解　説

新しいシステム開発手法の適用*

中　井　直　男**

はじめに

ソフトウェア開発上次のような問題がある.

① 生　産　性

近年におけるコンピュータ技術の発展はめざましく,ハードウェア面では記憶容量の拡大,仮想記憶システムあるいは仮想記憶システムの考え方を DASD*** の領域まで拡張した大容量記憶装置の登場，ソフトウェア面では,オペレーティング・システム，DB/DC**** あるいは高級言語の改善などによるシステム能力の向上がはかられてきたが，プログラマー1日当りの生産性は旧態依然としてオンライン・システムで 10 ステップ程度，オフライ・ンプログラムでも十数ステップというのが常識である．一方，人件費の高騰によりコンピュータ経費のうちソフトウェア経費の占める割合が年々上昇しているのが現状であり，プログラミングの生産性向上が第1の問題である.

② 維　持　性

1973 年度版コンピュータ白書によれば，人件費の 56% が維持費で開発費の 44% を上廻っていることは，アプリケーションの要件の変更にともなう修正がいかに多いかを物語っており，変更要求に迅速に対処できることが第2の点である.

③ 拡　張　性

システムに要求される今日の必要条件の1つは，多数の適用業務を容易に追加できることであり，DB/DC システムなどは，この拡張性の問題を解決することを1つの目標としている．ソフトウェア開発に際しては拡張性を考慮したオープン・エンデッドな設計が不可欠となって来ている.

④ 信　頼　性

コンピュータ・システムの利用層が拡大し，企業活動のみならず社会生活にもコンピュータの及ぼす影響が大きくなっている今日では，ハードウェアの信頼性が高いということが条件であるばかりでなく，ソフトウェアの品質をいかに高めるかが重要な問題である．いかに多量のテスト・ケースを使用してテストしたとしても，全ケースをカバーすることは不可能であることを考えれば，ソフトウェアの信頼性向上は難しい命題である.

⑤ 管　理　性

ソフトウェア開発過程における最大の問題は，管理性に乏ぼしいことである．これは，プログラムが手芸的性格をもつことに起因しているが，適用業務の量が多くなればなる程ソフトウェアの開発も家内工業から脱皮することが要求される.

以上のような問題を解決するために 1960 年代の後半からプログラム，ドキュメンテーションあるいはプロジェクト運営面での改善が試みられているが，本稿で紹介する高級技術者を頂点とするチーフ・プログラマー・チームの編成は，ソフトウェア開発における最も進んだプロジェクト運営形態であるといえる．トップ・ダウン・アプローチも高級技術者の存在があって初めて可能であるし，プログラミング・プロダクション・ライブラリーと言う考え方は，専業化することによりチームの運営の効果をあげようとの試みである．またストラクチャード・プログラミングおよび HIPO 手法の採用は如何にしたら高品質のプログラムをプログラマー個人の経験や能力に大きく依存する事なく，効果的に開発出来るかと言う命題に対する解答であり，これによりストラクチャード・ウォークスルーをより効果的にした．これら一連の新しい手法は各々をとってもソフトウェア開発上の問題解決に有効であるが，全てを採り入れた時その威力を最大限に発揮できる．以下個々の手法について概説する.

A. 新しいプログラミング手法

新しいプログラミング手法は,

---

* Apply of Improved Programming Techniques by Sunao NAKAI (IBM Japan Ltd., DP Service)

** 日本アイ・ビー・エム(株) DPサービス

*** Direct Access Storage Device

**** Data Base/Data Communication

① チーフ・プログラマー・チーム
② トップ・ダウン・アプローチ
③ ストラクチャード・プログラミング
④ プログラミング・プロダクション・ライブラリー

から構成されている.

1. チーフ・プログラマー・チーム

チーフ・プログラマー・チームは，チーフ・プログラマー，バックアップ・プログラマーおよびライブラリアンから構成されるチーム・オペレーションにより，作業分担を徹底し，各人の能力を最大限に発揮し得るように組織化された小人数のグループである．チーフ・プログラマー・チームは，プログラム開発作業の管理性，信頼性および生産性の面を本質的に改善することを意図したものであり，後述のストラクチャード・ウォークスルーなどの手法により，プログラムを個人の芸術作品からチーム全員による共同作品とし，プログラム作成過程をチーム内の適切なスキルや責任範囲を反映して，品質を保てるように仕事の流れを組織化することを可能とする．

1) チーフ・プログラマー

チーフ・プログラマーは次のような能力の持主が適当である．

① チーム・メンバーの報告を直接受けるテクニカル・マネージャー．
② 上級の，経験豊かなプログラミングの専門家．
③ 複雑なハードウェアやソフトウェアを効果的に組み合せシステムの性能を十分引き出せる高度な技術を持っている人．

また，チーフ・プログラマーの役割は，

① プログラムの中核となる部分や主要な部分の設計やコーディングをする．
② バックアップ・プログラマーに仕事を割り当てる．
③ バックアップ・プログラマーの作成したプログラムのレビューあるいはそのプログラムのシステムへの組み入れを監視する．

などである．

2) バックアップ・プログラマー

バックアップ・プログラマーは次のような能力を持っていることが要求される．

① 複雑なハードウェアやソフトウェアを十分使いこなせる技術．
② 上級プログラミングの専門家．

また，バックアップ・プログラマーは，

① チーフ・プログラマーの指導の下でシステムの重要な部分の設計やコーディングをする．
② チーフ・プログラマーがシステム開発の中核部分に集中できるようにプログラミング手法や方策の樹立のためにチーフ・プログラマーを援助する．
③ システムのテスト計画を作成する．

などの責任をもっている．

3) ライブラリアン

ライブラリアンの仕事は磁気ディスクやテープ，リスティングなどのライブラリーおよび後述のプログラミング・プロダクション・ライブラリーを維持したり記録をとり管理したりすることである．チーフ・プログラマーや，バックアップ・プログラマーがプログラミング作業に専念できるようにすることが目的でありプログラミング作業以外の全ての作業はライブラリアンの責任である．

2. トップ・ダウン・アプローチ

1) 目　的

トップ・ダウン・アプローチは図-1のごとくシステムを設計する場合，先ず全体を概略設計し，次に詳細あるいは部分を設計するのと同様に，開発段階においてもやはりシステム全体を上位のレベルのモジュールからだんだん下位のモジュールへと作成することであり，ストラクチャード・プログラミングの考え方をトータル・システムに拡張したものである．

トップ・ダウン・アプローチの目的はつぎのとおりである．

① システム設計や構造に関する大きな問題をプログラム開発の初期に解決する．
② 統合テストとそれに関連して発生するデバッギングや新しくコーディングすることを避ける．
③ 信頼性高く，維持性に富んだプログラムを作成することを可能にする．

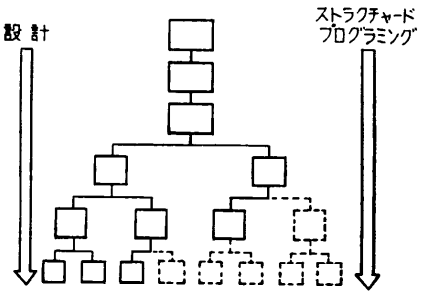


図-1　トップ・ダウン・アプローチ

2) 利 点

トップ・ダウン・アプローチの利点はつぎのような点である.

① インターフェイスをとるべき接点がしぼれるので,システム仕様や設計に対する変更が詳細な部分に及ぶことは少なく,容易にそれを吸収できる.

② 下位との接点(インターフェイス)が少ないので変更に伴うコミュニケーションの負担を軽減できる.この変更に関係する人が少ないということは,コミュニケーションのオーバーヘッドを更に軽減出来ることを意味している.

③ 構造上の問題はプロジェクトの初期の段階に検討され,解決され,テスト確認される.

④ テスト段階の初期において,ドライバー・プログラムを必要としない.

⑤ システム統合テストの段階が不要となり,論理テストを完了すればシステムとして使用できる.

⑥ 基本となるシステムが早期に使用可能となるので,オペレーターの訓練やハードウェアをサポートする場合には有利となる.

3. ストラクチャード・プログラム

1) 目 的

ストラクチャード・プログラムの目的は,つぎのとおりである.

① 読みやすく,理解しやすく,修正しやすいコーディングを可能にする.こうして書かれたプログラムは,プロジェクト・チーム内における技術情報のコミュニケーションを図るツールとしても適している.

② ストラクチャード・プログラミング手法を用いると速やかにコーディングができるばかりでなく,コーディングの完了したものをすぐコンピュータにのせてもかなりの確かさでロジックを遂行することができる.これは,ストラクチャード・プログラミング手法のねらいとしているプログラムの生産性と製品の品質向上を満たすものである.

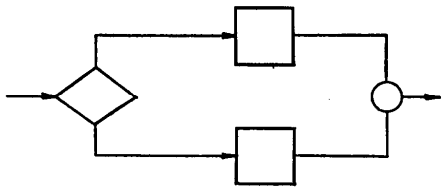
2) 規 則

ストラクチャード・プログラミング作成上守らなければならない規則はつぎの7つである.

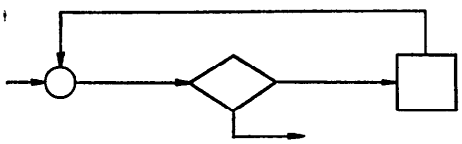
① いかなるプログラムも**図-2**に示した3つの基本型の組み合わせと反復のコンビネーションでロジックを組み立てることができる.

② ストラクチャード・プログラミング手法では,プログラムの明解さを保つため,ブランチ命令やGOTOステートメントの使用を禁じている.したがって,ブランチ先を表わすレーベルもコメントとしての目的以外には不必要である.ストラクチャード・プログラミングは,"飛び出し"や"飛び込み"のないIFとDOステートメントの組み合わせで表わされているので,本を読む時と同じように,上から下へとコーディングを読むことができる.やむを得ずブランチ命令やGOTOステートメントを使用しなければならない場合には,特別に仕様書を作成し,明確に記述する.

(a) 2つのオペレーションの連続(Sequence)



(b) 2つのオペレーションの1つに条件付でブランチし再び結合(IFTHENELSE)



(c) 一定の条件が満足されるまであるオペレーションの繰り返し(DOWHILE)

図-2 基本型

③ 1つのセグメントは,コンピュータからのアウトプット1頁に収まる大きさに制限する.1つのセグメントを1頁以内に収めるように制限することは,経験的にみて,プログラマーが1時点で容易に読み,理解できるロジックの大きさであるからである.

④ 1つのセグメントは,1つのインプットと1つのアウトプットから構成され,1つの処理をするようにする.

⑤ コーディング手法として,プログラムの構造的な特長を表わすために,機能上の階層に応じてインデンテーションをつける.このことにより流れ図に余り頼ることなく読むことができるようになる.

⑥ 機能仕様書で使用されている名称を引き合いにして意味のあるデータ名を付ける.こうすることにより,プログラマーが仕様書とコーディングとの間を円滑に理解することを可能にする.

⑦ 意味をもたせたコメントは,ある場合には,プロ

グラマーがストラクチヤー化されたプログラムを読み，理解する助けとなる．特にアセンブラ言語を使用する場合に有効である．

3) 利　点

ストラクチャード・プログラミング手法の主な利点はつぎの4つである．

① 機能やロジックを確認するために，プログラムを体系的に読むことができる．事実，エラーの大部分はこの方法で検出できる．

② プログラマーがお互いのコーディングをチェックすることあるいは参照することを可能にすることにより，教育的効果を引き出すことができる．

③ ストラクチャード・プログラミング手法はドキュメンテーションの性格を持っており，コーディング・リストの大部分はドキュメンテーションとして利用できる．

④ ストラクチャー化されたプログラムを作成するための規則がプログラムの品質を高め，修正を容易にするという予期せぬ効果をもたらす．

4. プログラミング・プロダクション・ライブラリー

プログラミング・プロダクション・ライブラリーは，プログラム開発における事務的な仕事，ライブラリーの整理や維持などの雑用，キイパンチあるいはコンピュータの操作などの繁雑な作業からプログラマーを解放し，プログラム作業に専念できることを目的としたオフィス手順およびコンピュータ手順を体系化したものである．

オフィス手順とはコンピュータ手順のためのインプットの作成，たとえばジョブ・コントロール言語，リンケージ・エジティング言語，プログラムのソースやテスト・データやコンピュータからのアウトプットのファイルや外部ライブラリーの記録帳の維持などをいう．

コンピュータ手順は，ディスク・ファイル上のライブラリーのデータをコンピュータを使用して維持したり，処理したりすることである．

B. 新しいドキュメンテーション手法

1. ハ イ ポ

HIPOとは Hierarchy Plus Input Output の略で，プログラムの機能構造を階層的に表わし，各機能をインプット，プロセス，アウトプットに分け，それぞれ関連づけて写実的に説明する文書化手法で，システム

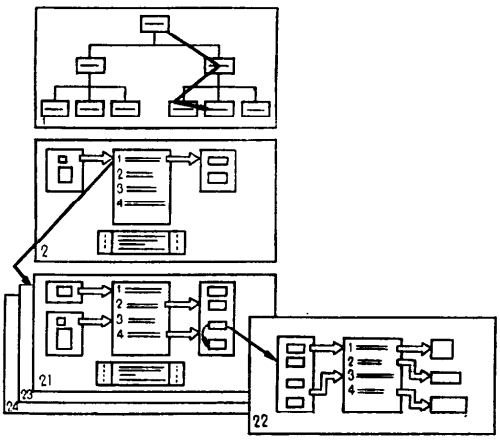


図-3 HIPO パッケージ

設計段階，プログラム開発段階で文書化のために使用される．その主な目的は，コミュニケーションの手順と手法を改善することであり，維持性，信頼性および管理性の向上に有効である．

① この手法は開発や導入過程の自然な姿を反映し，習熟も容易である．

② HIPO はそれを通してお互いに理解し合える共通の基盤であり写実的言語である．

1) 作 成 手 順

HIPO のパッケージは，

① 可視性体系図

② 概括ダイアグラム

③ 詳細ダイアグラム

により構成される．

可視性体系図は図-3 のごとく階層的に各機能の関係を全体的に理解することを目的に作成される．

概括的ダイアグラムはインプット，アウトプット及びプロセス要件をリスト・アップし結びつけ後，整理単純化して作成する．

詳細ダイアグラムは更に下位のレベルの機能（モジュール）を HIPO の規則に従って作成したものであり，写実的に表現できない部分や補足説明のために記述を加えることもある．HIPO で表現できないようなケースについては，従来より使われている流れ図を付けることもある，

2) 利　点

HIPO 文書化法の利点はつぎのとおりである．

① 上位のレベルの機能を写実的に表現し，階層的な構造を準備することにより，内部機能の問題のある箇所を容易に把握できる．

② この手法は，構造化されたダイアグラムを作成することにより，あるいはまた共通の可視的な構成図を用意することにより，教育やコミュニケーションの問題の軽減に有効である．プログラマーは最初は概要を把握でき，次々に下位のレベルの機能を知ることができるので理解しやすい．

③ その他の利点として，プログラムの機能を可視的に表現できることにより，間接的な効果としてシステムの設計，開発および導入をより適切に管理できることなどがあげられる．

C. 新しいレビューの手法

1. ストラクチード・ウォークスルー

プログラム開発プロジェクトにおける難かしい問題に，

① 進捗状況をどのようにして把握するか．

② 製品の品質が要件を満足しているかどうかをどのようにして把握するか．

③ どこに注意の力点を置くべきかをいかにして見出すか．

ということがある．このような問題を解決するために採られてきたこれまでの手法は，

① プロジェクトの段階の大きな切れ目（節）たとえば，システム設計やプログラム設計の終りごとにフェイズ・レビューをする．

② 一定の期間ごとに定期的にレビューする．

③ プロジェクトの重要な局面で第三者のオーディットを受ける．

などがある．しかし，これらのレビューは，必ずしもその意図したような効果は得られなく，いろいろな問題を惹起した．その主なものは，

① レビュー期間中におけるレビューをする人とされる人との関係は，敵対関係にあり，お互いに敵意に満ち，衝突を繰り返すばかりである．

② レビューする人は，あたかも検察官のごとく，相手の過ちを見つけ出すことのみに一生懸命になる．

③ レビューされる人にとっては，レビューによって何も得るものはなく，レビューする人にプロジェクトの状況を理解させたに過ぎず，スケジュールの遅れに拍車をかけるような検討項目のリストが作られる非生産的な会議に貴重な時間を費してしまったと感じる．

などである．

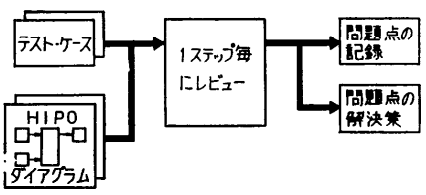


図-4 デザイン・ウォークスルー

1) 目 的

ストラクチャード・ウォークスルーとは，上述のごとき否定的態度を改め，あらかじめ決められた一定の運営ルールにしたがってシステム開発サイクルのいろいろな時点で，いろいろな目的で実施される一連のレビューの手法でありその主要目的は，

① エラーの早期発見につとめ，影響の及ぶ範囲を最小限にくいとめる．

② 完成度合を評価する．

である．

2) 特 長

ストラクチャード・ウォークスルーが，従来のレビューと大きく異なるのは，つぎのような特長を有するからである．

① 会議の準備は，レビューをする人の人選を含めて全てレビューを受ける側で行う．

② ウォークスルーは要員の評価をすることが目的ではないので，マネジメントは会議に出席しないことを原則とする．

③ レビューに必要な情報や資料は，会議が開催される以前にレビューする人に配布しておき，その内容についてはレビューをする人は熟知しているとの前提で会議は進める．

④ ウォークスルーの主催者側は，レビューをする人々にその役割を前以って知らせておく．

⑤ ウォークスルーの重点はエラーの訂正にあるのではなく，その発見に置かれる．

3) 運 営

ストラクチャード・ウォークスルーはつぎのように運営される．

① 書記を1人任命し，検出されたエラーや矛盾点を記録する．

② 書記は会議終了後直ちに出席者全員に記録したものを配布する．

③ 会議は1,2時間を予定し，必ず予定時間内に終らせるように運営する．時間内に終らない場合には次回廻しとする．

④ レビューする人はせいぜい4人から6人位で構成される．

⑤ レビューの内容は，時間を効果的に使うため，本質的な問題に限定し，小さな問題については個々に行う．

⑥ 何らかの処置が要求される項目についてのリストを作成し，出席者全員に配布する．出席者は，適切な処置がとられたか否かを監視する．

問題点の処置が適切に行われたか否かを継続して追求することがこの手法の重要な点である．

おわりに

以上述べたごとき新しいプログラミング，ドキュメンテーションあるいはレビューの手法を適用した結果，生産性，品質，維持性ともに5倍近い改善がされたとの報告もされている．これら新しいシステム開発手法のみならず，プロジェクト・マネージメントが重要であることは言うまでもない．中でも，プログラミングにおいては完成度合を測定することが難かしく，ある段階の次の段階で前段階で見落した問題が検出され再度同じことを繰り返すのが一般的な進め方である．

エラー発見に要するコストは，プログラミングのサイクルが進行するに従って急速に増大することを考えれば，エラーの早期発見が生産性の向上につながることは明らかであり，この問題を解決するためには各段階の完了基準を設け，先に述べた新しいレビューの方法などを有効に使用することが重要である．このようにプログラミングも芸術の域を脱し，生産工程にのった製品と同様に作成される日も近いことであろう．

(昭和50年6月4日受付)