解　説

NHK におけるシステム・メインテナンス†

——メインテナンスの体制・手法・ツール——

大　島　　昭††　　棚　橋　桂　太　郎††

1. はじめに

システムを導入し，運用の歴史をつみ重ねていくにつれ，DP 部門では，メインテナンス業務の占める比率がふえ，これが新しい開発業務の足かせとなってくる．NHK もその例外ではなく，当室の過半数の者が，メインテナンス業務にかかわり合っている現状である．メインテナンスの対象となる「NHK 番組技術システム (NHK-TOPICS*)」は，NHK の基幹業務である番組の編成・制作・送出活動を支える対話形のリアル・タイム・システムであり，1968 年に運用を開始した．ソフトウェアは，当時のマシンの性能・容量を反映して，アセンブル言語で，開発に当ってはメモリ節約が大きな課題であった．その規模は 30 万ステートメントをはるかに越えている．運用開始以来，20回におよぶシステムの改善・改良を継続してきたなかで，メインテナンス業務の効率向上は，日常的な重要な課題であった．本文では，NHK-TOPICS におけるメインテナンス業務を取り上げ，その業務体制，作業手法，使用ツールなどについて述べ，御参考に供したいと思う．

2. バージョン管理

当システムでは，運用当初から，システム機能の更新をバージョン改訂という形で実行・管理してきた．これは，システム機能の変更を，要求があった都度，散発的に実施するのではなく，いくつかの要求項目を総合的に検討し，とりまとめて開発・リリース（バージョン・アップと呼ぶ）する方式であり，このことから，“バージョンとは，一定期間同時に利用できるシステムの諸機能の組合せだ”ということができる．プログラムだけでなく，関連するドキュメント，ファイル形態もバージョンごとに設定するバージョン・ナンバーで管理する．各バージョンのリリース項目に何を選び，その時期をいつにするかは，システム・メインテナンスの出発点である．項目ごとのユーザの要求の強さ，開発負荷，マシン必要時間量，メインテナンス要員の状況，項目相互間の機能的あるいは構造的関連性，性能上の配慮など，様々の要因と総合的に勘案してバージョンにまとめる項目を決定した上，開発のスケジュールを立案する．

バージョン改訂の間隔は，改修プロジェクトを管理し易い大きさという見地から，3ケ月～6ケ月程度が手頃である．改定が小刻みにすぎるのは，改訂に伴うオーバ・ヘッドが重荷になるうえ，項目のうち開発期間が長くかかるものは，複数バージョンにまたがって開発作業を進めることとなり，ライブラリ管理やテスト・データの設定も難かしく，効率が悪いばかりでなく，ミス多発の原因ともなる．

逆に改修の内容が大きすぎると，テスト洩れがふえたり，仕様が陳腐化したりしてトラブルを起しやすく，改修項目の中には，リリースの時期まで待てないものも出てきて，単独のリリースがふえるなど，非能率なことになる．

適切なバージョン設定によるシステム・メインテナンスの管理は，

① プログラムの確実な管理と，安定的なテスト環境が保証され，ソフトウェアの信頼性が維持できる．

② メインテナンス業務が計画的に進められ，合理的な作業手順・マシンの利用・負荷配分を通して効率が上る．

③ テスト環境の準備，システム既存機能の確認テスト，リリース作業などのオーバ・ヘッドを軽減することができる．

などの効果をもたらす．

† “Maintaning Computer Program in a large on-line System: its Organization, Method and Tools in the case of NHK (Japanese Broadcasting Corporation)” by Akira OSHIMA and Keitaro TANAHASHI Management Information Bureau NHK (Japan Broadcasting Corporation)

†† 日本放送協会経営情報室

* Total Online Program and Information Control System

3. 文書管理

当システムでは，計画書・設計書・仕様書などシステム関連の文書類の管理に，集中ファイル方式を採用している．各人は常に更新された最新の情報で仕事をするために，個人用の文書を持たず，必要に応じ，マスターの原紙から，使い捨てのコピーをとって仕事をする．文書類は，分類基準・用紙・処理手続が決められているし，また，集中管理のための文書専任者（現在1名）を置き，文書番号を付しての文書登録・ファイリング・コピー・配布をおこなっている．

システム・メインテナンスは，既存文書の変更を伴うが，新規の文書と同じ手続きを踏み，具体化作業に先立って，必ず変更の仕様や手法を文書化し，監督者・管理者のチェックと承認を受ける．既存文書の更新には変更届をつけ，それに，変更が必要になった事情，変更内容の要約，変更ページ，変更実施の予定日またはバージョン・ナンバーを明記する．本文の変更・削除箇所は，消しゴムなどで完全に消すことをさせない．これは，文書の更新が実際のプログラム変更作業の前におこなわれるため，更新前の内容がしばらくの間有効で，時折，更新前の部分を参照する必要が生じるからである．従って，更新箇所は読める程度の線を引くことで抹消とし，必ず変更時点を注記することにしている．

4. メインテナンス・チームの運営

メインテナンス関係者は，業務別・機能別に分かれ，6～10 名程度の構成員からなるチームに所属している．メインテナンス項目は，各バージョン設定時点で，チームの受け持ちに従って分担が決められ，以降，各項目についての詳細要件の固めから，プログラム開発，機能テストまで，同一チーフが一貫して担当する．総合テストは各チームの共同作業となる．

各チームには，詳細要件に責任をもつデスクと，ソフトウェアの作成に責任をもつチーフとをおき，直接担当者の作る文書やプログラムを検証する．こうした体制をとっているのは，プロダクトの質は担当者のスキルに依存すべきではなく，チーム全体の力量を反映させるべきだという考え方に基づいている．従って複雑な，あるいは高度な設計に関しては，ラフ・ドラフトの段階でチーム内や関係チームとの検討会を開き，早期に，設計の落ち，ミス，不適切を発見し，ソフトウェアの品質を向上させる努力を重ねている．

5. マシン運用形態

現在，当システムの主計算機は IBM 370-158 で，デュプレクス・モードで運用している．予備機は通常 VMF 370 (Virtual Machine Facility 370) と呼ぶ上位 OS の下で，2系統の仮想計算機（略称 VM）と，下位の OS の CMS (Conversational Monitor System) を介して，数台の CRT 端末を時分割で稼動させている．この運用形態をとっている目的は，できる限りオンライン・モードでのマシン利用時間を提供できるようにすることである．

VM は各々独自の OS と入出力装置をもち，見かけ上独立した計算機のように扱えるので，利用者は性能面を除くならば，他の VM をまったく気にしないで済む．VMの中の1台は，当システムの開発・メインテナンス専用で，テスト用端末十数台，開発用ライブラリ，およびテスト用データベースが常時セットされているディスク・ドライブが接続されていて，1日5時間以上のオンライン・テストランが保証されている．また，バッチ・ジョブは，オンラインのバック・グランドで2ジョブまで実行できる．なお，もう1台のVMは，経営情報検索など様々な業務に使用されている．VM による運用時間帯は全体の8割以上におよび，これにより，実質5割以上の利用時間増を可能とした．

CMS 端末は計算機室と事務室に数台設置され，ライブラリの更新に使用している．また経営情報検索では単発プログラムの開発のために使用している．

6. ライブラリの運用

ソフトウェアの更新は，各バージョンごとに設定されるライブラリによって管理している．ライブラリは，①ソフトウェアのソースをカード・イメージで収容しているソースライブラリ，②アセンブルの結果得られる機械命令コードを収容するモジュール・ライブラリ，③モジュールを結合編集して得られる実行可能なコードを収容するプログラム・ライブラリからなっている．そして，これらライブラリは，200 MB のディスク・パック1個に収めてある．ライブラリ・パックは，マシンの運行やメインテナンス業務を円滑に遂行するため，目的別に数個運用に供している．現在は，

運用ライブラリ
……本番用とそのバック・アップと計2個
メインテナンス用ライブラリ

……修正作業用とリリース用の計2個
開発用ライブラリ
……次のバージョン用として1個（場合により更に先のバージョン用としてもう1個）

である．

開発用ライブラリは，前の代のバージョンの開発が落ちついて来た時点で，そのライブラリをコピーして新たに設定する．開発用ライブラリが出来上ると，通常は，まず制御モジュール・汎用モジュールの開発がはじめられる．そして，業務プログラムがテスト段階に入る前に，制御・汎用モジュールを完成させるとともに，テスト用データベースを準備する．この手順をふむ事により，業務プログラムの開発を，効率良く実施することができる．総合テスト段階では，リリース準備が並行して進み，テストの最終段階で，開発用ライブラリは，リリース用および修正作業用ライブラリに昇格し，運用開始の日をまつことになり，リリースされると運用ライブラリとなる．

7. ライブラリの検索更新ツール

(1) ソース・ライブラリと CMS

ソース・ライブラリはカード・イメージでプログラムの他に，次のような情報をもっている．

① レコード定義ライブラリ　構成フィールドのシンボル，位置，タイプが定義されているほか，テスト・ツールへのパラメータとなるカード上・プリント上での位置も規定されている．プログラムが当該フィールドを参照するときは，必ずこれを呼び出して，定義されている標準シンボルを使用する．

② マクロ・ライブラリ　ここにはOS提供のマクロも，ユーザが開発したものと共に含まれている．

③ 結合編集ライブラリ　メインテナンスの対象となるすべてのプログラムの結合編集ランの入力カードが，プログラム単位に収容されている．

④ パラメータ・ライブラリ　業務処理，ファイル処理，画面処理で参照する各種テーブルが収められている．

以上の各情報は，開発用ライブラリのものについてのみ，CMS 端末で随時検索・更新が可能である．メインテナンス用ライブラリについては，保安上の見地から，旧来のカード方式である．

CMSは，利用者からの入力を解釈して，端末にメッセージを返したり，ジョブを実行したりする手続き言語を備えている．当システムでは，この言語で CMS 端末の操作手順を組み立てることにより，利用者からみると，画面に出されるメニューを選択する完全会話方式で，ジョブの選別ができるようにしてある．この結果，プログラマーは自席近くの端末から，簡単にカード・イメージの修正・確認，ソース・ライブラリの更新，アセンブル，結合編集の各ランの操作ができるようになり，カードの保管・使用，ジョブ制御言語の煩わしさから解放され，作業能率にして数倍の効果をあげることができた．

(2) シンボル検索

当システムでは，レコード仕様の変更があったとき，そのレコードを参照しているプログラムを洗い出すために，ライブラリ走査用のソフトウェア・ツールを備えている．このツールに，シンボル検索を指定して，変更されるフィールドのシンボルをパラメータとして与えると，そのシンボルが出てくるプログラムの名前と，当該ステートメントをすべてリスト・アップしてくれる．メインテナンス担当者は，このリストのステートメントを調べて，プログラムの修正が必要かどうかを判定する．

(3) 結合編集情報の検索

モジュールを修正したら，そのモジュールを包含している全プログラムを再結合しなければならない．しかし，その数が多いと，落ちこぼれを出す心配がある．

メインテナンス担当者は，この洩れを防ぐために，結合編集ライブラリを全数走査して，プログラム別からモジュール別に再構成するソフトウェア・ツールを用い，モジュール・プログラム対応表の編集プリントをとって，チェックすることにしている．

8. ソフトウェアの構造改良

当システムのように，古い時代に開発されたソフトウェアは，種々の制約から，メンテナビリティを無視せざるを得なかったものが多い．さらにメインテナンスのつみ重ねの中で，その構造も複雑になってくる．1975 年 IBM S/370-158 にグレードアップをおこない．メモリにゆとりが持てるようになったのを機に，本格的にソフトウェアの構造改良に取り組むことにした．もちろん全プログラムに手を加える訳にはいかず，当時，仕様変更を要請されている範囲に限定された．判り易く，扱い易いソフトウェアの構造を実現するに

は，設計に対する考え方を切り替えることが先決であった．

（1） 複合設計

従来の作業手順は，まず，処理手法の概略を考えて総括流れ図にまとめ，その中複雑そうで，かつ，まとまりのありそうな部分を，サブ・モジュールとして設定し，次に，その複雑な部分の流れ図を描き，そこで，同じ様にして更に下位のサブ・モジュールを設定するという，解法中心のやり方であった．これだと，全体の構造は，流れ図の中に埋没して，手法の良し悪しは論じられても，構造の検討には進まず，サブ・モジュールの設定は，個人の判断に委ねられてしまう．

そこで，流れ図は後回しにして，先ずデータ処理機能を分析し，機能構成図を最初に作るという複合設計方式を採用することにした．分析結果の各機能構成要素に，I/O を規定し，相互に比較吟味することで，その独立性・単純性の程度も客観的に判って議論が噛み合うようになり，おのずから改善案が示唆されるようになった．

最終的な機能構成図を基準にして，モジュール設定に進むのであるが，このモジュールの規格の改善も一つの眼目であった．

（2） モジュール規格

モジュール・サイズは，従来 100～300 ステップのものが多く，しかも運用後の追加仕様をとりこんで，機能も複雑化したものが多くなっていた．そこで，新設モジュールは，単一機能に徹し，10 ステップ程度のサイズであっても意に介さず，大きくても 100 ステートメントを超えない範囲におさえ，ONE ENTRY/ONE EXIT とした．この大きさであると，プログラム・リストも2頁以内におさまって，机上での通覧も可能で，内容の理解も確実・容易になった．

このモジュール規格が徹底すると，①1プログラムの構成モジュール数の増加にともなうプログラム・ローディング時間の増，②保有モジュール数の増加にともなうライブラリ・スペースの増加，③モジュール仕様書の作成や，モジュール・テストでのオーバ・ヘッド増などの問題がでてきた．

このため，①仮想メモリ方式の導入を基盤としたプログラム常駐化の促進，②倍密度のディスク・パックを導入して，ライブラリ・スペースを拡大，③機能図の採用による仕様書書式の簡略化，トップ・ダウン方式による単体テストの略式化などの諸対策を講じた．

（3） 制御・汎用プログラムの機能強化

全プログラムの中で，図型処理関連の部分が特に輻輳していたので，これにかかわる制御・汎用プログラムを書き替えることにした．絵素仕様の情報については，極限までパラメータ化を推し進め，その上，制御・汎用プログラムのみがそれに触れるようにした．これによって，業務プログラムは，図型処理を意識せずに，単に抽象的な業務データを扱えば済むようになり，負担が半減した．しかし，その反面，汎用モジュールによるパラメータ・テーブルの走査が激増して，CPU ネックが表面化し，この対策として，テーブルの構成法や索引法を改めるなど，CPU 使用時間の削減に苦労させられた．

（4） 階層機能図（HIPO）の採用

以上の改良に並行して，機能分析に限定したソフトウェア設計書の標準化に着手した．機能分析内容の検索と理解を容易にし，かつ文書化作業の負担の軽減をめざしたHIPO (Hierarchical Input Process Output) 書式の採用である．HIPO は，図解による標準化という特質から，従来の文書記述方式にくらべて，誰もが気軽に設計書を書ける様になり，これで記述の形式，出来映えともに個人差が少なくなった．そして，チーフや管理者にとっても，チェックが容易になって，内容の洩れやミスが，たやすく発見できるようになり，機能分析の良し悪しの指摘も適切になるなどの効果を上げた．

9. コードの改善と ALL の開発

ソフトウェアの構造とともに，コードの改善も大きな課題であった．コードの読みやすさ，判りやすさを目指すコードの改善は，①制御の流れを標準化するための構造化コーディングの導入，②コードの記述レベルを仕様書の記述と大差のない水準に高める手法の開発と運用という，二つの目標があった．

（1） 構造化コーディング

構造化コーディングについては，IBM がアセンブル言語用のマクロを持っていたので，その利用を検討した結果，2～3 の補完措置を講じれば，運用可能と判断して試行に踏み切った．

ポイントは，IF や DO のネストが深くなるのを抑えることにあった．特に，エラー・チェックアウトの処理は，従来のコーディング手法の方が判り易いので，ネストを無視してエラー処理ができるような補完マクロを開発した．また，プログラムの修正で，ネストの数が変るのに応じて桁ずらしをするのが面倒なので，

補助プログラムを作り，自動桁ずらしをするとともに，ネスト構造の整合性もチェックできるようにした．

エラー処理を除けば，100 ステートメント以下のモジュールサイズの下では，ネストはさほど深くならず，制御が遠くへ飛ばないから，CRT 端末の画面で，プログラムを参照・更新する CMS にマッチするものと期待した．

構造化コーディングの導入によって，①制御の流れがパターン化して，他人の書いたコードが追いやすくなったこと，②バグの発生が減少して，プログラムの出来上りが早くなったこと，③ラベルがないので，追加・修正が楽になったこと，などの効果があがった．

しかし，コードの記述レベルと，仕様書との差は，依然として残されたままであった．

（2） 言語開発の方針

構造化コーディングの使用経験を積んだ時点で，高水準言語の開発に着手した．具体的には，アセンブラHの高度なマクロ・プロセッサ機能を活用して，計算式・論理式を自由な形式で記述できる構造化コーディング向きのマクロ命令体系 (Arithmetic and Logical Expression Language 略して ALL と呼ぶ) の設計開発である．既存の高水準言語の利用とプリプロセッサ開発の結合方式をとらず，マクロ方式を採用したのは，次のような理由による．

① 性能の良い，メモリ所要量の少ない目的コードが作れるので，ハードウェアの増強を要しない．

② 既存のレコード定義ライブラリがそのまま使える．

③ アセンブル言語による既存のプログラムを，ステートメント単位に置き替えることができて，二つの言語併用が避けられる．

④ 開発作業量が少なくて済む．(実績値……マクロ本体開発＝2人月，補助ツールなど＝3人月)

⑤ マクロ・プロセッサの機能テストによって，既存言語並みの表記法が可能と判断した．

（3） ALL の機能

ALL の言語仕様開発に当っては，次の諸点に留意した．

① 仕様書記述レベルとの1対1対応を目指して，計算式，論理式が通常のスタイルで記述できるようにし，かつ，式にシンボルをつけて，マクロのオペランドで参照できるようにする．

② 同じ目的で，メモリ内レコード群の操作マクロ，SORT・SEARCH・SELECT の開発．

③ 編集マクロを用意し，編集様式・位置の指定だけで済むようにする．

④ 構造化コーディングの使用経験に基づき，多数並列 (CASE) 構造マクロの機能を強化し，仕様書記述様式と対応させる．

⑤ 当システムで扱っているデータのタイプ（2進数・10 進数・16 進数の整数値，文字ストリング・ビット・エリア，汎用レジスタ）とリテラルを操作対象とする．もちろん，通常，数値の計算・比較においてタイプを気にする必要はない．

ALL の言語仕様の開発に当っては，はじめから利用者となるべき部内の人々と検討会を開き，広く意見吸収に努め，仕様の改善を計った．その結果，言語仕様の開発に7人月を要したが，言語の導入・定着は極めて円滑，迅速に運び，プログラマ1人半日，3回の研修でほぼ完全にマスターさせることができた．

（4） ALL の効果と評価

ALL のコンパイル時間は PL/I, COBOL なみで，その目的コードは，アセンブル言語による上手なコーディングに比べても，経験的に 10% 以下の冗長度である．文法も，PL/I などに近く習得が容易である．

利用の効果は顕著で，コーディング時間，テストでのエラー発生頻度，新人の育成期間など，いずれも半分以下に短縮された．特に複雑な条件文がシンボル1個で扱えるので，手順部分の記述が簡略化し，コードの読み易さ，判り易さが飛躍的に向上した．

10. テスト作業の効率化とツール

テスト作業の形態は，そのシステムのテスト環境によって異なる．当システムも，初期システムの開発期 (1966 年～1968 年) や第一次システム (1968 年～1973 年) においては，まずオフライン・モードでテストを重ね，最終段階になって，オンライン・モードに切りかえるケースが大部分であった．これは，当時テスト用端末が少なく，初期の使い込んでないプログラムはバグが多く，それによるシステム中断が，全体のテスト作業を妨げるからで，それゆえ，他への影響が少ないオフライン・モードのテストで，ほとんどのバグをつぶしておくことが必須だったからである．

オフライン・モードでのテストでは，本番の上位モジュールに代って，当該プログラムを制御してくれるドライバが必要となるが，当システムでは初期に標準ドライバを開発して，今日まで広く利用してきた．

1973 年，テスト用端末の増設を契機として，テスト

の初期段階からオンライン・モードでテストをするようになり，現在ではその方式が大半を占めるようになっている．この傾向を助長した要因としては，前記VM導入によるオンライン・テスト時間帯の増加や，改良技法に基づくオリジナル・コードの品質向上も見逃がせないが，テスト用端末の増設とならぶ最大の要因として，オンライン制御プログラムに組み込まれた様々の操作性改善の措置を挙げておかねばならない．そして，これらによるテスタビリティの向上があってはじめてトップ・ダウン・テスト方式も軌道にのせることができたといえる．

以下，テスト補助用ツールのいくつかに触れておく．

（1） ドライバ

ドライバはテスト・ツールとして作ったソフトウェアであるが，テスト・データの準備，テスト対象モジュールのドライブ，テスト結果の編集・表示など本来の目的のほかに，本番ファイルの異常の確認，パラメータの登録などにも使用している．

これらの処理は，図に示すような4種類のメディア変換機能と，プログラム駆動機能を組み合せておこなう．個々の機能に対応して，$ SET，$ PRINT，$ GET，$ PUT，$ TEST と呼ぶ制御カードを用意してあるので，これらに，パラメータを付けて，適当な順序でドライバに与えれば良い様になっている．

このうち，カードからのデータ入力（$ SET）は，文字形式から内部形式へのタイプ変換をともない，プリンタへの出力（$ PRINT）はその逆変換をともなう．この変換に必要なパラメータ（構成フィールドの内部レコードでの位置・タイプ，プリントの位置・見出し，カード・コラム）は，すべてレコード定義ライブラリに一元的に登録してある．ドライバは，制御カードで指定されたレコード名に基づいてこれらを参照し，変換処理を実行する．

レコード名を指定しないで，カードでデータを入力するには，アセンブル言語の常数定義と同じ表記法でコーディングする．プリント形式は16進表示になる．

（2） 異常終了処理の改善

テスト中のプログラムは，異常終了になるケースも多い．初期の当システムのオンライン・ジョブは，異常終了が即ジョブ全体の中断につながり，そうなるとダンプ・リストをとって再起動をおこなうしか方法がなかった．この非効率を解消するため，現在では，当該タスクだけを中断させ，他の端末のタスクは継続するようにしてある．バグ追求のためのメモリ・ダンプも，テスト・モードの時は制御情報のみプリントするようにし，そのプリントもオンライン・ジョブと多重でおこなうなど，他のオンライン・テストに与える影響を軽微にする改善をほどこした．

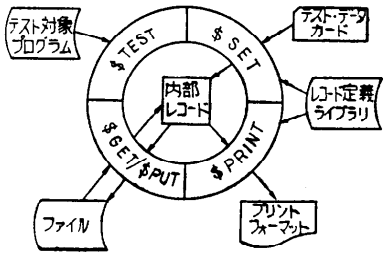

図 ドライバの基本機能

（3） リロード操作の簡略化

非常駐プログラムは，通常，制御がわたる直前にロードされるから，ライブラリの修正・更新の直後からテストが可能で便利である．そこで，業務プログラムに関しては，常駐のものもテスト・モードのときは，非常駐扱いで呼び出し，リロードの手間を省いている．

しかし，制御・汎用モジュールの非常駐扱いは不可能に近く，修正テストにはメモリ上の入れかえ操作が必要である．この操作は，ジョブの終結・再起動という手順をふむのが通常で，これには約10分のシステム中断をともなう．当システムには，コンソール端末からの指令で常駐プログラムをリフレッシュする機能を付加した．これにより，ジョブを継続したままで，単に端末サービスが2〜3秒中断するだけで，常駐プログラムの改修テストに入れるようになった．

（4） ファイル・ディスプレイ

これはCRT端末からファイルの略称とレコードのキーを与えて所要レコード（群）を指定し，その中味を16進数で画面に表示するソフトウェア・ツールである．これで，少数のレコードを更新するプログラムのテスト結果を確認したり，本番でのファイル異常の調査にも利用している．

（5） パフォーマンス・ディスプレイ

コンソール端末からのディスプレイ要求によって，動作中のオンライン・ジョブにおける直近10分間の性能測定値を表示するツールで，性能テストを手際よく進めるのに役立つほか，本番でも，性能管理基準値を超えたとき出す警報を受けて，状況確認の手段として用いている．

11. むすび

今まで述べてきた諸施策により，メインテナンス業務の効率は著るしく向上してきた．これからも，改良技法によって作られるソフトウェアの比率はふえる一方なので，一層の向上が期待できる．

しかし，一方では，保有ソフトウェアは増加を続けていくであろうし，要員の確保は年々難かしくなるであろうから，現状に安住することは許されないし，メインテナンス業務の改善のニーズは更に高まっていくものと思う．

この課題に応じていく上での最大の難関は，新規の開発やメインテナンス業務をおこなう傍ら，メインテナビリティの改善に携わる要員を如何にして捻出するかである．何故ならば，この仕事ほど重要でありながら，地味で，しかも他から理解されにくいものはないからである．

参考文献

1) 大島 昭他：オンライン・システムの計画的進化—放送センターへの移行と NHK-TOPICS の転換—ビジネス・コミュニケーション Vol. 11, No. 7～12 (1974) および Vol. 12, No. 3, No. 5～7, No. 9～10 (1975).
2) 池島慎一：高水準言語 ALL—迅速応答システムの高水準言語化—第 17 回 IBM ユーザ・シンポジウム論文集（2月，1979).
3) 永田靖人：NHK における CMS の導入について IBM REVIEW No. 65 (1977).

(昭和 54 年 6 月 18 日受付)