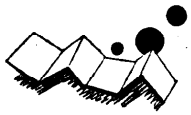
解　説



ソフトウェア製品の生産計画と工程管理†

芝　田　寛　二††

1. まえがき

ソフトウェアプロジェクトにおける生産管理上のトラブルの多くは，工程，工数両面にわたる見積不良，特にその大半は，アンダエスティメイトによるものであることは，ソフトウェア生産に携わるものにとって，共通的な実感であろう．J. H. Lehman[1] は，AIAA のコンピュータシステム技術委員会の調査結果として，調査対象プロジェクトの実に 46% は，工完遅延をきたし，しかも，その遅延平均は 7 箇月であったと報告している．また一方，G. J. Myers[2] は，「ソフトウェア信頼性」の中で，やはり，米国におけるアンケート結果として，トラブルを起こしたプロジェクトの原因は，技術的問題よりも，むしろ，

（1） きついスケジュールのために設計が完了しないうちにコーディングを始めなければならなかったこと．

（2） プログラマ教育に十分配慮がなかったこと．

（3） 割り当てた仕事とプログラマの能力が，不相応であったこと．

であったと述べており，これらは，いずれも，工期，投入資源等について正しい見積評価がなされていなかったことに起因しているといっても過言ではあるまい．したがって，ソフトウェアライフサイクル管理を考えるうえでの重要課題の一つは，まず，正しい生産計画の立案にあるとの認識のもとに，あえて，表記の題目をあげ，大きく，生産計画にかかわる項目と，工程管理の項目にわけて説明を進める．

2. 生産計画

個人の頭脳労働に依存するところの大きいソフトウェアの生産管理においては，従来，種々の科学的な管理手法は，適用しにくいものとの見方が強かった．しかし，大規模，複雑化し，今や工業製品としての様相を呈してきたソフトウェア製品の生産に対応していくためには，ほかの先進産業で開発されてきたさまざまな管理技術を積極的に導入し，ソフトウェアに適した管理技術として，生産管理を展開していくことが，我々ソフトウェア産業に携わるものにとっても必定となってきた．

2.1 管理基準値の設定

ほかの産業の生産計画立案に際して，ST（標準時間）と標準日程の展開がベースとなっているように，ソフトウェアの生産計画の精度向上を図るためには，まず，各種の管理基準値の設定が必要となる．

2.1.1 資源の基準値

ソフトウェアの製造原価は，ご承知のとおり，その 90% は，加工費（人件費）と計算機使用料で占められており，まず，この投入資源の見積基準の設定が必要である．

（1） 基準値設定のための前提条件

基準値を設定しようとするためには，まず，ソフトウェアの生産性そのものの十分な分析と作業の標準化がなされていることが前提である．ただ，Visual でない頭脳労働そのものを分析し，標準化することは，非常に困難なことである．この作業の Visual 化と標準化のよりどころとなるものは，言い古されていることではあるが，やはり，ドキュメンテーションをおいてほかにない．ここでは深くは触れないが，各生産工程における中間成果物としてのドキュメント体系，形式，および記述方法の標準化をすることにより，ソフトウェア生産業務の標準化の基礎が築かれていることが第一である．

次に，ソフトウェアの場合には，特に，そこに携わる個人の能力差，更には，マネージャの能力も含めたグループとしての能力差が生産性を大きく左右することも衆知のとおりである．このため，これも定量的把握が困難な事項であるが，各プログラマの能力評価とレベル別けをして，標準作業者に対応した基準値設定

† Project Planning and Phase Management of Software Product by Kanji SIBATA (Software Works, Hitachi, Ltd.).

†† (株)日立製作所ソフトウェア工場

を可能とするベース作りも必要となる.

(2) 実績の蓄積と分析

基準値設定のためには,当然のことながら十分なデータの蓄積とその分析が必要となる.データの収集は第三者が行えることが望ましいが,頭脳労働にとっては困難であり,各人の作業報告にたよらざるをえない.しかし,データの信頼度を高めるためには,時折タイム・カードとの照合,あるいはワークサンプリング法等によるマクロな検証を行うことも有効である.

ただ,このような標準化,データ収集,基準値設定等の管理制度を推進しようとするとき,もっとも障害となると思われることは,プログラマの"管理されること"に対する抵抗感であろう.

しかし,ここで考えなければならないことは,ソフトウェア開発のトラブルで,もっとも苦労しているのは誰かということである.過少見積によって大幅な工程遅延をきたし,最終工程で徹夜で挽回に努めているのも,また,形もないソフトウェアがなぜこんなに金食い虫なのかを幹部や顧客に苦労しながら弁明しているのも,共に,ソフトウェア開発者自身である.すなわち,標準化の推進や,管理基準の設定は,いずれも管理されるためにあるのではなく,自らを定量的に武装するための大切な武具である点を,プログラム部門の人々自身が自覚し,一つの目標管理の尺度を設定するために協力していく意識の徹底がまず何より大切である.

(3) 基準値の定義

以上のような作業の標準化,プログラマのクラス別け等の前提に立って,資源基準値の一般的定義を考えてみると次のようになり,一般の ST (Standerd Time) の定義とほぼ同じ表現になる.

「所定の作業方法のもとで,その作業について,標準的なプログラマが作業を遂行するのに必要となる基準工数および基準計算機使用時間」

(4) 基準値要因と構造

ソフトウェアの生産効率を左右する要因については,多くの事例が報告されているが,データ量,分析の詳細さ等から,いささか古典的ではあるが,SDC 社が行った研究結果の一連のレポート[4)~6)]は,非常に具体的であり参考となる.

これらの多数の要因の中から,自社に適した基準値の要因を選定する必要がある.選択に当たっては,実績の区分,把握が確実にでき,フォロー可能であること,また,運用が複雑にならない範囲に極力限定することが必要であり,作業実績報告書の設計と同時に検討し,それに合わせて地道に実績把握を進める必要がある.

これらの要因を踏まえて,基準値の構造を検討するわけであるが,この場合も極力シンプルなものとし,かつ,一度決めた構造は,よほどのことがないかぎり変更せず,確実に実績をフィードバックしつつ気長にテーブル値の精度向上に努めてゆかなければならない.

また,一方ソフトウェアの生産性向上のためには,プログラマ自身の意欲が大きく寄与することから,基準値設定にあたっては,単なる実績に基づく見積基準値という役割のみでなく,長期的な目標値としての役割を無視することはできない.

基準値テーブルの一つの例を図-1 に示す.

このテーブルでは,基準値の要因として,(a)開発対象機種(どのレベルの機種で使われるプログラムか),(b)プログラム種別,(c)開発の新規性(難易度),(d)使用言語,(e)使用機種(開発時に使用する機種)を取り上げている.このほかの要因として,例えば,担当者の類似製品の開発経験,チームの構成等が大きく生産効率に影響することが考えられるが,これらは前述のとおり,定量的に実績をフォローしたり,また,見積時に特定することが非常に困難なため,テーブルの要因からは省いてある.

2.1.2 規模の見積

生産計画策定のための根拠となる製品規模を表わす値としては,種々の問題点は指摘されているものの,当面は,やはりステップ数以外に実用となる指標は見当たらない.

モジュール数で表わす方が,より適切とする説もあるが,これも構造設計技法の徹底等,設計の標準化を進めることにより,モジュール数と,ステップ数の相関は,より高められてゆく方向にあり,特に区別する意義は薄らぐと思われる.また,パス(処理ルート)の数により評価しようとする説も参考にはなるが,見積時点でパス数を明確化することはより困難であり,かつ,完成品の評価に当たっても,パスの数は難易度を表わすと同時に,プログラムの出来,不出来を表現する結果となる点は,ステップ数でのデメリットと共通する点があり,ステップ数に比較して扱いが複雑なわりに,特に優れた指標とは考えにくい.機能を量的に適切に表現できる指標が見い出されるまでは,やはり,ステップ数にたよらざるをえないだろう.

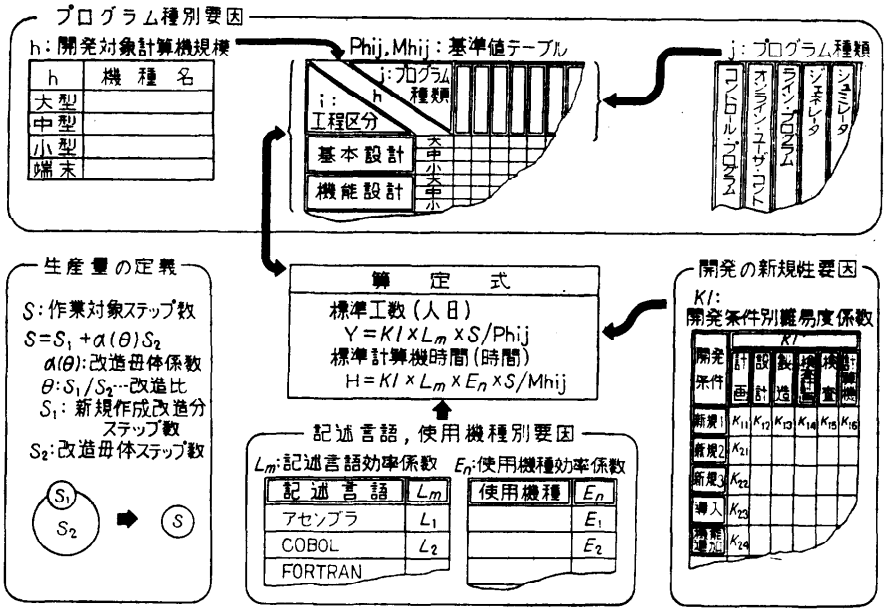


図-1 基準値の構造と要因例

ただ，ソフトウェア製品にとって非常に大きなウエイトを占める改造作業において，単に新たに手を加えたり，追加したステップ数だけで見積をすることはナンセンスであり，既存部分の見直し，あるいは，インタフェースの確認等による工数増を定量的に評価するための補正係数の設定は不可欠である．

2.1.3 標準工期

以上，生産計画立案のための投入資源基準値を主体に説明してきたが，ソフトウェア生産の場合，標準工期の設定は，更に困難な要素を含んでいる．この点は，F. P. Brooks, Jr. がその著書に述べている工数と工期の関係からも明らかである．

しかし，工期を左右する要素も，ほぼ工数基準値の設定要因と類似しており，基準値要因ごとに標準工期を設定しておくことは，生産計画の策定のために有効である．特に，投入工数とはある程度無関係に，システムの種類と規模に対応して，各工程ごとに最低限かけなければならない必要工期があり，この一定期間を割ると，急速に原価が高騰し，品質が悪化してくることが経験的に把握されており，このように無理な計画を排除するためにも標準工期の設定が必要である．

3. 工程管理

2項で述べてきた標準工数，標準工期等をもとに，かなり精度の高い生産計画が策定されていることを前提に，次に工程（進度）管理について述べる．

3.1 工程区分の明確化の中間成果物の評価

ほかの産業のように，各ジョブ・ショップを仕掛品が流れていくのと異なり，一つのプロジェクト・チームの中で，長い仕掛期間を掛けて徐々に製品が完成されていくソフトウェア生産にとって，工程の進度把握は至難の業である．特に大切なのは，中間工程での品質状況の把握であり，ソフトウェア工程管理にとって，その困難さの大半は品質管理にあるといっても過言でなく，これを解決するためにさまざまな施策がとられている．

まず，同一部署において作業が進められているソフトウェアの工程にとって必要なことは，工程の区分とその各工程における作業内容の定義を明確化することであり，そのための作業手順が，確立されていることが大前提となる．

図-2 に，工程区分と作業の流れの例を示す．この流れの特徴は，工程および，品質上の問題点を少しでも早い工程で把握し，結果をフィードバックしていくために，製品開発部署とまったく独立の組織として検査部署を設定し，最初の工程から中間成果物としてのドキュメントの検査をして，一工程ずつ確実にチェックしながら工程を進めてゆく体制をとっている点である．

また，検査部署は，このドキュメントの検査をとお

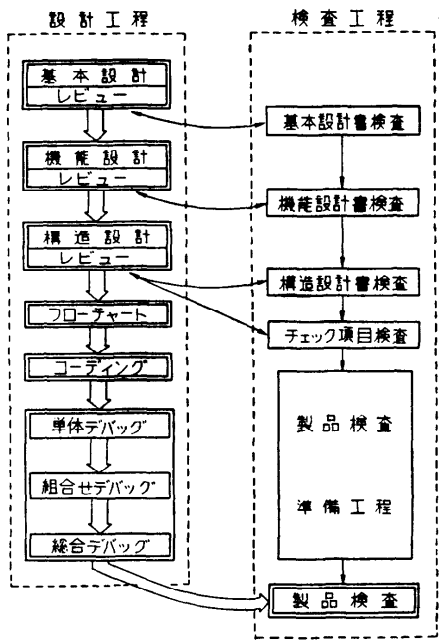


図-2　ソフトウェア生産工程

して製品を十分理解し，検査独自の立場で検査項目の設定から，検査プログラム，検査データの作成まで行い最終製品検査に備える．

このようなソフトウェアの検査の存在意義については，効率，技術両面から疑問視する意見もあるが，G. J. Myers は前述の著書の中で，ソフトウェアといえども，テストのためにはテストのための特別な技術を必要とすること，および作成者とテスト者の不良摘出意欲の歴然たる差の両面から，テストの専門部署の必要性を力説しており，この点まったく同感である．我々の経験でも，設計者が一応完成と判断してから，検査および検査の指摘に基づき，設計者が新たな見地で見直すことにより摘出する不良の比率は，非常に大きな値である．

このように質的チェック・ポイントとチェック機構をもつことで，はじめてソフトウェアの工程管理の基礎作りがなされているといっても過言ではない．

3.2　設計工程の管理

全体の工程のマクロな管理には，ほかのプロジェクト進捗管理と同様に，アローダイヤグラムを用いることが有効であるが，ここでは，よりソフトウェア工程管理に特徴的な点に絞り，管理の対象物の違いから管理方式に差のある設計工程と，デバッグ工程に別けて記述する．まず，設計工程における進度状況を把握する指標としては，そこで作成されるドキュメントの完成度以外にはない．

各ドキュメントの設計完了，デザインレビュ完了，検査完了，検査合格といった各工程の節の管理を主体として，更に大規模なプロジェクトについては，その中間段階での出来上り量をとらえてゆくことも必要となる．ただし，ドキュメントの出来上り量が進度管理指標として意味をもつためには，ドキュメントの冗長度，粗密度に個人差が少ないことが重要であり，そのためには，ドキュメントの標準化，定型化，更には図表化がなされていることが必要である．

その意味で，最近種々開発され発表されている各種の要求定義手法，設計技法等は，設計効率面のみならず，管理精度の向上のためにも大きく寄与するものである．

また，デザインレビュの重要性については，今までにも十分取り上げられているので詳しくは触れないが，単に品質向上のみならず，工程区分の明確化とその品質進度把握のためにも非常に大切な役割を果たして有効である．

3.3　デバッグ工程の管理

3.3.1　環境条件の整備

デバッグの工程管理をスムーズに進行させるために特に重要なことは，事前にその環境条件が十分に整えられているように工程をフォローしていくことであり，その主なものは次の2点である．

（1）　前準備工程

設計工程と並行して進められるべき

（a）　プログラムチェック項目設定

（b）　テスト仕様書作成

（c）　テストデータ，テストプログラム作成

といったデバッグ準備工程はややもすると，ドキュメント作成，フローチャート，デバッグといったメインの工程の推進に隠れて軽んじられ勝ちであり，これらの工程計画をオーソライズして一件ずつ確実にフォローすることがまず大切である．

デバッグ工程の立ち上がりの悪さの要因を調べてみると，意外とプログラムの質以前の問題として，泥棒を捕えて縄をなっているような準備不足によるケースが少なくないものである．

（2）　治工具類の準備

これも当然なことではあるが，デバッグ工程上大きなウエイトを占めるのがデバッグ効率向上のために準

表-1 プログラム・チェック項目設定基準

量的基準

項番	プログラム種別	新規項目	再確認項目
1	コントロール・プログラム	a_1〔件/kStep〕	b_1〔件/kStep〕
2	オンライン・コントロール・プログラム	a_2	b_2
3	通信制御プログラム	a_3	b_3
4	ランゲージ・プログラム	a_4	b_4

質的基準

（プログラム種類別）

項番	項目分類	比率	項番	項目分類	比率
1	基本項目	C_1	6	互換性項目	C_6
2	限界・境界項目	C_2	7	インタフェース項目	C_7
3	異常項目	C_3	8	特殊条件項目	C_8
4	組合せ項目	C_4	9	その他	C_9
5	周囲条件項目	C_5	10	合　計	100

備される各種の支援プログラムおよび，デバッグマシンの準備状況である．

これらの，デバッグのための環境条件の整備を製品そのものの進捗度と同期化させながら推進するためには，全貌を示すアローダイヤグラムに確実に計画を盛り込むことにより着実にフォローしていくことが有効である．

（3） プログラム・チェック項目設定標準

デバッグ工程における進度管理の中核をなすプログラム・チェック項目そのものの量的・質的な妥当性の是非が，その後のデバッグ工程管理上，非常に重要なウエイトを占めていることはいうまでもない．

最も理想的には，仕様書とプログラム自身から適切なチェック項目が自動的に設定されることであるが，まだ，汎用的に実用化されているものがない現状では，少なくとも，チェック項目の設定のための量的標準が設定されていることが必要である．この設定標準も，ほぼ前述の基準値の設定要因に対応して設定されるのが自然である．

表-1 にチェック項目設定標準の例を示す．

3.3.2 デバッグ工程管理

デバッグ進度管理の中心指標は，

（a） 設計工程で作成されたプログラムチェック項目の確認状況

（b） （a）に対応して摘出された不良件数とその解決状況

であり，この2つをたよりに品質向上度合いをいかに早く，的確に把握して対策を講じるかが，工程管理のポイントである．

F. P. Brooks, Jr. の「ソフトウェアプロジェクトの遅れに際しては，不用意に人を追加投入することは火に油を注ぐようなものである」との法則はよく取り上げられ，一面の真理ではあるが，これだけでは我々にとって救いのない話である．これに対する補足法則として，R. L. Gordon と J. C. Lamb[7] が報告している「遅延を早期にとらえ，かつ，小刻みでなく，まとめて人員を投入すれば，追加人員への情報伝達ロスを補って余りあり，結果的に工期をキープできる」との説こそ我々にとってむしろ意義がある．

このためには，あらゆる品質予測手法を取り入れて，目に見えないソフトウェアの品質状況を少しでも早く把握することが重要である．品質管理の詳細は，別セクションで取り上げられているので，深くは触れないが，工程管理の面から見た品質把握手法の事例を図-3 にまとめて紹介する．

（1） 管理図

デバッグ工程における量的な進捗状況管理手段としては，単体，組合わせ，総合各デバッグ検査の工程ごとにプログラム・チェック項目の確認予定／実績件数

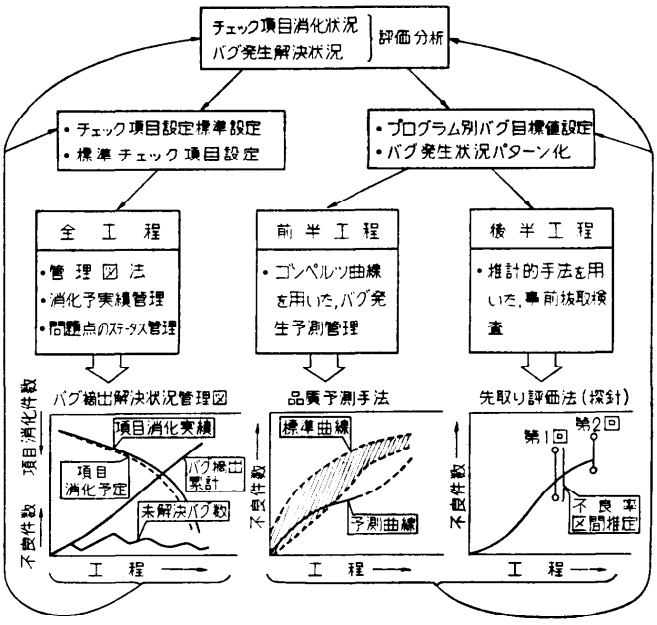


図-3 デバッグ工程における多角的品質把握と早期フィードバック

および，不良摘出件数と解決ステータスを管理図を用いてフォローしてゆくことが管理の基本となる．

バグ摘出累計曲線の収束状況と，チェック項目の残数との相関を見つつ，チェック項目の追加，見直し，工数，計算機資源の追加投入等の対策を早期に打ってゆくことで，遅延を生じはじめたデバッグ工程の完了日を確保することも可能となる．

また，未解決バグがたまり出すという最も危険な徴候も，管理図化することで早期に把握可能となる．

（2） 品質予測手法

チェック項目の確認進度，および不良の発生状況ともに，一般に成長曲線をたどって進行することが知られており，品質状況の早期把握と収束時期の予測のためには，この特徴を生かして種々の品質予測技法を用いることが有効である．

（3） 先取り検査

デバッグ工程中に，少しでも早い時点に，第三者の立場で品質を評価してみるために，検査部署による検査項目の抜き取り検査を実施することも非常に有効である．これにより，推計的手法を用いて母不良率を求め，全体の品質を予測するとともに，具体的な弱点の指摘を行うことにより，設計者は新たな見地で残工程に取り組むことができる．

（4） 不良要因の分析

一方，摘出された不良については，単に量的管理にとどまらずその新規性，組み込み工程，難易度等，不良要因を一件ずつ分析しておくことが必要である．また，ソフトウェアの粗品質は，残念ながら現状では作成者の個人的能力差に大きく左右されるのが実状であり，各個人の不良率，不良の特徴等も同時に把握分析し，当該製品は元より，その後の製品の開発時にもフィードバックできるようデータの蓄積ができる仕掛けが必要である．その内容により次の工程対策が大きく異なってくるはずである．

もちろんこれらの手法は，いづれもまだ一つ一つでは不十分な手法ではあるが，図-3 に示すとおりそれぞれの工程で各手法を併用し，結果を照合，確認し合うことで，より高い精度で品質進度を把握して工程管理を推進していくことが可能である．

4. 生産管理システム

以上，ソフトウェア生産管理における生産計画と，工程管理を中心にその概略を述べてきたが，一見単純そうな管理でありながら，ソフトウェア製品の特徴として，原価（工数），品質，工程，三者間の相互関連度が非常に強く，これらを横断的に関連づけながら把握分析するためには，管理データは膨大なものとなる．

また，ソフトウェアも部品，組立部品，プログラム，システムといった階層構造をもつ大規模な製品が増加してくるにつれて，これらのデータを取りまとめ集計分析する必要性も増加してきている．このため管理工数の削減と，管理の即時性を維持するためにコンピュータを用いた生産管理システムを用いることが望ましい．

ソフトウェア生産管理システムを考えるうえでの一つの特徴として，ほかの管理システムのように，第三者が実績をカウントしたり，自動的にデータを収集することが非常に難しく，すべての粗データは，各プログラマ自身に，握られていることがあげられている．したがって，単に管理者や管理部門のためのシステムでは，プログラマの協力は得られず，また，必要性も認めてもらえない．粗データを提供するプログラマ自身にタイムリーに，かつ，ありがたみのある管理データがフィードバックされるシステムでないかぎり，有効なシステムを開発し，スムーズな運用を進めることはできない．

その意味から，例え小規模な管理システムでも，データの蓄積，分析機能を十分にもったオンライン・データベース形式のシステムであること，また，単に集計表出力だけでなく，各種の管理図出力のできるシステムであることが望ましい．**図-4** に前述のバグ摘出状況管理図の生産管理システム出力例を示す．

ソフトウェアのプロジェクト管理システムとしては，特に米国において，

MITER 社 の SIMON

SDC 社 の ソフトウェア・ファクトリ

等種々報告されており，それぞれ参考になる．しかし，生産技術，生産管理両面の標準化が，まだまだ進んでいない実状から考えると，これらのノウハウは，大いに参考にしつつも，それぞれの会社のソフトウェア生産形態に最も適した独自のシステムを開発し，改善を重ねてゆくことが，現段階では最も有効かつ，現実性のあるソフトウェア生産管理システムの実現につながると思われる．

5. む す び

以上本稿では，ソフトウェアの生産計画と工程管理について，かなり，泥臭い実務面から述べてきた．一

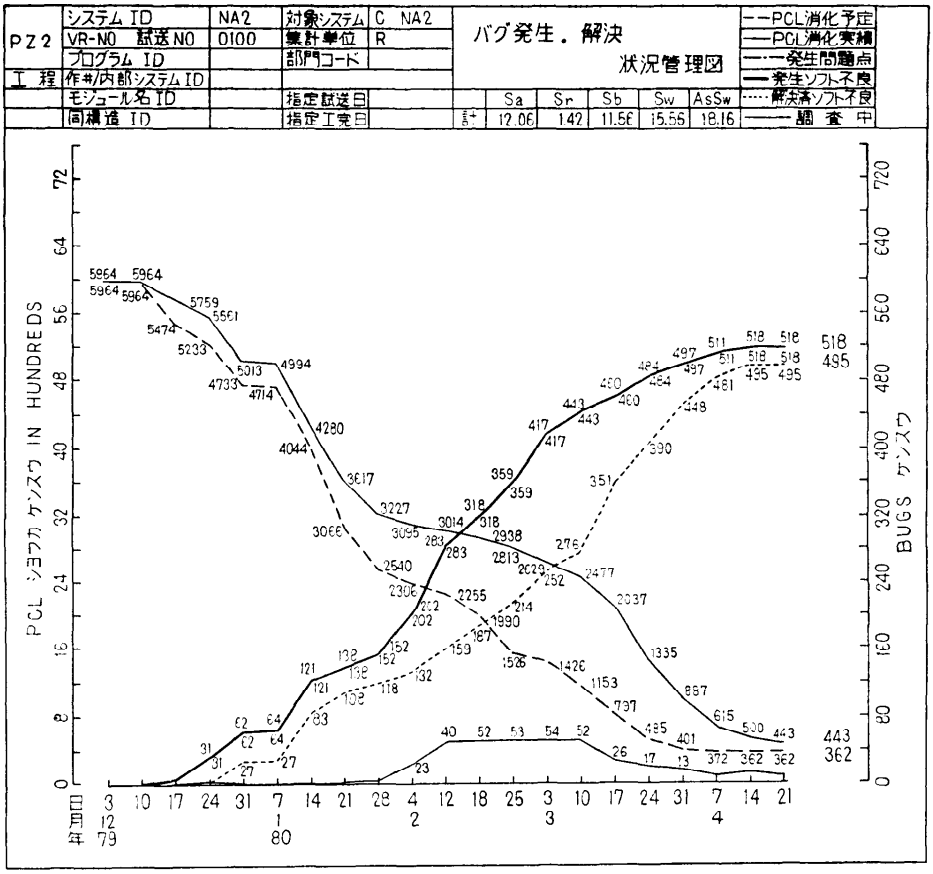


図-1 生産管理システムの出力例

般の生産管理の発展の経過を振り返ってみても，常に生産技術と生産管理は表裏一体をなしており，しかも新技術の開発に管理がピッタリ追随する形で発展してきている．その意味で，ようやく生産技術的なものが形をなしはじめたばかりのソフトウェア産業にとっては，生産管理についても，まだまだ揺籃期に入ったばかりであるのが実態である．しかし，一方，最も管理のゆきとどきにくい設計部門的要素の強いソフトウェア部門が，急膨張を続けていることを思えば，ソフトウェア生産管理方式を1日も早く確立することも急務である．

そのためには，本論でも述べたとおり，まず作業と物の標準化を推進し，その上に立って工程，品質，原価各面にわたる実績データの完全な収集と分析方式を確立することが必要である．本論で述べた各管理方式がごくありふれた方式でありながら，弊事業所で実用に耐えているもの，十余年のデータの積み上げと定期的なフィードバックに裏付けられているからである．また，今後の発展方向を考えると，ソフトウェアは管理の対象となる成果物が，もともとコンピュータに登録されているわけであり，この面から，生産のためのファイルと管理のためのファイルを共用することで，管理データの自動収集を最も効率よく行える分野とも考えられ，今後，ソフトウェア生産自動化ツールの充実と相まって，生産管理システムも急速な発展が期待できるものと思う．

参 考 文 献

1) Lehman, J. H.: How Software Projects are Really Managed, DATAMATION, pp. 119-124 (Jan. 1979).
2) Myers, G. J., 有沢誠訳：ソフトウェアの信頼性，近代科学社 (1977).
3) Brooks, F.P. Jr., 山内彌訳：ソフトウェア開発の神話，企画センタ (1977).

4) Farr, L. and Nanus, B.: Factors that affect the cost of computer programming, Directorate of Computers Electronic Systems Division Air Force System Command. USAF (1964).
5) Farr, L. and Nanus, B.: Some cost contributors to large scale program, AFIPS, Proc, SJCC (1964).
6) Labolle, V.: Development of equations for estimating the costs of computer program production, Electric System Division Air Force Systems Command (Jun. 1966).
7) Gordon, R. L. and Lamb, J. C.: A Close Look At Brook's Law, DATAMATION, pp. 81-86 (Jun. 1977).

(昭和55年6月10日受付)