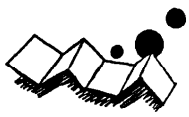
解　説



ソフトウェア生産性の評価と管理†

花　田　収　悦††

1. まえがき

ソフトウェアの生産性を評価しようとすると我々は種々の疑問に遭遇する．それらは本質的な問題を含んでいるものと思われ，その解消には長時間を要しよう．このため，開発環境に則した現実的な前提の下に簡便な方法を用いることにより，正確性の点では若干の難点はあるが大筋をは握できる程度の評価を実施しているのが実情である．

本稿では，我々の開発環境における幾つかの事例に基づきソフトウェアの生産性の評価と基本的と思える管理法について述べる．なお，これらの事例は参考値として理解していただき具体的に応用する場合には自社の開発環境を加味した修正が必要であることをお断わりしておく．

2. 評価上の問題点

生産性の評価の前提を明確にするためまず主要な問題点について整理する．

2.1 生産物と尺度

ソフトウェア開発における生産物として何を選択し，どのように測定するかは基本的な問題であるにもかかわらず統一的な見解がない．ソフトウェア開発における生産物としては少なくともドキュメント（仕様書，設計書など）とプログラムがあるという共通認識はあるが，いまだにドキュメントについては直接の生産物として扱っていない場合が多い．その理由としてはドキュメントの種類や記述内容が規定されておらず多様であることから除外しているものと思われる．しかし，最近ではドキュメントを生産物に加える方向に移りつつあり，その量的な表現法について種々試行を重ねている．

† Evaluation and Management on Software Productivity by Shuetsu HANATA(Processing Program Section Data Process Division, Yokosuka Electrical Communication Laboratory of NTT).

†† 日本電信電話公社横須賀電気通信研究所データ処理研究部処理プログラム研究室

一方，プログラムを生産物とする場合には生産量の単位（尺度）に関する問題が顕在化する．一般には，プログラムの尺度としてソース・コード（行またはステップ）数を用いるものが最も普通であるが，この場合においても以下のような問題点がある．

（1） 処理方式によるステップ数の相違

ソフトウェアにおいては，同一の機能を実現するのに複数の方法が存在しステップ数が異なる．一例として「四捨五入」のアルゴリズムの相違による差を図-1に示す．仮りに，両者が同等の工数で実現したとすれば，一般に「単位工数当りのステップ数」の尺度ではステップ数の大きいプログラム（例1）が小さいプログラム（例2）に比較して2倍以上の生産性が高いという評価を与えることになってしまう．

（例1）

```
GET LIST (X);
S=FIXED (X);
IF X>=0
  THEN IF X-FLOAT (S)>=0.5
         THEN S=S+1;
         ELSE;
  ELSE IF X-FLOAT (S) <=-0.5
         THEN S=S-1;
                    (8ステップ)
```

（例2）

```
IF X>=0
 THEN S=FIXED (X+0.5);
 ELSE S=FIXED (X-0.5);
                    (3ステップ)
```

注1　PL/I 言語による記述例である
注2　変数 S (BIN FIXED), X (FLOAT) については両例とも別に宣言されているものと仮定している

図-1　四捨五入のコーディング例

（2） 記述言語による相違

さらに，同一のアルゴリズムで実現していたとしてもそれを記述する言語によってステップ数が異なる．一般に高水準言語ほどステップ数が少なくてすむ．したがって，場合によってはアセンブラで記述すると生産性が良いという奇妙な現象が生ずるかも知れない．

（3） 生産量に含めるステップ数の範囲

プログラム・リストに表示される種々の情報のうち，どれをステップ数としてカウントするかの問題である．

ステートメント(文)のうち，たとえば「コメント文」は生産量としてのステップ数に含めない，とする場合

がある．コメントはプログラムの読解性の向上には有用な情報があり，「コーディング手引書」などにおいては，むしろ積極的に使用することを指導しているにもかかわらず生産量としてカウントしていない．これは，コメント文をステップ数としてカウントした場合には有用なコメントか否かの判定が必要となりその基準の設定が難しいためと考えられる．我々のプロジェクトにおいてもコメントがいずれも同程度に記入されているという前提の下に，生産量としてはそれを除外した簡便法でわりきるケースが多い．

（4） ツール，テストデータ等の処置

特に大規模なソフトウェアの開発においては，開発を支援するためのツール類や多量のテスト・データを作成する．これらを生産物として認知するかという問題である．さらに，テスト・データについてはプログラムの場合のステップ数への換算をどのようにするかという問題もある．

（5） 改造におけるステップ数の計数

最近のソフトウェアの生産においては既存ソフトウェアを流用して開発する（これを改造と呼ぶ）場合が増加しつつある．ひとまとまりの機能単位の置換または追加の場合には，その生産量としてのステップ数は計算しやすいが，部分的に機能の削除と追加が混在する場合とか，部分的な機能の削除のみの場合にはステップ数の計数が煩雑となる．

現実には，改造前後の差分のステップ数，または改造部分を含むモジュールの改造後の全体ステップ数を生産量と見なすなどの方法が採用されている．

2.2 工　数

一般に開発工数と呼称されているが実態にあいまいさが存在するので考え方を整理する必要があり，以下の3点に集約される．

（1） 開発工数に含めるアクティビティの明確化

ソフトウェアの開発に付随するアクティビティは多岐にわたるため，それらの要否はプロジェクトによって異なる．したがって，複数プロジェクトの生産性を比較する場合などに備えて開発に必須のアクティビティを規定する必要がある．具体的には次のようなアクティビティの取捨を明確にしなければならない．

（ア） システム分析等の評価および研究的活動
（イ） サービス実施部門への技術協力・教育活動
（ウ） マニュアル，運転説明書等の作成
（エ） デバッグマシンの保守・運用等の複数プロジェクトにわたる共通業務

（2） 工数の計数範囲の規定

ソフトウェアの生産は人間の頭脳労働が中心であり，いわゆる紙と鉛筆でできるアクティビティが多い．したがって，必ずしも会社または工場という指定された場所でしか作業ができないというものではない．また，現実に納期がせまりプログラムリストを家に持ち帰りデバッグしている事例を見聞する．

これらの工数を開発工数として計数するのか，さらに計数するとして正確にできるのか，という問題がある．これらの開発形態は好ましいものではないし，また全体の開発工数に占める比率も小さいことなどから指定場所におけるアクティビティの所要工数を計数するのが妥当であろう．ただし，絶対値を基本とする，精度の高い生産性の評価が要求される場合にはこれらの工数の取捨を再考しなければならない．

（3） 標準工数の設定

工数の単位としては「人年」，「人月」などを用いることが多いが，年または月を時間数に換算する場合に問題が生ずる．これは開発体制等の影響が大きいのでそれに合わせた標準工数に換算して評価すべきである．

2.3 品　質

生産物の完成度の尺度のひとつが品質であるが，ソフトウェアの場合には統一的な規準が存在せず，そのソフトウェアが適用される環境等を配慮して経験的に個別の手法によって生産物が使用に耐え得るかを確認している．

一方，サービス開始後にも相当数のソフトウェアの不良（バグ）が検出されている現状からは品質の検証が十分になされていないと推測せざるを得ないが，具体的・客観的な品質の定量化は難しい．したがって，当面の生産性の評価に際しては各ソフトウェアの生産物は，製品としての最少限度の品質について何らかの方法によって確保されているという前提を設定せざるを得ないだろう．

3. 生産性の評価

前節で述べたように，現状においてソフトウェアを評価するには基本的事項において幾つかの問題点がある．しかし，概況を認識する程度であれば現実的に想定される前提の下に，評価対象の特性および生産環境などの開発条件を明確にしてソフトウェアの生産性を評価することができる．

3.1 前提および開発条件

（1） 生 産 量

各ソフトウェアでは最適なアルゴリズムを採用していることを仮定し，ソースプログラムのうちコメント文を除くステップ数で表現する．

改造プログラムについては追加，修正のステップ数のみを計数し削除したステップ数については配慮しない．

また，ツール，テストデータ，およびドキュメントなどについても生産量としての計数の対象にはしない．

（2） 工 数

生産に関連する工数としては，開発工数とシステム分析等の評価，教育および技術普及，マニュアル・説明書等の作成に要する工数のみを計数することとし，さらに，指定の作業場所における労働時間を集計する．

デバッグ・マシンの保守・運用等の複数プロジェクトにわたる共通業務の工数についてはプロジェクト数の変動などがあり着目するプロジェクトの分担率のは握が困難であるため計数の対象から除外する．

また，標準工数としては，1人年＝12ヵ月×21日(252人日)＝12×21×8時 (2,016人時) を用いる．

（3） 対象ソフトウェア

この評価例において対象としているソフトウェアの種類は言語処理プログラム（コンパイラ，ライブラリ・エディタ，デバッグ・ユーティリティなど）である．

改造と新規開発の2種類が混在しているが，規模的には新規作成が多い．なお，改造の内容は機能の追加，アルゴリズムの変更が大部分である．

（4） 記 述 言 語

高級言語とアセンブラの混用であるが大部分が高級言語で記述している．

（5） デバッグマシンの利用形態

最終的な品質確認，新規開発の基本ソフトウェア(制御プログラムなど)とのインタフェースの確認，性能評価などのアクティビティはデバッグ・マシンを専有使用するが，それ以外の大部分のマシンの使用は複数の利用者がリモート端末から会話形式で共同利用する形態である．

（6） 開 発 組 織

複数社による分担開発を基本とする開発形態を採用している．分担部分の相互間のインタフェースの調整は定期的な情報交換等によって行う．

3.2 評 価

生産性のデータはバラつきが大きく常識的には説明のつかないデータも存在するが，数年間の蓄積データに基づき統計的にながめて我々が現時点で認識している点を要約する．

（1） 生産性の推移

最近数年間の単位規模当りの開発費の推移傾向を昭和48年度を基準として図示したものが，図-2である．人件費の増加が48年度に比し，53年度は約70%と想定されるのに対し開発費の増加は 40～50% である．この両者の差，すなわち年平均 4～6% の比率で生産性が向上してきたものと解釈される．

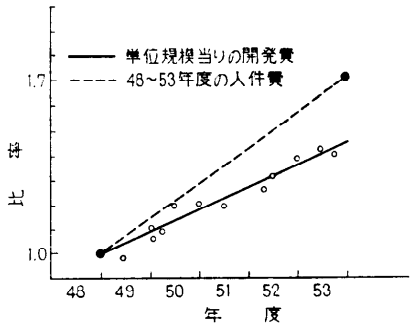


図-2 生産性の推移

（2） 改造比率による生産性の低減傾向

従来から改造の場合は新規開発の場合に比較して生産性が低減するという経験的な常識が存在していた．約4年間にわたる 12 のプロジェクトの分析により言語処理プログラムという特定分野に限定されるがその傾向を確認できた．改造率（改造規模／総規模）が小さいと改造規模当りの生産性（規模／工数）は新規開発における生産性より数倍悪い．我々のデータでは改造率が10% 程度で新規開発の5～6倍という実測値であり，改造率の増加と共に格差は減少する．

上記（1）のように生産性の向上度は意外に小さいが，これは上記の（2）が原因のひとつでもある．今後はますます改造によるソフトウェアの開発が増大する傾向にあり保守性（改造）を考慮したソフトウェアの設計技術の開発が急務である．

また，従来の生産性向上は，ソフトウェアの各ライフサイクル対応の開発用ツールやプログラミング言語の高水準化・改良などの種々の生産技術のつみ重ねで実現してきたが，一般に，これらのツール等は効果の大きい部分，あるいは技術的に容易な部分から具体化してきたので，今後は従来と同程度の生産性向上を達

表-1　SIMPLE と SYSL-2 の機能比較

項　目	SIMPLE (構造化言語)	SYSL-2 (非構造化言語)
プログラム構造	ブロック構造をもつ	同左
	データ宣言部（データセグメント）と実行部を分ける	データ宣言の位置は自由
制御構造	IF 文	同左
	CASE 文	—
	SEQUENCE グループ	—
	REPEAT グループ（DO ループの拡張）	DO グループ
	—	GO TO 文
	—	ENTRY 文（2次入口）
データ	—	ラベル変数，入口変数
	—	ラベルデータ，入口データのパラメータ授受
	データのアクセス法指定 (SET/USE 指定)	—

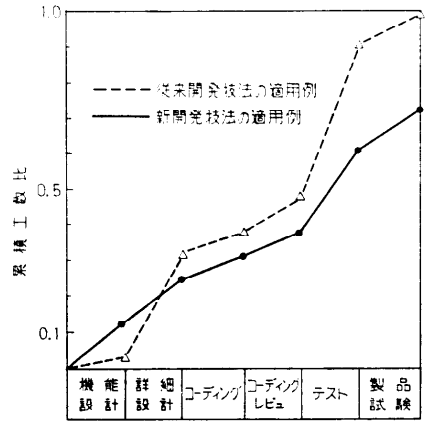


図-3　生産技法の適用例

成することさえきびしいことを認識しなければならない．

（3）　生産技術の評価例

生産技術の評価は，担当者の能力や，さらには同一人であっても評価を意識することによって結果に大きく影響するため客観的に評価するのはきわめて難しい．我々の評価事例を以下に示す．

（ア）　構造化プログラミング言語の評価

当研究所で開発した PL/I に類似のシステム記述言語 (SYSL*) と，これに構造化機能を導入した言語 (SIMPLE**) とを用いて記述実験をすることにより，

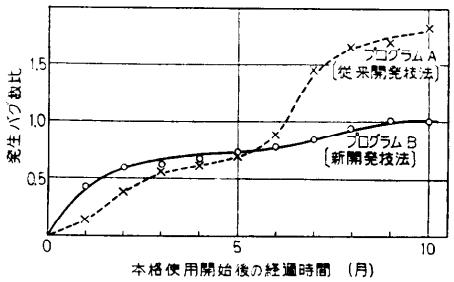
* System description Language
** The reliable Software system Implementing higher-level Programming Language

表-2　構造化言語の効果例

比較項目			比 (SIMPLE/SYSL)
プログラム特性	バグ件数		0.78
	ソースプログラム規模	宣言文	1.47
		実行文	0.82
		小計	0.95
作業時間	完了までのコーディング時間	仕様書の理解，GF の作成	0.79
		コーディング	1.05
		小計	0.90
	デバッグ時間	結果の確認，バグ原因の究明	0.71
		修正方法の検討	1.23
		修正箇所のコーディング	0.74
		小計	0.74
	合計		0.87

表-3　新，旧開発技法の主要な相違点

比較項目	旧技法	新技法
設計ドキュメント	フローチャート中心	コンパクトチャート中心
記述言語	SYSL	SIMPLE (SP 言語)
テスト	マシンテスト中心	机上テスト中心
レビュ	比較的少量	レビュ重視（各工程の3割の工数）



注1）FORTRAN コンパイラの例
2）Bの10カ月累積値を1とする

図-4　バグ（品質）の推移（例）

構造化プログラミング言語の有効性を評価した例を表-1, 2に示す．バグの修正とコーディングの工数が嵩むが作業工数全体としては約 10% の生産性の向上がはかられている．

（イ）　設計技法の評価

設計技法の評価は設計工程の終了時点では評価できず，開発工程全体が終了した段階で行わねばならない．その事例を図-3に示す．この事例は商用に提供する新規開発の FORTRAN コンパイラへの適用例であるが，両者の開発技法には設計技法以外の差もあるため設計技法単独の評価はなされていない（表-3）．しか

し，約 30% の工数削減のうち最も効果のあったのが設計技法[4]であるとの担当者の意見が多かった．

また，品質のひとつの尺度として体格使用開始後のバグの発生状況を示すと 図-4 のとおりであり，新規開発技法が品質においてもすぐれているといえる．

4. 生産性の管理

生産性の管理の究極の目標を生産性の向上と捉えるならば，管理の基本は担当者に生産性に対する強い意識を持たせることである．意識の低い担当者をどんなに上手に管理したところで生産性の向上はあまり期待できない．

担当者の意識高揚を図る具体的管理法について実施例をベースに標準作業要領と開発モデルについて述べる．

4.1 標準作業要領の制定

ソフトウェア開発における標準的な作業要領と遵守すべき設計上の規準等を収録したドキュメントである．一般に，遵守すべき事項のほかに標準を設定し，プロジェクトの特性に応じてプロジェクト・リーダが変更できる事項とに分けられる．

我々の要領は作業標準と総称されているが，開発管理規定，作業標準，作業要領から構成されている(図-5)．

作業標準制定の理由のひとつは，複数社による共同

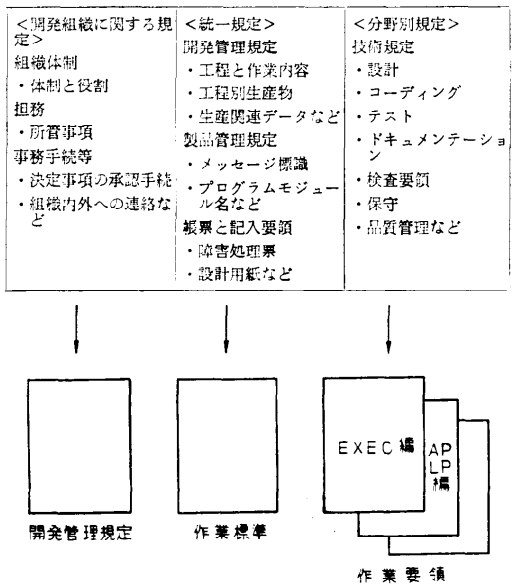


図-5 作業標準の構成と主要な記載事項

表-4 実験データの評価例[1]

担当者	コーディング時間		デバッグ時間		合計時間		
	プログラムA	B	A	B	A	B	A+B
S 1	11	20	15	4.5	26	24.5	50.5
2	76	23	29	20.5	105	43.5	148.5
3	24	4	6	7.0	30	11.0	41.0
4	58	6	57	26.0	115	32.0	147.0
5	66	5	15	8.0	81	13.0	94.0
6	111	50	85	8.0	196	58.0	254.0
7	88	16	170	12.5	258	28.5	286.5
8	19	2	31	2.0	50	4.0	54.0
9	12	2	20	3.0	32	5.0	37.0
10	60	16	27	2.0	87	18.0	105.0
11	7	2	23	3.5	30	5.5	35.5
12	24	10	30	1.0	54	11.0	65.0
最大，最小比	1：16	1：25	1：28	1：26	1：9	1：15	1：8

開発体制であるため，社間の作業差を吸収することであったが，最近では改良を重ねて一社のみの開発においても十分に有用であるものになっている．

ソフトウェア開発における担当者による格差は，生産性で8倍[1]，品質（バグ数）で10倍[2]という報告がある(表-4)．しかし我々の例ではそれぞれ3倍程度であり，均質なソフトウェアの生産に対する作業標準の効果は大きいと評価している．

4.2 開発モデル

プロジェクトの開始にあたり開発計画書の審議・承認の時点で開発モデルの作成を義務づけるのが効果的である．

開発モデルの作成にはデータの蓄積が基礎となるので通常の生産性関連データの収集が重要であり，これらのデータをデータベース化することが必須となる．モデルの作成を通じて，どのようなデータが不足しており，そのためにはどのような測定をすべきか，などの問題意識が生まれる．

以下にモデルの作成例を，手順的に説明する．

ステップ 1 開発対象システムの規模の推定

類似システムからの推定など現時点ではプロジェクト・リーダの経験に負うところが大きい．

ステップ 2 開発期間の算定[5]

これまでのデータ分析結果等に基づき大筋を設定しプロジェクトの特殊性などを考慮して調整する．我々の実測例を図-6に示す．

ステップ 3 工程別工数配分

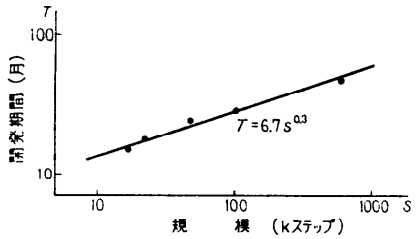


図-6　開発期間と規模（10～1,000 KS）との関係

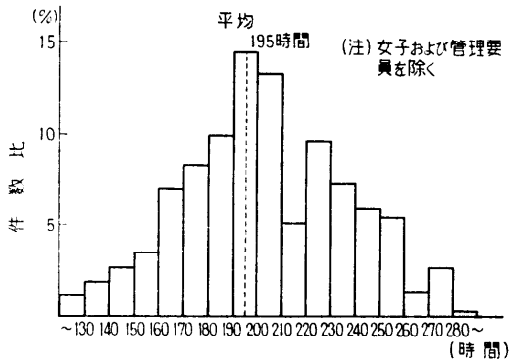


図-7　担当者の1人月の実稼動分布

既存のデータ分析と利用可能な生産技術を勘案して配分する．我々のデータ分析によると既発表のデータに比較しテスト工数の比率が大きい（表-5）．

ステップ 4　稼動工数の見積り

開発モデルで使用する工数は実際の稼動可能な工数であり前述した標準工数ではない（評価用と計画用を区別する）．担当者の稼動工数は平均値と最大値とでは約4割，最小と最大値では約2倍の格差が生じている（図-7）．

また，年間を通じての月別の工数の変動幅は中央値から ±10% 以内である（図-8）．

稼動工数の変動傾向としては，計画値を超過するケースと計画値未満のケースの比率は7対3で超過のケースが大きく（図-9），変動要因としては稼動推定の誤りと計画変更とは同程度である（図-10）．

ステップ 5　プロジェクトの要員構成

たとえば，（5人×2月）と（1人×10月）の工数は等価ではない[3]．したがって，工程別の総工数が推定できても要員×期間が一意に定まらない．要員構成については手軽に実験することも難しいので，多数のプロジェクトの開発データを蓄積する必要がある．一方，現実には先に要員数の枠が決められ，稼動可能な要員の範囲で，総工数見合いで期間を修正するなどの

表-5　システム開発における工数比[9]

	開発システム例	開発年時	設計	コーディング	テスト	記述言語規模
通研	オペレーティングシステム (A)	1975	28	10	62	高級言語，アセンブラ 340 KS
	〃 (B)	1976	29	11	60	高級言語，アセンブラ 370 KS
	言語処理プログラム (A)	1977	29	20	51	高級言語，アセンブラ 900 KS
	〃 (B)	1977	35	9	56	高級言語 110 KS
	〃 (C)	1978	44	26	30	高級言語 50 KS
参考	C. Weissman[8]	1973	40	20	40	不　明
	M. Zelkowitz[9]	1978	35	20	45	不　明

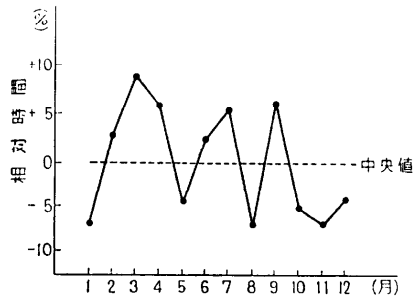
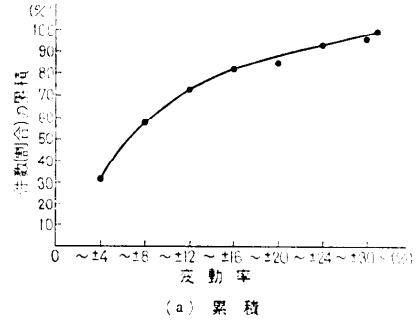
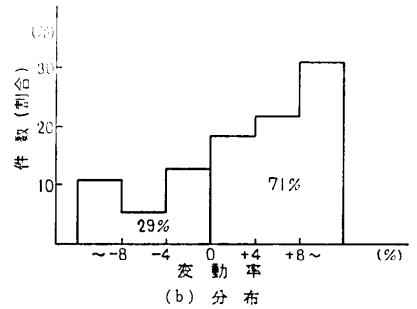


図-8　実稼動時間の年間変動

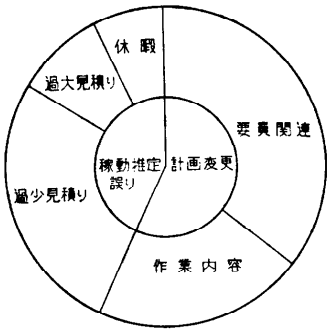


(a) 累積



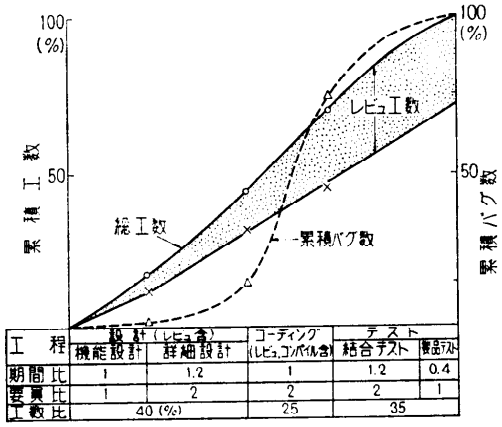
(b) 分布

図-9　工数の変動傾向



（備考） 各原因の群細内容は以下のとおり
要員関連：不足／余剰
作業内容：繰上げ／繰延べ，作業追加／削除
過少見積り：不測作業の出現，勤務時間の延長
過大見積り：作業の立上り不足，予想外に順調

図-10 工数予測値ズレの原因別割合



工程	設計（レビュ含）		コーディング（レビュ，コンパイル含）	テスト	
	機能設計	詳細設計		結合テスト	製品テスト
期間比	1	1.2	1	1.2	0.4
要員比	1	2	2	2	1
工数比	40 (%)		25	35	

図-11 開発モデルの例（新規開発の場合）

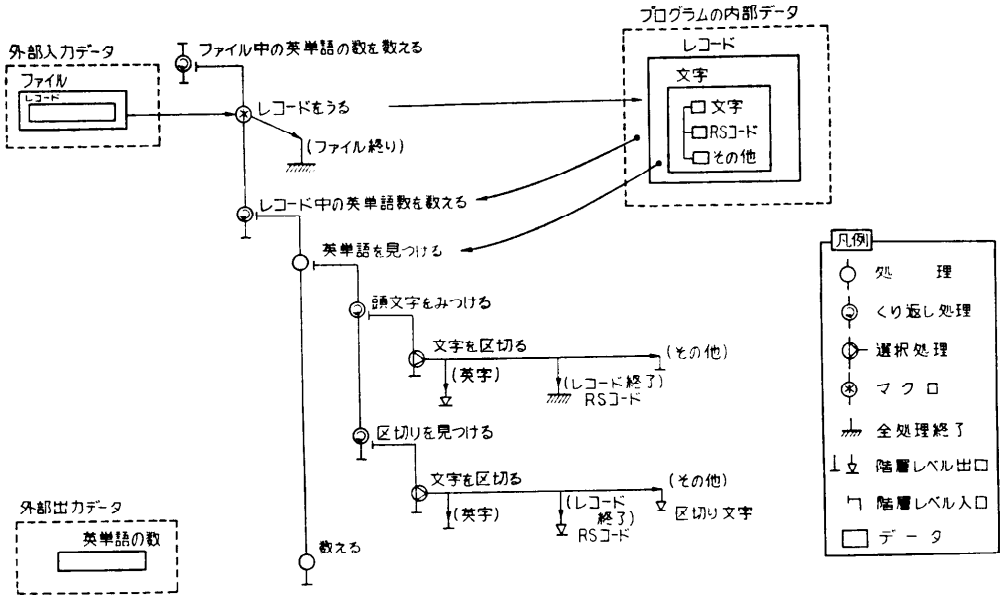


図-12 HCP チャートの例

調整をしているケースが多い．

以上が開発モデルの大略の手順であり，累積バグ数の予測も加えた実際の例を**図-11**に示す．

モデルを作成してみると種々のデータの不足と技術が未確立であることが再認識させられるが，特に上記のステップ1とステップ5については今後の技術的課題である．

5. むすび

生産性の評価と管理に関する問題点を整理し，それらに現実的な前提を置いた上で若干の評価事例を紹介した．

一方，生産性の向上を究極の目標とする管理においては，担当者の生産性に対する意識の高揚が基本であるとの認識に立ち，このためには開発計画の策定時点で具体的な開発モデルを作成する作業を義務づけることが効果的であるという経験に基づきその概略の手順について述べた．

今後取り組むべき主要な技術的課題としては

（1） ソフトウェアの生産量の規定とその測定技術

の確立

（2） 開発工程別の工数配分とそれを実現する要員と期間のトレード・オフ手法

（3） ソフトウェアの開発規模の見積り技術

などが考えられる．

最後に，生産量の尺度に対する私的な提案を述べる．2章でも述べたように生産量を〔単位工数当りのスラップ数〕，あるいはその逆数などで表現することには疑問が多い．

その代替案のひとつとして「機能」を単位とする測定法を提案する．それは我々が現在広く使用している，従来のフローチャートに代る設計ドキュメントのひとつであるコンパクトチャート (CP)[4] を用いる方法である．CP では処理機能のある「まとまり」を1つの記号で表示しようとしているので，その図記号の個数で機能量を表現することとし，さらに従来のステップ数に代えてこの機能量を生産量として用いることができないかと考えている（**図-12**）．

一方，ほかの観点からの尺度としては，今後のソフトウェア生産の実態に即した尺度として，既存のソフトウェアを流用することの効果が表現できるうまい尺度がないだろうか．このような尺度が考案できればソフトウェアの利活用，あるいは流通がより一層活発になることが期待される．

参 考 文 献

1) Halrod Sackman: Computers, System Science, and Evolving Society—The Challenge of Man-Machine Digital System John Willy & Sons, Inc, 1967 (竹中監修マン・マシン・ディジタル・システム).
2) Boehm, B. W.: The High Cost of Software, Software World, Vol. 6, No. 1, pp. 2-10.
3) Brooks, F. P.: The Mythcal Man-Month, Addison-Wesley, 1975 (山内訳，ソフトウェア開発の神話).
4) 花田，佐藤，松本，長野：コンパクトチャートを用いたプログラム設計法，情報処理学会論文誌，Vol. 22, No. 1，掲載予定.
5) Waltson, C. E. and Felix, C. P.: A Method of Programming Measurement and Estimation, IBM Syst. J., Vol. 16, No. 1, pp. 54-73.
6) 花田，山本，佐藤，岡田：プログラム生産技術の体系と評価，通研報，Vol. 28, No. 12 (1979).
7) 花田，山本，佐藤，岡田：プログラム生産技術の体系と評価，第3回 DIPS シンポジウム予稿集，pp. 101 (1979).
8) クラーク・ワイスマン：最近のソフトウェア技術の展望と将来，情報処理，Vol. 14, No. 10, pp. 739-745 (1973).
9) Zelkowitz, M. V.: Perspectives on Software Engineering: Computing Surveys, Vol.10, No. 2, pp. 197-215 (1978).

（昭和55年7月22日受付）