することが可能であり，システム間のシームレス化も実現している。

ツールのシームレス化に当たっては，仕様・プログラム情報の一元管理，ツール間のユーザーインタフェースの統一が必要となる。従来，個別のツールを1箇所に集め，データの授受方式を定めて接続するツールボックス形統合方式が主流であった。これに代わる新しい方式として，SEWBではツール横断的に対話管理，データ管理などの共通機能を独立なソフトウェアとし，この上で各種のツールを搭載するカーネル

対象＼工程	システム設計	プログラム設計・開発	テスト	運用・管理
EAGLE2 ホスト用ソフトウェア開発支援	システムフロー自動生成	パターン・部品によるプログラム生成	COBOL/SEWB接続プログラム	データ辞書管理
	画面・帳票定義			
	データベース定義	XDM開発言語	COBOL 85 テスト機能	自動ドキュメント支援
	ファイル・レコード定義			
	バッチ帳票定義	バッチ帳票作成支援言語	テストファイル作成	プログラム作成進捗管理
	トランザクション仕様定義*	標準パターン・部品	APテスト支援	プログラム変更履歴管理
		金融用パターン・部品	JCL自動生成	
	SDF/システムフロー管理			稼動情報管理
	EAGLE2 SEWB伝送プログラム			

＊　金融・証券ユーザー向け

注：略語説明　EAGLE2(Effective Approach to Achieving High Level Software Productivity 2)
SDF(Structured Data Flow Diagram)
SEWB(Software Engineering Workbench)
XDM(Extensible Data Manager)
JCL(Job Control Language)

図2　システム開発支援ソフトウェア“EAGLE2”のツール群　　システム設計から運用管理に至る一連の開発工程全体を支援するツールが準備されている。これらを組み合わせて利用することにより，効率の良いソフトウェア開発が可能となる。

(核)化構造形統合方式(**図4**)も実現されている。

3.2　ビジュアル化ソフトウェア開発方式

従来ソフトウェア開発を支援するツールで主として用いられた表現形式は，スクリーンエディタに代表されるように文字形式であった。これはツールを利用するための端末が文字を主体としていたことが主な原因となっている。

ワークステーションの普及により，この状況は一変し，文字に加えて図形を用いたツールが数多く開発されている[11]。**図5**はSEWBを用いたシステム設計からテストに至る一連の開発の流れを示したものである。表現は図形式が主であり，ソフトウェアの構造を図の形や配置，線の接続でビジュアル化することによって，ソフトウェアの設計不良の発生防止とレビューの容易化を図っている。

3.3　プロトタイピング技法

プロトタイピング技法は，ウォータフォール モデルへの批判から生まれた。ウォータフォール モデルでは開発の初期段階で仕様を明確に確定し，これを仕様書に記述することが前提となっていた。これに対してプロトタイピング技法では，初期段階での仕様の確定は困難であり，仕様書による設計者間の意思伝達には限界があるとの認識がある。開発の初期段階で実際に眼前で動く形でシステムの利用者に仕様を提示し，レビューを容易化するのがこの技法である。**図6**にプロトタイピングを支援するツールの一例を示す。このツールを用いることにより，対話システムの画面レイアウトと画面の遷移条件を定義することによって，あたかも実際のソフトウェアが動作しているかのように画面の動きをシミュレートすることが可能である。

3.4　分散形開発支援システム

1980年代の後半に入ってワークステーションが急激に普及しつつある。16ビットCPUから32ビットCPUへと移行しつつあるワークステーションは，10年ほど前の大・中形コンピュータに匹敵する性能を備えている。また，複数のワークステーションをローカルエリアネットワークで接続することによって，より高度な機能を実現できるようになった。

このようなハードウェアの発達に伴い，ソフトウェアの開発環境も，従来の大形コンピュータを端末から利用する方式からワークステーション主体の分散形方式へと転換しつつある。

3.5　知識工学の応用

知識工学はソフトウェア生産技術の舞台に登場して間もない段階であり，基本技術の開発が進められている。日立製作所では，知識工学を適用したプログラム再利用方式の基本技術を開発した。

この再利用方式では，ソフトウェアの仕様書をあらかじめ，ライブラリに蓄積し，必要に応じて検索し，反復利用する。

知識工学は仕様書の検索に適用する。利用者はまず，ソフトウェアの機能に対する要求を日本文で入力する。この日本文をもとに仕様書を検索するためのキーワードが推論により

対象＼工程	システム設計	プログラム設計・開発	テスト	運用・管理
SEWB 共通		PADエディタ		
		ソースエディタ		
	SEWB基本部			
	SEWBユーティリティ			
WS・マイクロコンピュータソフトウェア開発支援		PAD↔C自動生成	CPADテスタ	
ホスト用ソフトウェア開発支援	業務設計支援	PAD↔COBOL自動生成	COBOL PADテスタ	
	システムフロー設計支援			
	画面遷移シミュレータ	PAD↔PL/I自動生成	COBOLソース実行テスタ	
	画面・帳票定義			
		デシジョンテーブル→COBOL自動生成		
		PAD↔FORTRAN自動生成		

注：略語説明 PAD(Problem Analysis Diagram), WS(Workstation)

図3 SEWB(Software Engineering Workbench)の構成 ワークステーション上でシステム設計からテスト，保守までを支援する。各工程に対して複数のツールが用意されている。図2のEAGLEと組み合わせることによってより効果が上がる。

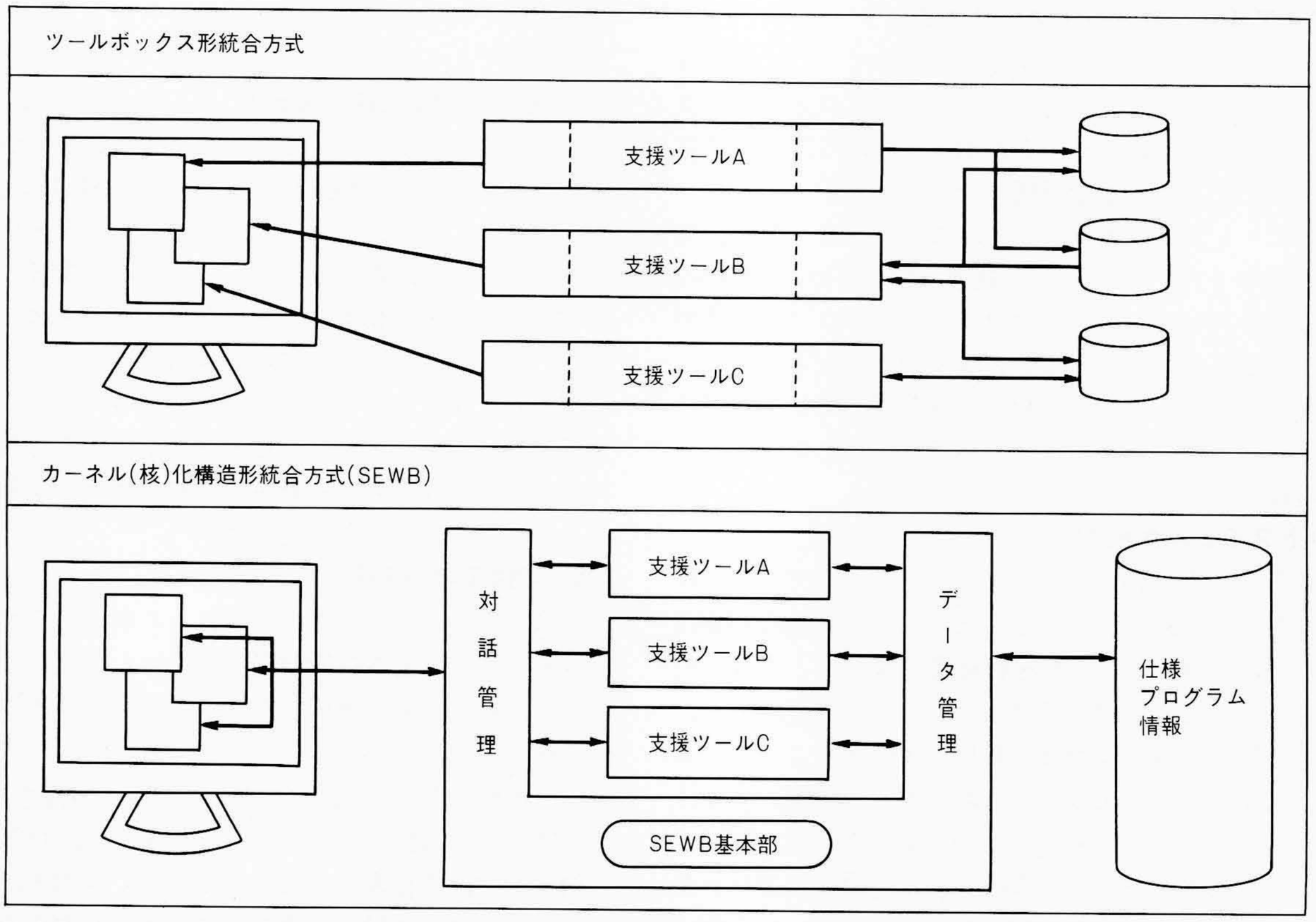


図4 カーネル(核)化構造形統合方式(SEWBの例) SEWBでは，対話管理，データ管理を一括して行う基本部の上に，ツールが各種搭載される統合化構造となっている。

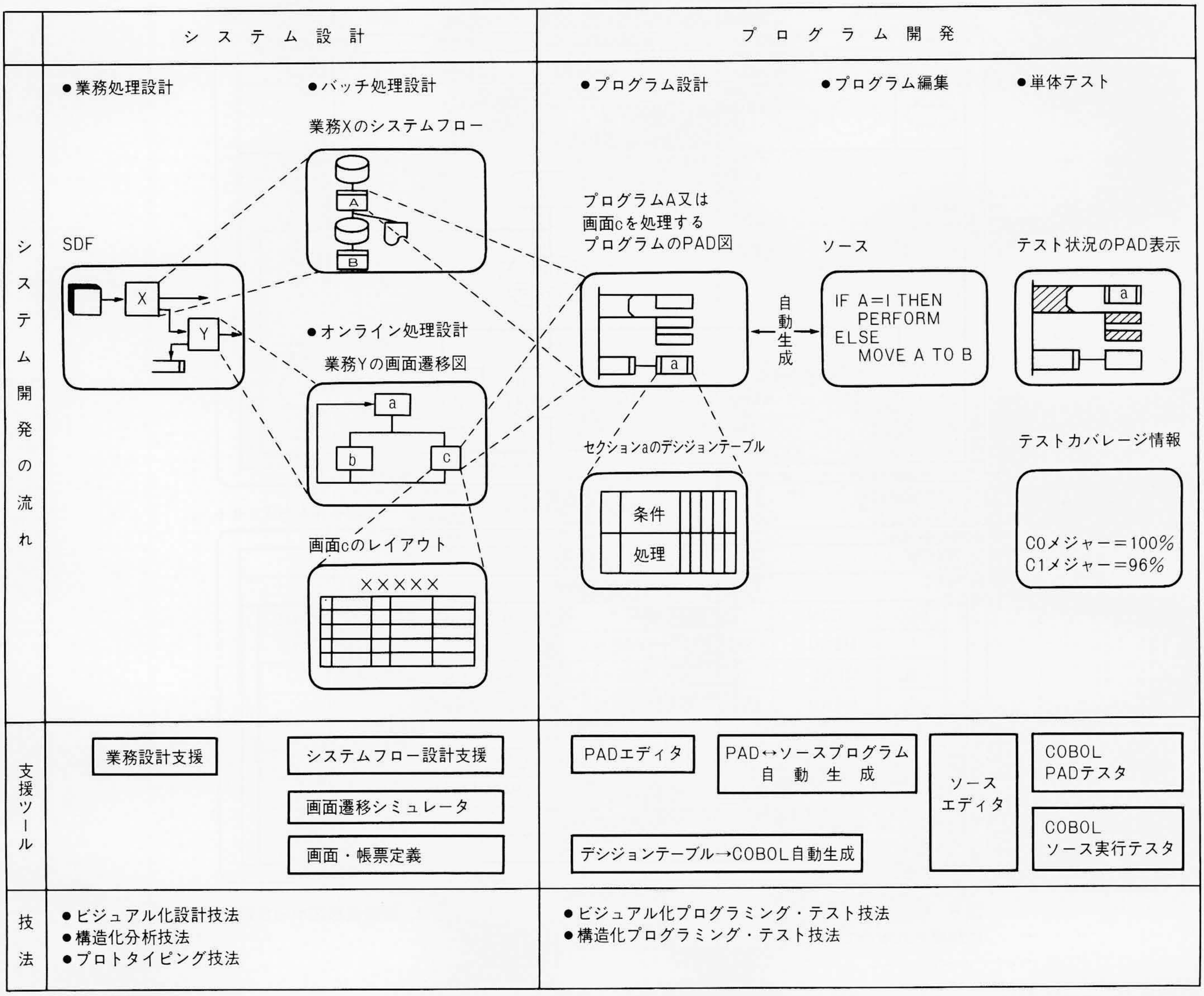


図5　ビジュアル化ソフトウェア開発方式(SEWBの例)　図形を用いてソフトウェアの静的構造を記述し，設計での頭の整理を助けるとともに，レビューを容易とする。図形上のシミュレーションによって動特性の検証も可能である。

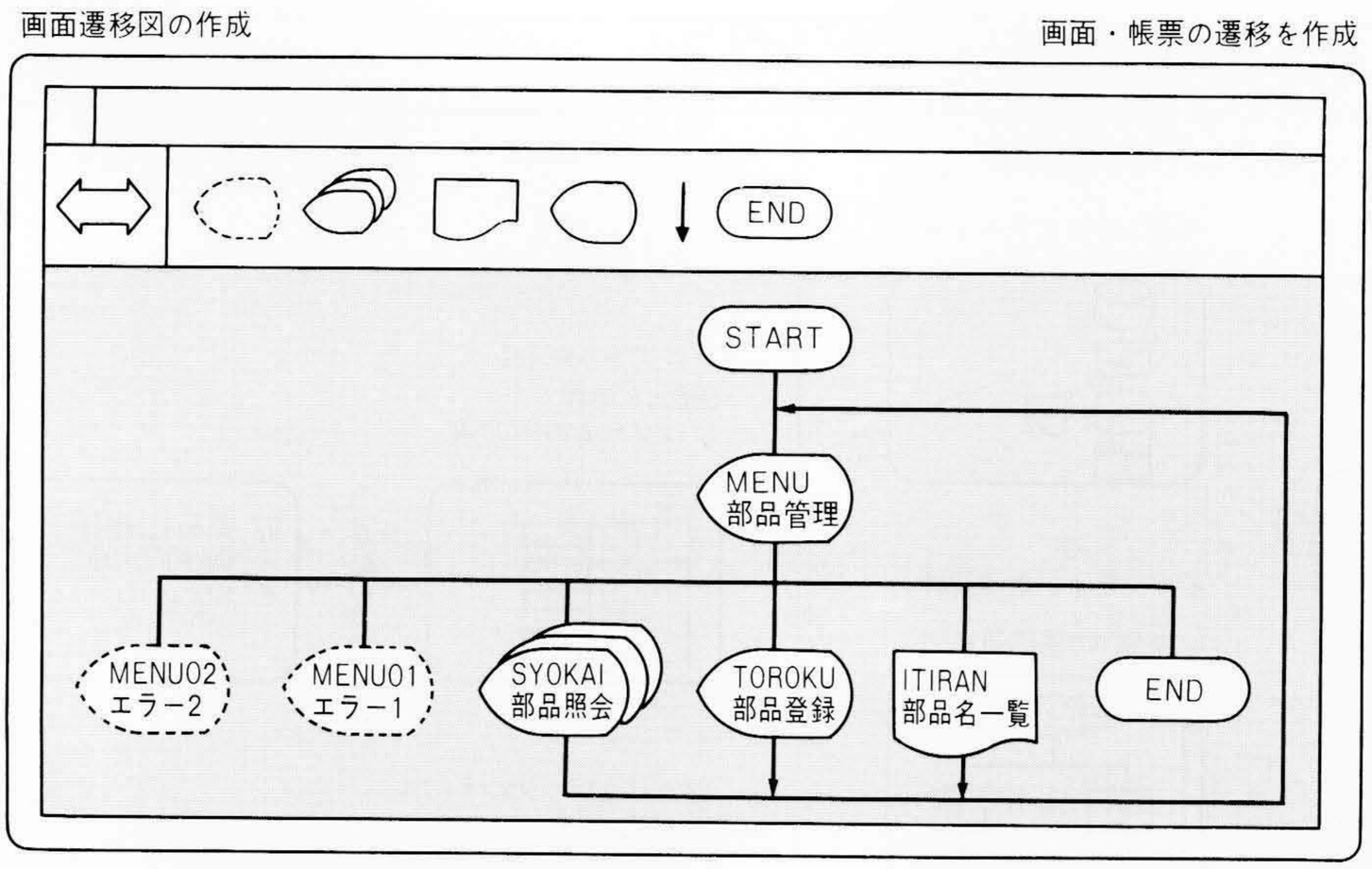
自動抽出される。もとの日本文には，ソフトウェア作成上必要のない語も入っている。このような語は，業務知識ベースと照合してフィルタにかけて削除する。要求に沿って，この機能なら，こういうデータや処理も必要なはずという推論によってキーワードが追加される。得られたキーワードをもとに類似な仕様書が検索され，これを修正することにより新しいソフトウェア仕様書が完成する。仕様書からプログラムへの変換は，上述のSEWB及びEAGLEを用いて対話的にプログラムの組み合わせを指定することにより，自動的に行われる[18]。

4　結　　言

以上でソフトウェア生産技術の歴史と最近の動向の概略を説明したが，紙面の都合で割愛した中にも重要な技術動向は幾つかある。既存のソフトウェアを再構成して再利用する技術[14]，あるいは従来の手続形言語とは異なるオブジェクト指向言語[16]などである。これらの技術は新たにソフトウェアエンジニアリングの舞台に登場して間もない段階であり，現在はその評価が定まっていないが，今後のソフトウェア生産技術の方向を示唆するものである。いずれにせよソフトウェア生産技術の分野では，一部の専門家だけが使いこなせる技術でなく，非専門家が使っても効果のある技術の開発を一歩ずつ積み重ねていくことが肝要である。

参考文献

1)　末宗：“SEWB”の開発思想と機能，日立評論，**70**，2，101～108(昭63-2)
2)　原田　編：構造エディタ，共立出版(1987)
3)　二村：プログラム技法—PADによる構造化プログラミング，オーム社(1984)



遷移条件の定義　　遷移条件を完成させる

SEQ	FROM	TO	遷　移　条　件	TIME
1	MENU	SYOKAI	CODE＝1	1
2	MENU	TOROKU	CODE＝2 & NO！＝SPACE & NAME！＝SPACE	
3	MENU	ITIRAN	CODE＝3	
4	MENU	END	PF10	
5	MENU	MENU01	CODE！＝1 & CODE！＝2 & CODE！＝3	
6	MENU	MENU02	CODE＝2 & (NO＝SPACE！NAME＝SPACE)	
7	SYOKAI	SYOKAI	PF11, PF12	
8	SYOKAI	MENU	ALL	

図6　画面遷移シミュレータ　　遷移条件表のFROM，TOの内容は，画面遷移図から自動的に作成されるので遷移条件だけを定義すればよい。

4) B.W.Böehm：Software and Its Impact：A Quantitative Assesment, Datamation, Vol.19, pp.48～59(1973)
5) T.De Marco：Structured Analysis and System Specification, Yourdon Press (1978)
6) M.Kobayashi et al.：ICAS：An Integrated Computer Aided Software Engineering System, Proc.Compcon'83, Spring(1983)
7) H.L.Hausen, et al.：Software Engineering Environments：State of Art, Problems and Perspectives, Proc. Compsac'82, pp.326～355(1982)
8) H.Maezawa, et al.：Interactive System for Structured Program Production, Proc.7th ICSE, pp.162～171(1984)
9) T.Chucho, et al.：HITS：A Symbolic Testing and Debugging System for Multilingual Microcomputer Software, Proc.AFIPS, pp.73～80(1983)
10) Z.Furukawa, et al.：AGENT An Advanced Test-case Generation System for Functional Testing, Proc.AFIPS (1985)
11) 葉木，外：システム開発支援ソフトウェア"EAGLE"，日立評論，**68**，5，374～378(昭61-5)
12) S.K.Chang：Visual Languages：A Tutorial and Survey, IEEE Software, Vol.4，No.1，pp.29～39(1987)
13) 中村：分析から保守まで一貫支援，機能拡大する開発ツール，日経コンピュータ，No.161(1987)
14) W.Tracz：Reusability Come of Age, IEEE Software, Vol. 4，No.4，pp.9～12(1987)
15) D.Barstow：Artificial Intelligence and Software Engineering, Proc. 9th ICSE, pp.200～211(1987)
16) B.J.Cox：Message/Object Programming：An Evolutionary Change in Programming Technology, IEEE Software, Vol.1，No.1，pp.50～61(1984)
17) "Seamless System"(特集号)，IEEE Computer, Vol.20, No. 11(1987-11)
18) 千吉良，外：システム仕様書の再利用によるソフトウェアの開発技法(ICAS-REUSE)，日立評論，**69**，3，pp.249～254(昭62-3)