プログラム制作上の進行管理と品質管理*

竹　下　　享**

1. はじめに

電子計算機は 1950 年代の後半から出現したが，急速に普及，発展して今や会社・銀行・官公庁・大学・研究所・軍関係で多く使われるようになった．数年前までは，そのプログラミングとは個人芸的机上の頭脳作業であって，その細かい進みぐあいはプログラマの独り独りに任せばなしの場合が多かった．しかるに次第に大きなプログラム（または一組のプログラム）が多人数の協同作業により，開発されるようになって来た．このような場合その人事管理もさることながら，いかにしてそのグループの作業の進行と質を管理するかが問題となる．

物を生産または制作するに当たって重要なことは，期限どおりしかも要求された質のものを完成させること，換言すれば工程管理（進行管理）と品質管理である．一般の製造工場ではこれらは以前から行なわれており，その技術も高度に進歩している．しかるにソフトウェアの生産（プログラムの制作）は，その歴史と経験が浅く，また工場生産とは相異なる諸種の要素を含んでいるので，それに関する管理の技術や理論はまだ余り紹介されていない．

工場製品の場合は，通常試作により生産に必要な(物的および人的）資源と時間が確かめられており，生産工程も多くは流れ作業であり，工作機で加工されるので人間の思考・判断は比較的僅かしか要しない．これに反しプログラミングでは，いずれの場合も初めて新しいものを制作するのであり，仕様書き（入出力データおよびプログラムの specification）とブロック・ダイアグラムができていても，実際にプログラムを組んで見ないとわからないようなものが多く潜在し，しかも作業の大部分は人間の頭脳で行なわれるので，制作者の経験・能力・意欲といったものによってその遅速や品質が大きく左右される．

プログラムの開発に当って期限どおり完成させようと思えば，諸種の要素を綿密に検討してから，詳細かつ適正なスケジュールを立て，定期的に進行を検討・評価し，スケジュールや作業の配分に必要な修正を適宜行ない，周到で弾力性のある工程管理をしなければならない，

さらにでき上ったものによい品質を保証するには，最初から統制のとれた標準化・規格化を行ない，制作過程における検査および完成したものの製品試験に万全を期すべきである．

2. スケジュール作成前の問題

2.1 必要とする作業量

生産もしくは制作のスケジュールを組むには，「何を」，「誰が」，「何を使って」，「どの方法で」，「どれだけの期間内に」作るのかを知らねばならない．

まず「何を」であるが，これは換言すれば必要とする作業量（workload）を確かめることである．プログラムの場合にまず調べるべきことは，実際に要求されているもの（計算機メーカからユーザに提供するものあるいは一般のユーザ内ではプログラミング・グループからオペレーション・グループに引渡すもの）のみでなく，その制作や維持に必要なテスト用プログラム（シミュレータや診断ルーチンなど)や各種の補助プログラムを作る必要性があれば，それらを合計した**大きさ**である．場合によっては補助的なものの制作が全体の仕事の中で決して無視できない比率を占めることがある．全体の量はプログラム数，サブルーチン数および命令（instruction）またはステートメント（statement）の数で算出する．

次にそれに要求された**品質**の高低，すなわち機能・能力・性能の程度が問題になる．単独にそれだけで使われるプログラムと他を制御・監督する管理機能を持つプログラムとは，自ら質と複雑さが大いに異なる．特に処理時間と占有するメモリー・スペースから見た効率，入力データのエラーの検出およびその修正に応ずる能力，データや処理方法のいろいろな組み合わせや特殊なケースをカバーする融通性，データの準備や機械操作等の使用上の便利さ，後日の変更・拡張など

* Progress Control and Quality Control in Program Development, by Toru Takeshita (Programming Systems, IBM Japan)

** 日本 IBM

に対する保守のしやすさ，事故の際にさかのぼってやり直し(restart)や他の装置への切換え(switchover)の能力等において，どの程度のことが要求されているか見極める必要がある．

しかるのちにその制作に当っての**難易**を検討する．それは対象業務や問題そのものの性質ばかりでなく，プログラムの内部構造（program logic）により，またメモリー・スペースの使用と処理時間をどれほど小さくするかで，かなり異なってくる．またコーディングに symbolic language (assembler language) を使うか COBOL, FORTRAN, ALGOL, PL/1 といった high level の言語を使うかでも大きな差異がある．前者の場合，命令の形態，インデックシング・レジスタの有無等機械固有の要素と，どの程度 macro やサブルーチンが使えるかも検討すべきである．また他のプログラムとの関係，たとえば別のプログラムの出力データを入力とし自分の出力が他のプログラムの入力となる前後関係とか，多数のプログラムを支配・統轄し，それらに入出力関係を始め種々のサービスを行なうコントロール・プログラムとその支配下にあるオペレーショナル・プログラム（アプリケーション・プログラム）との結び付け等の相互関係が重要である．それに加えて検討しなければならないことは，機械の構成である．それが複雑であればあるほど，プログラムの制作は難しくなる．単にカードか紙テープの読取り・穿孔装置と印刷装置しか付いてない電子計算組織に使うプログラムと，共用ファイル（shared file）を持ち2台のコンピュータが直結（direct couple）し多数の通信回線が接続したシステムに対するプログラムではその難しさに相当の差異がある．

2.2 利用できる資源

制作するものが何であるかわかったら，それに使える人的および物的資源（resource）が問題となる．

まず得られる人（man power）のことを考えるべきである．それは数量的にプログラマ何人，コーダ何人というだけでなく，その質すなわち経験と能力および作業意欲を考慮に入れるべきで，それらにより制作の進行速度と完成品の品質がかなり影響される．プログラマの質を調べるには，過去において経験した機種，プログラミングの期間・量・種類，制作したものの質，またこれまでの作業態度，勤務成績なども参考にするとよい．プログラミングでは経験や能力より仕事に対する熱意がしばしば物をいうことがあることも注意すべきである．

プログラマ個々の質と同時に，制作グループの人員構成がその生産能力に大いに関係して来る．強力なリーダーシップを備え，しかも長年の経験ある指導者の下に構成されたグループと，そうでない場合とでは大分差が生ずる．グループ内では密接かつ友好なチームワークが是非必要であるが，意見の喰違いが出たり，経験の少ない人が方法に迷っているとき，これを直ちに解決して正しい指示ができる人で，極端にいえば，(技術的な面で)専制的権力者がグループの管理者であるとよい．またテストを繰り返えしたが，どうしてもエラーの原因が見付からなくて行詰ったとき，これを速かに解決するようなトラブル・シュータ（trouble shooter）的な人がいると好都合である．

さて次に，プログラム制作に不可欠な段階であるテストに使用する**機械の時間数**が問題となる．新に導入される機械であれば，それがいつ入って来るかが重要な意味を持つ．最初から機械が使えるときは，コーディングを終えた部分から少しずつテストして行き，テストに出したあいだ次の部分のコーディングができるので作業が効率的に進み，正しいとわかったもの（たとえば特定の命令の使い方）は次のプログラムに安心して利用でき，また一度犯した誤りは繰り返えさぬよう気を付けることができる．しかし機械の設置がおくれると，機械に掛けて始めて発見できるような同種のエラーを多く含み，かつプログラム間の関係が確かではないものを一度にテストすることになるので，機械の時間も不足するだけでなくプログラムの修正や書き直しが多くなる恐れがある．

機械がすでに稼動している場合でも，利用できる度合（他の仕事の合間を見てしか使えないか，こちらが要求すればいつでも使わして貰えるか）によって，テストの進行速度は大分変わって来る．またそれは機械の設置場所にも多分に関係する．機械がプログラミング室の隣の部屋にあるのと，隣接したビルにあるのでは大きな違いがあり，遠隔地にある機械を使うときはその不便さからプログラム開発の能率はぐんと低下する．

実際に機械を使ってテストを進めるに当って，その作業を少しでも容易にし，プログラム・エラーの検出が速くできるように，機械そのものをうまく利用するいろいろな**テストの道具**を考えるべきである．その中には trace, memory dump, snapshot 等の診断ルーチンや，始めから全装置と完成したコントロール・プログラムを使わずともテストが可能のように一部ハ

ードウェアとソフトウェアの代用の役割を果たす（テストの段階に応じた）各種シミュレータがある．

またテストの作業に付随する操作，たとえばカードをテープに変換，テープを印刷，ディスクをテープに書出すための utility program と，テスト・データを自動的に作り出すものや，その変更・追加・抽出を行なうものと，テストの結果を見易いように編集し直すようなプログラムのことを十分考慮する，

2.3 開発に要する時間

以上を総合したものが開発に要する時間を決定するといって過言ではないと思う．それは次式のように制作すべきものの量や質的なものより決まる作業量を，人員，機械使用時間や道具といった生産資源(resource）より求まる関数値で割ったもので近似できる．

$$\text{開発に要する時間} \fallingdotseq \frac{w(\text{量，質，難易度})}{r\ (\text{人員，機械時間，道具})}$$

実際にはこれらの関数式と変数値を正確に求めることはまず不可能である．しかし w/r の値としては，(制作量および人員以外のものを無視すれば)，これまでの経験的あるいは**統計的数値**から推測できる．プログラマ一人当たりどれぐらいの仕事ができるものか，自社内のこれまでの実績や計算機メーカやユーザで得られた統計的数値を参考にする．もしこれがないときは，数種の例題か実際の作業の一部をやらせて見て測定する．いずれにせよ開発を始めたなら，一定期間ごとに進行度の統計を取り，かつ平均値を更新して行き，さらにそれを投影もしくは外挿することにより，その後の進行速度に見当を付ける．もちろん学習曲線の上昇による加速度を考慮する．

プログラマの効率のほかに，ブロック・ダイヤグラムの中の丸や四角の図標識は平均何個の命令になるか，それらは1枚の流れ図に幾つあるか，したがって1枚の図は平均どれほどのコーディング量になるかなどのデータがあると，コーディングを始める前にプログラムの大きさ（命令の数）をかなり正確に計算できるであろう．

統計的な一人当たりの1日の制作量に人数を掛けたもので制作量を割って求まる数値を，他の四つの要素を勘案して補正し制作に要する時間を推定する．

航空機座席予約システムのプログラミングでは，1日に10以下の命令が製作されたといわれる．一般には，2進法や10進法の違いとか，固定，可変長あるいは single，3-address などの相違によりかなり差異が生じるが，大体通常の場合10ないし15の命令が製作されると考えてよいと思われる．

2.4 その他の配慮

制作に要する時間を一応推定したならば，別の面（納期や安全率など）から，もう一度制作するものとそのための資源を再検討する必要がある．**製品の納期**すなわち，プログラムの完成期限が絶対変更不可のものであるか，あるいは期限に遅れても大した問題にならないかで，計画の立て方が自ら違って来る．前者のときは，期限に間に合わすために量や質の変更を納入先に交渉したり，また人員や機械の使用時間を増加させなければならないかも知れぬ．

もう一つ忘れてはならないことは，ある決められた期限内に間違いなくやり遂げるためには，すべてにある程度の**余裕**（contingency）を見込んで計画を立てることである．この安全係数（safety factor）の大きさは，納期に間に合わなかった場合の影響の大小によって定まる．制作すべきプログラムの命令数は大きく増加する（多くの場合初期の見積は低く過ぎる）かも知れぬし，納入先の都合その他で多くの変更を要する可能性があるし，テストに使う機械が故障その他で使用できぬことがあるし，プログラマが病気・休暇・結婚・講習・会議等で作業をしないことや，台風・地震などの天災やストライキなどで出勤できないこともあるので，これらをすべて見積っておかねばならぬ．

3. スケジュールの作成法

ソフトウェアの開発には，通常次の六つの段階を経過する．

（1） 業務分析または問題の解析
（2） システム設計
（3） 仕様書き作成
（4） 流れ図作成
（5） コーディング
（6） テスティング

最初の三つの段階は，対象や取扱いの方法によってスケジュール作成の仕方が大いに異なって来るので細部を一般化することは難しい．ここでは主として仕様書き作成後の3段階のスケジュールの組み方について考察したい．これらの段階では，多くの場合人員が多数となり経験の浅い人達の比率が高いので，管理は容易ではない．

まず全体の仕事を見渡して，それを幾つかの**まとまったブロックに分割**する．たとえば一つのプログラムであれば，イニシャリゼーション・ルーチン，主ルー

チン，計算サブルーチン，入出力サブルーチンなどに分け，多数のプログラムの集合よりなるシステムでは，コントロール・プログラム，個々のオペレーショナル・プログラム，補助プログラム等に分ける．

しかるのちに，これらを量・質・難易を考慮して，それと経験・能力・意欲の点で対応できる**プログラマに割当てる**．主ルーチンやコントロール・プログラムをまかせる人は，プログラマとして技術的に優秀でしかも他との協調性の高い人でなければならない．その理由はいうまでもなく，主要部の開発の遅延および品質の低劣は全体の作業に重大な支障を来たすし，他の多くのサブルーチンやプログラムは単独では完全なチェックができないので，主要部の担当者は他の人に協力してテストをせねばならない．

仕事の割当てができたら，その優先順位を決めると同時に，**制作工程を設定する**．流れ図の作成とコーディングの段階はそれ以上分割することは不必要かも知れないが，テストの段階はコンパイラ，オペレーティング・システム，リアルタイム・システムなどの大きなプログラムになると，それぞれに応じた数段階の工程（test, phase または test level）を設定すべきである．普通は簡単で少量の通常なデータでプログラムを個別に全装置を使わずにチェックする作業（単位テスト）から複雑で大量のしかも例外のケースを含むデータを他のプログラムと同時に全装置を使ってテストする工程（システム・テスト）に移行するのがよい．こうすれば，機械の使用時間，操作員の人数，機械の運転費を減らし，なおかつプログラム・エラーを発見し易く，初心者の訓練にむいており，さらに都合がよいのは進行管理をやり易くすることである．

制作工程が確立したら，できるだけ**細いスケジュールを立てる**．それには各プログラマに割当てたプログラムまたはサブルーチンの一つ一つについて，制作工程ごとに開始日と完成目標日を定めて，たとえば次の項目を用紙に記入する．

（1）　プログラム名
（2）　プログラム名
（3）　命令の数（推定）
（4）　流れ図作成開始日
（5）　流れ図完了目標日
（6）　コーディング開始日
（7）　コーディング完了目標日
（8）　テスト第1工程開始日
（9）　テスト第1工程完了目標日
⋮
（16）　テスト最終工程開始日
（17）　テスト最終工程完了目標日
（18）　備考

ここで一つの工程が終ると直ぐ次の工程が開始できるとは限らない．それは他のプログラムができないと次の段階のテストに入れないかも知れぬからである．そこでプログラム間およびプログラマ間の**相互関係をよく考えて**，仕事の優先順位を決定する必要がある．このため PERT のテクニックを応用するとよい．

なお上記のスケジュール作成には，まず各プログラマが予定を立て，それを所属の小グループ内でそのリーダを中心として検討する．その後プログラミング・グループの責任者と一緒に再検討して，全体のスケジュールとにらみ合わせて適当な修正を加えて承認を得る．

4.　進行度の調査と計画の再検討

スケジュールを立てるだけでは工程管理とはいえない．そのスケジュールがどのように守られているか，予定と実際との喰違いがあるか否かを調べ，あるとすれば何らかの措置か対策を講じなければならない．そのために進行状態の定期的検査．分析・評価と計画の修正を行なう．

まず第1に，一定期間の終りごとに**進行状態を示すデータを収集**する必要がある．一つのやり方としてはプログラマに各プログラムの工程別の進行度を記入させ提出して貰う．プログラマが仕事振りをチェックされるとか，記入のため時間を費すのを好まぬなどの嫌悪感を起させないように，できるだけ簡単で手軽に書込める様式を用意することが望ましい．データは遅くとも24時間内に収集・処理して，**現状分析に適当な集計表**を作るべきである．このため特別なプログラムを制作して，コンピュータを利用することが考えられる．この表には各プログラムおよび各プログラマについて，工程別の今期間の進行量（命令数と百分率で表わす）と現時点の累積進行度などの詳細なデータと，一見しておよその現状がつかめるようにそれらを集約した各工程で進行中の数量の合計と全体との比率等が含まれており，さらにそれぞれのスケジュールとの比較が示されているとよい．

この表とは別に，その期間に達成したおもなことや，生じた出来事，解決せねばならぬ問題等をまとめた定期的**進捗状況報告書**を作成する．

以上によって，マネージメントは現状をミクロ的およびマクロ的に把握すると同時に，生産性を分析することができる．スケジュールと実際の進行度との比較から計画どおり進んでいるか否かを知り，もし遅れているとすればその原因を追求し，どこに問題点があるかを見出す．かくして現在の進行状況が正確に評価でき，仕事や人員の配分を変更するとか，スケジュールをより現実的なものに修正するなど，時宜を得た措置を取ることが可能となる．小さな遅れはできるだけ早く取り戻すようにし，積り積って重大な遅延を来たさぬようにする．

なお機械の使用時間やプログラマの作業内容の統計を作れば，上記に多少なりとも参考になるであろう．

5. 品質管理の方法

でき上った製品（プログラム）は，下に記すような条件を完全に満さなければならない．

（1） 仕様に指定された機能を正しく行なう．
（2） プログラム・エラーがない．
（3） 例外事項を含むあらゆるケースを，カバーする．
（4） 誤ったデータを検出し，その訂正は容易である．
（5） 取扱いおよび保守は便利である．
（6） 使用するメモリー・スペースと処理時間の点で効率がよい．
（7） 運転中に事故が起きたとき，さか上って操作し直しや他のシステムに切換えられる．

これらの点につき完成後の検査は必須であるが，あとで修正ややり直しを少なくするため，始めから計画立って高い品質のものを作るようにすべきである．それには制作グループ内では，品質管理の方法をよく研究し実施しなければならない．それに加えて，制作グループとは，独立した小グループによる検査が望ましい．なぜならば，制作者自身がその作品を見た場合に，主観的や自己ひいき目な面をゼロにすることがむずかしく，また独りでは気付かぬ点が存在するためである．ここでは開発グループと製品試験グループのそれぞれにおける方法を考えてみたい．

まず標準を守ることが極めて大切である．経験あるプログラマの中にも，入出力データとプログラムの仕様書きや流れ図を書かないで，いきなりコーディングをしてしまう人がいるが，多数の人員でしかも経験の短い人が混ったグループは，この方法は避けるべきだと思う．標準形式の仕様書きを必ず作り，それに従って大まかな流れ図と詳しい流れ図を作り，コーディングに入ってからの修正を最小にすると同時に，第三者をしてその内容が容易に理解でき，検査と評価を受け易くし，あとの保守に便宜を与えることを考えるべきである．

仕様書きの中には，共通なメモリー・スペースの取り方，プログラムとプログラム（またはサブルーチン）との結び付け（linkage），データの引渡しの方法とその形態，使用できるメモリー・スペースの範囲等について詳細に明記したものを作り，全プログラマに配布して周知徹底させ，必ずその規定を守るようにする．仕様書きは変更があれば直ちに更新して，関係者全員に伝達する．また定期的および折に触れてミーティングを開いて，進捗状況や問題点について情報を交換する．このようにして**プログラマ間のコミニケーションをよくする**ことが大切である．

先に開発を能率的に行なうために（特にテストの段階での）制作工程の設定について述べたが，よい品質管理を行なうためには，仕様書き作成の段階から最終テストに至るまでの**工程の順を厳守**させることである．各工程でできぐあいを検査し，エラーのないことを確かめてから次の工程に進むべきである．たとえばテストの第1工程を経ずに第2工程に行くようなことを許すと，その後発生したエラーの原因を見付けることが困難になり，他人に少なからぬ迷惑を掛けることになる．

グループの責任者または経験ある上級プログラマは，下のプログラマのブロック・ダイアグラムやコーディングを時々**スポット・チェック**（部分的抜取検査）する．細いところまで見ることは到底できないが，この方法により個々のできぐあいを知ることができる．ブロック・ダイアグラムの書き方が粗過ぎたり，変則的であったり，コーディングのやり方が冗漫であったり，説明書き（remark または comment）のないものは早期に発見して是正せねばならない．この時期が遅れれば遅れるほどその修正はむずかしくなり，多くの時間を無駄にする．

しかし人間の目で見るチェックでは，コーディング上の約束違反やプログラム・エラーの発見は難かしいので，規則に反するものをシンボリック・プログラムのままかテスト中に自動的に摘発する**違反検出ルーチン**や，プログラム・エラーを発見する助けになるいろいろな**診断ルーチン**を作るとよい．

さらにあらゆる特殊な場合やエラーの入ったケースも考慮されているかを見るために，グループのリーダは**テストに使うデータとその結果**を仕様書きを参照しながら検査すべきである．

前述したように，**制作グループとは全く独立したグループ**があって，品質検査と製品試験が行なわれると理想的である．構成メンバーはその分野の専門家（プログラミングの経験を要しない）と経験あるプログラマよりなり，忍耐強く，疑い深く，協調的で好感を持たれ，しかも押しの強い人が望まれる．

このグループはまず制作グループの作成した入出力データやプログラムの仕様書きを受取り，要求されている条件を満しているかどうか，そのようなものが開発可能か，解法や処理法に誤りがないかを，あらゆる角度から検討する．

その後このグループは独自のテスト・データの作成に当たる．すべての可能性を包含したデータのファイルを作り，予想できる入力データのエラーも故意に挿入する．このデータの作成には自動的にコンピュータ内で発生させるようなプログラムを使うことが考えられるし，一度完成したデータのファイルに追加・取り除き・抽出などのファイル・メインテナンスのプログラムが必要になるかもしれない．

このようなデータを使って，制作グループが完成したものを実際にテストし，その結果を厳格に検討・評価する．そこでエラーが発見されると，入出力データのリストを付して速やかに制作グループに伝達し，また改善の必要性を認めたときはその旨勧告する．制作グループが修正したならば，再びテストをやり直す．

このグループはエラーのないことを確かめることが最大の任務であるが，その製品の性能についてもいろいろとテストを行なう．

こうした制作グループと製品試験グループ両者の品質管理により大きな成果が期待できる．

6. おわりに

以上述べたように，プログラムの制作においても，生産工場のような工程管理や品質管理が可能であり，製品の納期と質はそれによって保証される．また，最低のコストで効率的制作も可能となる．適切な管理のためには，十分な分析，系統立った計画と注意深い評価・修正が必要である．これらの努力は，いずれの場合も必ずよい成果をもたらすと信ずる．

プログラム開発における管理技術が，今後一層発展することを期待する．

（昭和40年6月12日受付）