CURRENT TRENDS IN SOFTWARE ENGINEERING

(ソフトウェア生産技術の動向)

宮本勲 (日本電気(株) 公共システム事業部)

1. ソフトウェア生産技術とは?

今、ソフトウェア・エンジニアリングという言葉が流行している。これは、「ソフトウェア工学」もしくは「ソフトウェア生産技術」という2つの意味で使われていることが多い。ソフトウェア工学は(研究者の立場から?)、ソフトウェアに関する各種の科学的概念(もしあれば)を明確に抽出し、工学的に厳密な取扱いができる様にしようという意向を示している。一方、ソフトウェア生産技術は(生産者の立場から)、ソフトウェアに関する科学的知識をもとに、ソフトウェア生産をより工業的にしていこうというものである。
つまり、ソフトウェア生産技術とは、

> "The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them." [1]

である。

ソフトウェア生産技術には、どの様な分野があり、どの様なことが行なわれているのか? 世の大勢に従い、かつ、私見(独断と偏見?)を混じえながら、これらについて簡単にながめることにする。

2. ソフトウェアのライフサイクル

ソフトウェアの生産に関する技術をより良く理解するには、まずソフトウェアの生産手順を知ることが必要である。そして、ライフサイクルの順序に従って、技術動向を観ていくのが現実的であろう。
ソフトウェアのライフサイクルは、各所各様のバリエーションがあると思われるが、大略、図1に示す如くであろう。

以下、ソフトウェアのライフサイクルの順序で話しを進める。

3. システム要求定義技術

データ処理システムに対する要求を分析し、ハードウェア、ソフトウェア、その他のサブシステムへの要求に分解する作業に関する技術で、いわゆるシステム・エンジニアリングである。ソフトウェアの機能や性能をうまく定義するには、まず、最初のシステム・レベルの要求を正しく定義することが鍵となる。このためには、完全性、無矛盾性を保証し、次のレベルへの追随性の記述だけでなく、アブストラクションを含んだシステム記述性などを保証できる様な分割・分解のアプローチが必要である。

この様な点を解決すべく、BMD システムでは次の様な3つのアプローチを試しているが、まだまだ良い結果が得られていなくて、この領域の研究が必要だとしている。

- Verification Graph Method [2]
- Petri Net Method [3]
- Finite State Machine Approach [4]

今のところ、経験的な方法で対処せざるを得ない状況である。

4. ソフトウェア要求定義技術

ソフトウェア・サブシステムに対する要求を分析し、ソフトウェア機能集合として定義づける作業に関連する技術一般を、要求定義技術と呼んでいる。

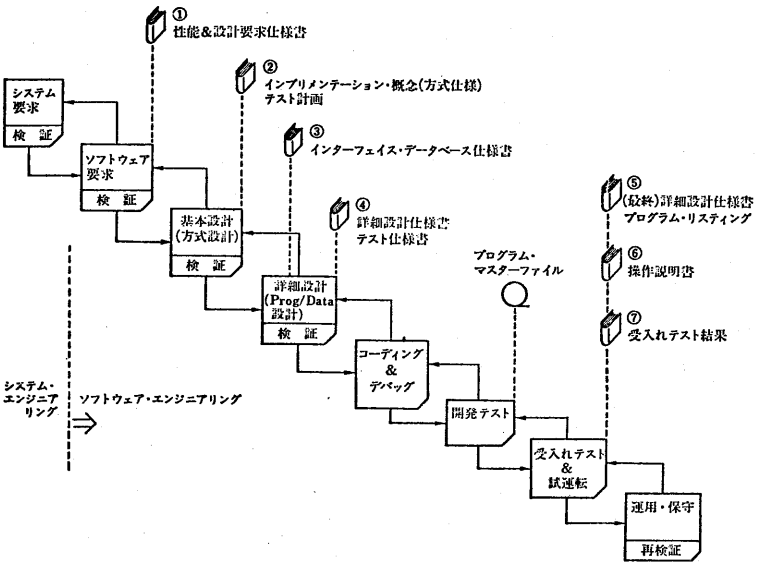

図1. ソフトウェア・ライフサイクル

従来、要求仕様の定義はソフトウェア開発の出発点であるにもかかわらず、あいまいなままでそれ以降の作業工程に進むことが多かった。このため要求内容の矛盾やあいまいさが多く残り、設計のやり直しやプログラムの作り直しをたびたび強いられた。そして結果として開発工程の大きな遅れやコストの増大をもたらしている。そこで、もし、この段階であいまいさと不統一な要求仕様記述をなくすることが出来れば、以後の作業能率と品質の向上につながるという立場から、要求仕様をフォーマルな記述方法で定義しようということが積極的に取り組まれ始めている。また、要求仕様を自動処理しようという動きもある。つまり、要求仕様を定義する言語とその解析プログラムを準備し、定義内容をデータベースで管理しようとするものである。

この例として Michigan 大学の ISDOS [5]、TRW の SREP [6]、Softech の SADT [7]、MIT Draper 研の HOS [8] などのアプローチがある。これらの特徴を表1にまとめておく。

また、要求定義(もしくは問題定義)から自動的に設計仕様やプログラムを生成する自動プログラミングということも考えられ始めている。MIT の Martin [9] や USC の Balzer [10] らのアプローチがある。

5. 設計技術

ソフトウェア設計は要求分析結果の機能集合を最適に実現するアルゴリズムやデータ構成を見いだすことである。方式設計とプログラム/データ設計の2段階がある。

表1 要求定義アプローチの比較

		SREP	ISDOS	HOS	SADT
一般的特徴	・適用すべき開発工程	・Subsystem/System要求定義	・System 定義	・System/Subsystem定義	・System/Subsystem定義
	・最適な応用分野（初目的）	・実時間システム（ミサイル防衛システム）	・事務用システム	・Space 制御システム	・MIS
	・自動化の程度	・高度	・高度	・計画中	・手作業
	・高水準言語	・RSL	・PSL	・AXES	・英語
	・要求定義の機械処理	・可能	・可能	・計画中	・不可能
	・使用状態	・評価中（作成完了）	・運用（多数のユーザ）	・開発中	・運用（多数のプロジェクト）
方法論	・段階的手順（明確なアプローチ）	・有	・無	・無	・有
	・ドキュメンテーション	・自動化，flexible なフォーマット	・自動化，固定フォーマット	・未定	・無
	・管理的な側面	・手順が明確かつ測定可能，支援言語	・支援言語	・未定	・レビュー手続きが明確
	・解析サポート				
	・無矛盾性・完全性チェック	・自動解析	・(解析のための)自動化レポーティング	・未定	・無
	・シミュレーション	・有	・無	・無	・無
要求記述	・フォーマルな構造化メカニズム	・要求ネットワーク (R-Nets)	・機能ツリー／ネットワーク	・プロセス・ツリー	・SADT ストラクチャード・ダイヤグラム
	・機能要求の形式	・論理フロー記述	・機能インタラクション	・コントロール・ストラクチャ	・Data/Activity グラフ
言語的特性	・記述手段				
	・ストラクチャ	・有	・有	・有	・(英語)
	・データ	・有	・有	・有	・(英語)
	・性能要求	・有	・無	・無	・(英語)
	・シミュレーション・モデル	・有	・無	・無	・(英語)
	・traceability	・有	・有	・有	・(英語)

(1) 方式設計

要求仕様をもとに特定のマシン、OSにとらわれずにアルゴリズムやデータ構成を考える段階である。これには2つのアプローチが考えられる。

ⓐ 誤り非許容設計

バグレス設計とも呼ばれ、信頼性を阻害する要因を前もって取り除いておくことにより信頼性を保証するアプローチである。信頼性を阻害する要因を完全に取り除いておくことは実際には不可能なので、それらの要因を許容できる範囲まで減らすことがこのアプローチの現実的な目標である。つまり、設計誤りが入り込みにくい方法、また誤りが発生した場合も取り除き易い様な設計手法である。この様の方法論としては、トップダウン設計、ボトムアップ設計などの階層的設計法があるし、構造化プログラミングも基本的にはこの範ちゅうのものであろう。

ⓑ 誤り許容設計

冗長設計、回復設計、自己診断方式の設計などは誤り許容を目指したものである。これらはシステムのアルゴリズム全体もしくは一部分に冗長性を持たせ、エラーが発生してもその影響が他に波及しない様にしたり、障害を自動的に回復することによって、ソフトウェアの信頼度を向上させる手法である。誤り許容設計の考え方は、ハードウェアの分野では進歩しているが、ソフトウェアの領域ではまだまだである。ハードウェアと合わせて、フォールト・トレラント・コンピューティングとしてこの先もっと盛んになると考えられるが、現段階では、Aviziemis による冗長設計の議論[11]、Parnas による回復設計[12]、Yau による自己診断方式[13]、Randell や Anderson らの回復ブロック[14,15]、Chow によるアサーション技術[16]、Denning や Hecht らによる実時間システムや OS における議論[17]、そして SIFT というSRIの誤り許容システム設計の例[18]などがあるだけである。

誤り許容設計は、性能・コスト面で問題があるが、ソフトウェアの信頼性を上げるためには誤り非許容手法だけでは不十分であるから、両者を相互補足の形で取り入れることが望ましい。

(2) プログラム/データ設計

方式設計の結果を使用対象のハードウェア、OSに載せるために、物理レベルで具体的にプログラムとデータ構成を決定することである。この段階では、モジュール分割技術が中心的なものとして挙げられる。
モジュール化の目的は、プログラムの作成、デバッギングおよび保守を可能な限り容易に行なえる様にすることである。モジュール化の議論としては、Myersによる複合設計[19]、Stevensらの構造化設計[20]、Parnasのモジュール分割法[21]、などがある。また、Edwardsによるモジュール化とテスト容易性との関連の議論[22]もある。
また、最近、定義の唯一性ということも指摘されている[23]。

設計技法もしくはその支援ツールを表2に挙げる。電化定義に使用することができるものも若干、含まれている。

表2. 各種方法論

アプローチ		TAG	ADS	SODA	LOGOS	CSC THREADS	MODEL DRIVEN	SOP	Engagement LOGIC	ARDI	AUTA SIM
適用システム	・シミュレーション		L		H	L	M			M	H
	・リアルタイム・システム	L					M		H	L	
	・MIS	H	H			H	L				M
	・事務用システム	H	H	H		H	M	H		H	
	・データ変換	L	L			H	M			M	
	・テレコミュニケーション	M	M			M	M			L	H
システム規模	・大規模 (>100k)	L				L	H	L	H	L	
	・中規模 (10~100k)	H	H	H	H	H	H	H	H	H	M
	・小規模 (<10k)	H	H	H	H	H	H	H	H	H	H
特徴	・システム設計への支援	M	M	H	H	M	H	H	H	H	M
	・設計シミュレーション				H						
	・シミュレータ自動生成										H
	・要求の伝達	H	H				M	M	H	M	
	・体系的システム設計		H					M		M	
	・要求分析への支援	M					H	H		H	
	・discipline	M	M					M		M	
	・モジュラリティ										H
	・(入力-処理-出力)記述性	H	H			H				M	

H : High Orientation　M : Medium Orientation　L : Low Orientation

(3) 設計記述

設計内容やプログラム仕様記述は設計者の意図が簡潔・明瞭に表現でき、読む者に正確に伝えられなくてはいけない。従来、設計記述は自然言語やブロックチャート、フローチャートの様な図表現を用いて行なわれてきたが、記述言語のもつ"あいまいさ"とか記述する人によって記述の仕方が異なるなどの問題があった。

現在までに考え出されている設計記述法はおよそ表3、表4に挙げられているものであろう。夫々特徴があるし、付随の方法論も異なるので、一概に優劣を論ずることは出来ないが、全体にデータ構造の設計に便利なものが少ないようである。また、設計記述言語として形式的なものが考えられつつある。CaineらのPDLシステム[24]やBMDのPDSシステム[25]は、設計記述言語の機械処理を目指したものである。

6. プログラミング技術

詳細設計仕様をもとにプログラミング言語を使って実行形式のプログラムとデータセットを作り上げる作業である。

(1) プログラミング言語

最近、ストラクチャード・プログラミングやアブストラクションの考えを反映した

表3. 設計記述法 (1)

CHARACTERISTIC	GLOBAL REPRESENTATION SCHEME			
	DESIGN TREE/ CONTROL GRAPH	HIPO	SOFTECH	STRUCTURE CHARTS
DATA STRUCTURE SHOWN	NO	NO	NO	NO
DATA FLOW DEPICTED	YES	YES	YES	YES
I/O MEDIA SPECIFIED	NO	NO	YES	YES
IMPLEMENTATION ISSUES ADDRESSED	NO	NO	NO	YES
MATHEMATICAL BASIS	YES	NO	NO	NO
HIERARCHICALLY ORDERED	YES	YES	YES	YES
MODULE ENVIRONMENT DESCRIBED	NO	NO	YES	YES
CONTROL FLOW PORTRAYED	YES	YES	NO	NO
DATA & ACTIONS DEPICTED SEPARATELY	NO	NO	YES	NO
CONTROLLABLE AMOUNT OF DETAIL	YES	YES	NO	YES
DEPICTS MODULARITY	NO	NO	NO	YES
FORMAL STRUCTURING MECHANISM	YES	NO	YES	NO
INTERMODULE RELATIONSHIPS PORTRAYED	NO	NO	NO	YES
SYSTEM ANALYSIS AID	YES	YES	YES	NO

表4. 設計記述法 (2)

CHARACTERISTIC	LOCAL REPRESENTATION SCHEME								
	CONTROL GRAPH/ DESIGN TREE	DECISION TABLE	DILL, HOPSON, & DIXON	FLOW CHART	HAMILTON & ZELDIN	SCAT	NASSI & SHNEIDERMAN	TRANSACTION DIAGRAMS	WEIDERMAN & RAWSON
MATHEMATICAL BASIS	YES	YES	NO	NO	NO	NO	NO	NO	NO
DATAFLOW DOCUMENTED	YES	NO	NO	NO	NO	NO	NO	NO	NO
USEFUL ON LARGE OR SMALL SYSTEMS OR PROGRAMS	YES	YES	YES	YES	YES	YES	NO	NO	YES
DISSIMILAR FROM CODE	YES	YES	YES	YES	YES	YES	NO	NO	YES
SYNTACTICAL (FORMAL) ADHERENCE TO BASIC CONTROL CONSTRUCTS	NO	NO	NO	NO	NO	NO	YES	YES	NO
CAN DESCRIBE AN ARBITRARY PROCESS	YES	YES	YES	YES	YES	YES	NO	NO	YES
DEPICTS EXECUTION DETAILS	YES	YES	YES	YES	YES	YES	YES	YES	YES
UTILIZES VISUAL COMMUNICATION VIA GRAPHICS	YES	NO	YES	YES	YES	YES	YES	YES	YES
DISPLAYS CONTROL STRUCTURE EXPLICITLY	YES	NO	NO	NO	NO	NO	NO	NO	NO
DEPARTURE FROM FLOWCHARTS	YES	YES	NO	N/A	NO	NO	YES	YES	NO
INDUSTRY STANDARD EXISTS	NO	NO	NO	YES	NO	NO	NO	NO	NO
DISPLAYS CONTROL FLOW	YES	YES	YES	YES	YES	YES	YES	YES	YES
DATA STRUCTURE DISPLAYED	NO	NO	NO	NO	NO	NO	NO	NO	NO
CONTROLLABLE LEVEL OF DETAIL	YES	NO	NO	NO	NO	NO	NO	NO	NO
SELF-CONTAINED (DOES NOT REQUIRE TEXT IN ADDITION TO GRAPHICS)	NO	NO	NO	NO	NO	NO	NO	YES	NO

言語が多数、出現している。これらは大雑把に次の様なアプローチがある。

まず第1は Go To-less プログラミングを行なうための制御構造を既存言語の命令でシミュレートして実現しようとするものである。最も手軽で、CODASYL のシンポジウム[26]や、Hough[27]、Galeo[28]、筆者[29]らの報告がある。

第2のアプローチは、基本制御構造の表現をマクロや拡張言語の形で実現しようというものである。この種の試みが、Sullivan[30]、Higgins[31]、Meissner[32]らによって為されている。

3番目の方法は更に積極的に SP の概念を取り入れて新規に言語体系を開発しようというものである。Liskov[33]、Wirth[34]、White[35]、Stewert[36]、下村ら[37]、Wulf[38]などが取り組んでいる。

また、言語機能として、手続きの抽象化に加えて、データの抽象化が表現可能になりつつある。"class"、"cluster"、"process"、"monitor"、"form"、"group" などの概念はこの例である。更に新しい流れを示す言語として、Wasserman の PLAIN[39]、Chow のアサーション言語[16]、DOD の HOL[40]などがある。

(2) プログラミング

プログラム作成は、① その使用者自身が作成する場合や、プログラムのライフタイムが、例えば一度使われるとその使命を終える様な短期的なものの場合、と、② 他人が使用することを想定して作成する場合や何度も繰り返して使われる長期的使命をもつものの場合とで、その作成方法が大きく異なる。後者は前者に比べ、予め色々な使われ方を充分に想定しておく必要があったり、長い使用期間中に生じる仕様の変更などに対する改造や保守上の考慮が重要になる。いずれの場合もプログラムが正しいことは自明の前提条件であるが、後者の場合は、作

られるプログラムの読み易さ、理解のし易さ、保守のし易さなどが問題になる。ソフトウェア生産技術の立場は後者である。

プログラミングでのトピックとしては、GO TO-less コーディング、プログラミング・スタイルの問題[41]、コーディング規約[42]、作業環境(体制、支援ツールなど)の整備による作業能率の改善などがある。

7. テスト技術

ソフトウェアのテストの目的は、コード化されたソフトウェア・システムが定められたとおりに正しく機能することを確認することである。従来、この工程ではテストケースの抽出、大量のテストデータやテストプログラムの作成、テスト実施、テスト結果の確認など人手のかかる作業が多く、大変であった。

プログラムが正しく動作することを確認するには、次の様なアプローチが考えられる。すなわち、

- 選択されたテスト (selective test)
- 徹底的なテスト (exhaustive test)
- 正しさの証明 (correctness proof)

選択されたテストとは、適切に選択された動作条件においてプログラムを検証するものであり、徹底的なテストは、全ての動作条件を実行しようとするテストである。また、正しさの証明はプログラムの正当性を理論的に証明しようとするものである。現実的なものとしては、選択されたテストが唯一で、これの実施上のストラテジイとして、

- トップダウン・テスト手法
- ボトムアップ・テスト手法
- 一斉テスト手法

がある。

これらの違いはインプリメンテーションの状況にも関係するが、一言でいうと、どこからテストに取り掛かるかということである。つまり、上からか、下からか、それとも全体に対し一挙に取り掛かるかである。これらは実際には組み合わせて用いられることが多い。

テストでの主なトピックスとして次の様なものが挙げられる。

- ソースプログラムの静的解析
- プログラムの動的解析
- テストケースの自動選択と生成
- シンボリックなプログラム実行
- プログラムの正当性の証明

そして、これらを助ける道具の開発などが盛んである。

8. 保守技術

保守とは、運用中のソフトウェアをモディファイすることで、これには性能拡張などの更新作業と、ソフトウェアのエラーなどの修理作業の両方が含まれる。

ソフトウェアの更新作業は現行のソフトウェア仕様(性能、効率、操作性など)

に対する変更要求、新しい仕様の追加要求などによって行なわれる。母体となるソフトウェアが既に存在する点が異なるが、通常の開発作業とほぼ同様の手順で作業が行なわれると考えられる。

ソフトウェアの修理作業は、使用中に検出されたエラーの原因を、場合によってはエラー発生状況を再現して見つけ出し、その解決法を考え、実際にソフトウェアを修正し、そして正しく修理されたことを確認するという一連の手順で行なわれる。修理保守は、修正保守、環境変化に対応する保守、より完全にするための保守の3つに分けられる。[43]

修正保守としては次の様な場合が考えられる。

- プログラムの異常終了や不適当な情報を出力するという様な処理上の誤りを修正する。
- 平均応答時間、トランザクションの誤り発生率などの性能上の誤りを修正する。
- プログラミング標準に準拠していないとか、性能仕様と設計内容の不一致などの、ソフトウェアの作り方に関する誤りを修正する。　など

環境変化に対応する保守としては、たとえば

- 分類コードの変更、データベースの論理構造の変更などのデータ環境の変化による修正
- ハードウェアやオペレーティング・システムなどの処理環境の変更にともなう修正、　など。

より完全にするための保守としては次の様な例が考えられる。

- より良いアルゴリズムへの修正や、ハードウェアのより効率的な使用を目的とするような変更など
- より便利なものにという目的で、出力形式の改善とか新しい出力情報の追加などの、性能上の洗練
- コーディング規約は守っているが、それ以上にコメントを充実させるなどして読解性を向上させる。　など.

ソフトウェアの保守段階での外役に立つ特効薬的な技術は余り考えられないので、ソフトウェアの開発段階で予め、保守が行ない易い様に設計・作成しておく以外に根本的な解決法はないと思われる。

9. 管理技術

ソフトウェアの開発・保守に関する管理技術は非常に重要である。管理の問題としては次の様なものがある。まず計画が貧弱であり勝ちなこと。またそのために無駄な労力を費したり、逆にアイドルタイムが生じたりすること。いくらりっぱな計画が立てられたとしても、その遂行・管理がおろそかにされると無意味であること。プロジェクトで必要な資源量の推定がずさんであったり、管理者に適材が得られなかったりすることも多い。また、プロジェクト管理を行なうために最低限必要な情報の収集がなされていないなどである。

これらに対し、資源量や費用の見積り技術[44]や、プロジェクト管理におけるガイドラインなどが考えられつつある。また管理技術とソフトウェア開発技法とを結びつけたアプローチも取られ始めている。管理に必要な情報の自動収集シス

テムなども実験的に開発されつつある様である。[45] また管理支援を目指した開発支援システムとして、Bell研のPWB [46]、SDCのSoftware Factory [47]、Softechの SEF [48] などもある。

10. ソフトウェア品質評価技術

これまでソフトウェア品質というと主に信頼性と効率だと思われてきたが、最近は変化してきている。Boehm [49] によると 図2 の様に階層的に定義される という。この分野では、品質特性の定義とそれらの評価尺度の定量化、そして夫々の特性の評価技術の研究がされている。

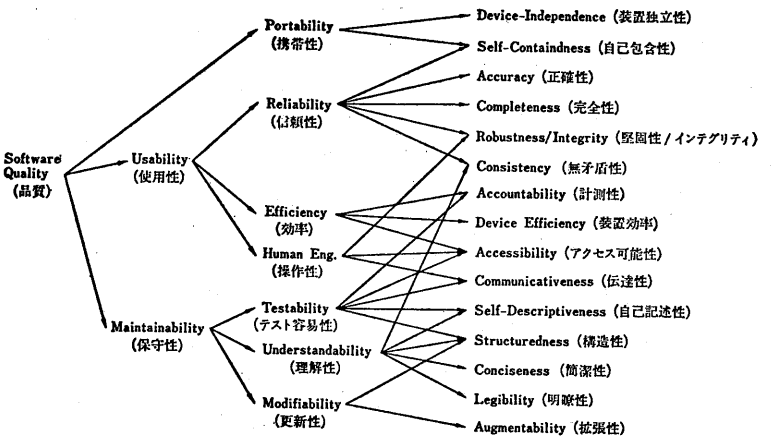

図2. ソフトウェア品質特性

定量的評価では信頼性と効率の評価が盛んである。

信頼性評価では、信頼性データの収集と分析、そしてそれらを基にした信頼度予測モデルの作成などが行なわれている。筆者の報告 [50] では、オンライン・システムのテスト期間中の信頼性データの測定・分析、それらを基にした基礎的な信頼度推定モデルを提案している。また最近、Musa [51] の推定理論を我々風にアレンジしてプログラム化した。

効率評価では、従来から解析的アプローチ、シミュレーション、実測評価などが行なわれている。最近ではシミュレーション技術などがソフトウェア要求定義ツール、設計ツールで用いられ始めている。前述のSREPでは要求定義時にその性能をシミュレートし、効率を検証することが可能である。効率評価はコンピュータのアーキテクチャの評価からネットワークや、データベースの分野にまで応用されている。筆者の報告 [52, 53] は、データベース・システムの効率評価の為の階層的解析モデルについて述べている。また文献 [54] は、データ参照の局所性を扱ったものである。

11. 最後に

以上、ソフトウェア生産技術の輪郭と主な話題を極く簡単に紹介した。いずれの分野も問題が山積みされ、我々の挑戦と努力を待っている。

ここで省略した点については、参考文献や、筆者らが今年4月から連載中の「ソフトウェア・エンジニアリングへの道」(bit 共立出版、全10回の予定) を参照されたい。

〈参考文献〉

1. B.W. Boehm, "Software Engineering", IEEE Tr. Computers, Dec. 1976

2. P. Belford, et al,"Specifications - A key to Effective Software Development", Proc. 2nd Int. Conf. SW Eng. Oct. 1976

3. E. Balkovich & E. Engleberg, "Research Towards a Technology to Support the Specification of Data Processing System Performance Requirements", ibid

4. K. Salter, "A Methodology for Decomposing System Requirements into Data Processing Requirements", ibid

5. D. Teichroew, et al, "PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems", IEEE Tr. SW Eng Vol.SE-3 NO.1, Jan. 1977

6. M.W. Alford, "A requirements engineering methodology for real-time processing requirements", ibid

7. D.T. Ross, "Structured Analysis (SA): A language for communicating ideas", ibid

8. M. Hamilton & S. Zeldin, "Higher Order Software- A Methodology for defining Software", IEEE Tr. SW Eng. March 1976

9. W.A. Martin, et al, "Requirements Derivation in Automatic Programming", Proc. MRI Symp. Comp. SW Eng. April 1976

10. R.M. Balzer, " Imprecise Program Specification", USC-ISI, NO.ISI/RR-75-36, Dec. 1975

11. A. Avizienis, "Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing", Proc. ICRS, April 1975

12. D.L. Parnas & H. Wurges, "Response to undesired events in software systems", ibid

13. S.S. Yau, et al,"An approach to error-resistant software design", Proc. 2nd Int. Conf. SW Eng. Oct. 1976

14. B. Randell, "System structure for software fault-tolerance", Proc. ICRS, April 1975

15. T. Anderson & R. Kerr, "Recovery blocks in action: A system supporting high reliability", Proc. 2nd Int Conf SW Eng, Oct 1976

16. T.S. Chow, "A generalized assertion language", ibid

17. ACM Computing Surveys, Vol.8, NO.4, Dec. 1976

18. J.H. Wensley, et al, "The design, analysis and verification of the SIFT fault-tolerance system", Proc. 2nd Int Conf SW Eng. Oct. 1976

19. G.J. Myers, "Composite design facilities of six programming languages", IBM Sys. J. Vol.15, NO.3, 1976

20. W.P. Stevens, et al, "Structured Design", ibid Vol.13, NO.2 1974

21. D.L. Parnas, "On the criteria to be used in decomposing systems into modules" CACM, Dec. 1972

22. N.P. Edwards, "The effect of certain modular design principles on testability" Proc. ICRS, April 1975

23. P.D. Greim, Jr., "On the principle of unique definition", Proc. NCC AFIPS 1975

24. S.H. Caine & E.K. Gordon, "PDL: A tool for software design", ibid

25. R. Koppang, "Process design system- An integrated set of software development tools", Proc. 2nd Int Conf SW Eng. Oct 1976

26. CODASYL-PLC: Proceedings of a Symposium on Structured Programming in COBOL-Future and Present", April 1975

27. K.A. Hough, "Structured Programming in COBOL", Proc. 2nd US-Japan Conf. Aug. 1975

28. L.E. Gales,"Structured FORTRAN with no Processor", SIGPLAN Notices, ACM Vol.10 NO.10, 1975

29. I. Miyamoto, "Some considerations in database application programming", Proc. 2nd Int Conf SW Eng. Oct 1976

30. J.E. Sullivan, "Extending PL/I for structured programming", Computer Languages Vol.1, 1975

31. D.S. Higgins,"A structured FORTRAN translator", SIGPLAN Notices, Vol.9 NO.4 1974

32. L.P. Meissner, "A compatibles structured extention to FORTRAN", ibid Vol.9, NO.10 1974

33. B.H. Liskov & S. Zilles, "Programming with abstract data types", ibid Vol.9 NO.4 1974

34. N. Wirth, "An assessment of the programming language PASCAL", Proc. ICRS, 1975

35. J.R. White & L. Presser," A structured language for translator construction", Computer Journal, Vol.18 NO.1 1975

36. S.L. Stewert, "STAPLE: An experimental structured programming language", Compu. Languages, Vol.1, 1975

37. T. Shimomura, et al, "SPOT: A structured system development system'', Proc. ACM Annual Conf 1974

38. W.A. Wulf, et al, "BLISS: A language for systems programming", CACM, Vol.14 NO.12, 1971

39. A.I. Wasserman, "PLAIN: Programming language design and systematic programming" IEEE COMPSAC Conf. No. 1977 (to appear)

40. L.C. William & A. Whitaker,"A Defence View of Software Engineering", Proc. 2nd Int Conf SW Eng. Oct 1976

41. B.W. Kernighan & P.J. Plauger,"The elements of programming style", McGraw-Hill 1974

42. H.F. Ledgard, "COBOL under control", CACM Vol.19 NO.11 Nov. 1976

43. E.B. Swanson, "The dimensions of maintenance", Proc. 2nd Int. Conf SW Eng. Oct. 1976

44. R.W. Wolverton, "The cost of developing large scale software", IEEE Tr. Computer Vol.C-23, NO.6, 1974

45. J.A. Clapp & J.E. Sullivan, "Automated monitoring of software quality", Proc. NCC, AFIPS, 1974

46. T.A. Dolotta, et al, "An introduction to the Programmer's Workbench", Proc. 2nd Int Conf SW Eng. Oct 1976

47. H. Bratmn & T. Curt,"Software Factory", Computer, May 1975

48. C.A. Irvine & J.W. Brackett, "Automated software engineering through structured data management", Proc. 2nd Int Conf SW Eng. Oct 1976

49. B.W. Boehm, et al,"Quantitative evaluation of software quality", ibid

50. I. Miyamoto, "Software reliability in on-line real time environment", Proc. ICRS, April 1975

51. J.D. Musa, "A theory of software reliability and its application", IEEE Tr. SW Eng. Vol.SE-1, NO.3, 1975

52. I. Miyamoto, "Performance analysis models for database management system", ACM Computer Science Conference Feb. 1975

53. I. Miyamoto, "Hierarchical performance analysis models for data base systems", Proc. Int Conf Very Large Data Bases, Sept 1975

54. I. Miyamoto, "Data reference characteristics of database application program", Proc. NCC, AFIPS, 1974, p1053 (abstract)
Full paper is presented on Journal of NEC R&D, NO.35, Oct. 1974, pp46-56