上武大学ビジネス情報学部紀要
第 11 巻第 2 号（2012 年 12 月）

〈論　文〉

ソフトウエア開発マネジメントに関する考察
―ソフトウエア開発のコスト／スケジュールマネジメント編―

A Study on Management of Software Development
―Phase2 : Cost and Schedule Management of Software Development―

吉 崎 浩 二
YOSHIZAKI Kouji

現在の高度情報化社会ではあらゆる分野にコンピュータが適用され、今や社会のインフラになっている。したがって、ひとたび故障すると大きな社会問題となる危険性が高い。規模的にハードウエアに比してソフトウエアの占める割合が急増しているのである。

また、ソフトウエアの開発コストはハードウエアのそれと比べて膨大なものになりつつあり、開発工程もハードウエアのそれと比して複雑で不安定なものになっている。

したがって、ソフトウエア開発マネジメントの必要性は高まるばかりであるが、ソフトウエア開発の品質、コスト、開発期間そして満足度などに関して、十分にマネジメントされているとは言い難い。むしろますます大きな問題を抱えつつある。

その大きな要因は「ソフトウエアを開発する者は開発状況を見せる工夫や努力が十分とはいえず、ソフトウエア開発を管理する者は見ようとする努力と工夫が十分ではない。」ところに課題があるといえる。

なかでも、ソフトウエア開発の品質マネジメント、コストマネジメント、スケジュールマネジメントが重要であるが、ここでは特にソフトウエア開発のコストとスケジュールのマネジメントについて考察する。

キーワード　ソフトウエア開発、マネジメント、ソフトウエア開発コスト、ソフトウエア開発期

間、コストマネジメント、スケジュールマネジメント、ソフトウエアコストモデル、
PERT技法、見せる努力と工夫、見る努力と工夫

1　はじめに

現在の高度情報化社会ではあらゆる分野にコンピュータが適用され、今や社会のインフラになっている。ひとたび故障すると大きな社会問題となる危険性が高い。

また、規模的にもハードウエアに比してソフトウエアの占める割合が急増している。ハードウエア自体はLSI化が進み品質はますます向上している中、コンピュータを動かすソフトウエアが高品質でなければならない。

したがって、ソフトウエアシステムの社会的責任が拡大しつつある。その背景にはシステムの大規模化と複雑化がある。直接的または間接的にもその影響度は高くなりつつあり、かつ広範囲にわたる傾向が強い。

このように、ソフトウエアマネジメントの必要性は高まるばかりであるにもかかわらず、ソフトウエア開発の品質、コスト、開発期間そして満足度などに関して、十分にマネジメントされているとは言い難い。むしろますます大きな問題を抱えつつある。

その大きな要因はソフトウエア開発のマネジメント教育が十分いきわたっていない事にある。また、「ソフトウエアを開発する者は開発状況を見せる工夫や努力が十分といえず、ソフトウエア開発を管理する者は見ようとする工夫と努力が十分ではない。」ところに課題があるといえる。

ソフトウエア開発をマネジメントすべきことは品質、コスト、スケジュールと多々あるが、ここではソフトウエア開発のコストマネジメントとスケジュールマネジメントについて考察する。

第2章では、標準値法による工数予測について考察する。過去の実績データから生産性標準値を設定しておき、開発規模（ステップ数）をこの基準値で割ることによって、開発工数を求めるものである。最もポピュラーな方法であるが、標準値に対して、使用言語、開発規模（大／中／小規模）、システムの特性（業種、オンライン／バッチ、集中分散、パッケイジ使用など）などに依存するので、特性に応じて分類をしておけばより実用的になる。

第3章のCOCOMOモデルは、米国TRW社のソフトウエア工学研究者のB. W. Boehmが1981年に発表したソフトウエア開発工数と期間に関するモデルである。63のプロジェクト

データを分析した結果、得られたモデルでもある。

第4章のFunction Point法は、1979年にIBM社のA. ALBRECHTによって開発された方法論であり、米国だけでなく日本においてもかなり普及している。「ファンクションポイント」法は、規模と工数の予測の問題に対して、ソフトウエア内部で使用されるデータからソフトウエアの機能を規定する方法論である。

第5章のPUTNAMモデルは1977年にPUTNAMがNorden/Rayleighのマンパワー理論式を活用して提案したモデルである。別名、3者の頭文字をとって“PNRモデル”とも云う。PUTNAMモデルの特徴は、他のモデルが工数予測を主体としているのに対して、最適な要員配置を示す“マンパワー曲線”をモデル化した点にある。

第6章でのソフトウエアのプロセスモデルは大きく分けて、下記の3つのモデルがある。

①ウォータファール・モデル

②スパイラルモデル

③プロトタイピングモデル

それぞれに特徴があり、スケジュール管理をするにあたっては、開発する情報システムに適合したプロセスモデルを選択し、デザインする必要がある。

第7章のWBSはプロジェクト全体を成果物を生み出す作業単位に分割することにある。大日程計画ではWBSの最小レベルである「ワークパッケージ」まで分割し、中日程計画や小日程計画では成果物の有無を考慮しない作業単位である「アクティビティ」まで分割する。

第8章のPERT技法は、製品開発の日程計画を立てる時に用いられる技法である。複雑な仕事の処理順序の関係をネットワークの形でアローダイアグラムによって表現し、プロジェクトの開始から終了に至るまでの仕事の処理時間に余裕のない経路（クリティカルパス）を明確にして、予定工期までにプロジェクトを完成できるかどうかの計画の実行可能性を検討し、管理の重点を明らかにする手法である。

第9章の構成管理システムによるソフトウエア開発マネジメントは構成管理システムを単に、ソフトウエア資産の保管と変更管理の活用にとどめるのではなく、構成管理を必要とするソフトウエア開発プロセス全範囲において、構成管理システムに管理されるデータならびに構成管理をアクセスするプロセス〔過程〕自体の情報を意図的に取り入れ、効率的に活用し、ソフトウエア開発プロセス自体を目視化し、分析可能にすることによって、ソフトウエア開発プロセスの改善につながる手法を提言するものである。

以下、順次、詳細に考察してゆく。

2　ソフトウエア開発に必要な工数と期間の予測方法―標準値法による予測―

ソフトウエア開発にかかるコストを予測（見積もり）する方法には大きく分けて、開発プログラムのステップ数からコストを予測する方法と開発するシステムの機能を予測し、その機能数からコストを予測する方法がある。

前者には標準基準法とCOCOMOの方法があり、後者にはFUNCTION POINT法がある。以降、詳細を考察する。

2．1　標準値法による予測

過去の実績データより生産性標準値を設定しておき、開発規模（ステップ数）をこの基準値で割ることによって、開発工数を求める。

最もポピュラーな方法であるが、標準値に対して、使用言語、開発規模（大／中／小・規模）、システムの特性（業種、オンライン／バッチ、集中分散、パッケイジ使用など）の別といった分類をしておけば、より実用的になる。

この手法の系列としては古典的なJ. P. ARONの計算式が有名である [1] [31]。

総開発工数

＝A／B＊(1＋C)＊D

ここで

A：開発ステップ数

B：生産性（標準値）

C：間接要員比率

D：余裕率

Aronのモデルの生産性（B）テーブルの例

アセンブリ言語ベースである。高級言語の場合はこの2倍の基準値とする。

図2．1　Aronの生産性テーブル

難易度	規　　模		
	小(1年以内)	中(2年以内)	大(2年以上)
容　易	20	500	10000
普　通	10	250	5000
難しい	5	125	1500
	Steps／人日	Steps／人月	Steps／人年

開発工数に大きな影響を与える要因（難易度）の例は下記のとおりである。

①開発ステップ数

②作業行程（全行程か、部分工程か）

③技術的難易度

④プログラムの種類

⑤開発する人の経験や知識の程度

⑥プログラミング言語

⑦開発技術（開発環境、ツール）

⑧コンピュータの規模

⑨使用形態（オンライン／バッチ）

等である。

2．2　標準基準法による総工数予測の手順

〈ステップ1〉プログラム構成の決定

プログラム構成を出来るだけ細かく、明確にする。

〈ステップ2〉プログラム規模の決定

プログラム規模（ステップ数）の予測は、次の手順で行う。

（ｉ）3−4人のメンバーが構成プログラムのそれぞれについて、次の3つの値を推定する。

a＝可能な最小値

m＝最尤値

b＝考えられる最大値

（ii）各構成プログラムについて、次の値を計算する。

$$Ei=(avg.a+4avg.m+avg.b)/6$$

ただし、Ei：第iプログラムの予測値

avg.a ：aの平均値

avg.b ：bの平均値

avg.m：mの平均値

（iii）総数Eは、次のように計算できる。

$$E=\Sigma Ei$$

〈ステップ3〉開発工数の見積り

ステップ2で推定したプログラム規模を基準生産性データで割ることにより、必要な開

発工数を予測する。

実際の開発に必要な工数は、開発環境や難易度の影響を受ける。そこで、工数や費用を増大させる要因によって補正係数を設定して掛けるようにすべきである。

例

開発範囲が要求分析からテストまでの、基準生産性データを利用する計算例を示す。

高級言語使用の例である。

開発工数

＝(補正係数)＊(予測プログラム規模)／(基準生産性)

K＝1.3＊5000／20＝325人日

ここで

補正係数：1.3（難易度係数）

5000：予想ステップ数

20：ステップ／人日：標準生産性

(Aronのアセンブラー言語基準＊2倍：10＊2)

〈ステップ4〉各工程の見積り

ステップ3で計算された総工数を基にして、各工程毎の工数を計算する。

各工程の開発工数＝(総工数)＊(各工程の工数比)／100とする。

例えば、詳細設計の工数比が25％である時、

詳細設計の工数＝総工数＊25／100

＝325＊25／100≒82人日

で求められる。

2．3　工数比／期間比の例

図2．2に工数比／期間比の例をしめす。

図2．2　工数比／期間比の例

	要求定義	外部設計	内部設計	プログラム設計・作成	総合テスト	システムテスト
工数比	10	15	25	25	10	15
期間比	15	20	20	20	10	15

2．4　工程別適用範囲の例

全開発工程のうち、

○基本計画

○実行計画

○システム計画

は個別見積もりとする。

○ソフトウエア基本計画

○ソフトウエア詳細設計

○コーディング

○単体テスト

○結合テスト

はステップ数に基づく見積もりの範囲とする。

○総合テスト

○現地調整／ユーザ引渡し

○保守

は個別見積りとする。

2．5　（新規作成分、再利用分、標準部品）考慮の例

開発工数

＝〔ステップ数／ステップ生産性〕＊難易度係数

＝｛新規分　S1／P

＋再利用分S2／P＊β

＋標準部品S3／P＊γ｝＊難易度係数

手直し係数の例

再利用分　　β＝0.4

標準部品係数　　γ＝0.1

2．6　例（難易度係数）

図2．3に難易度項目得点表の例を示す。

図2．3　難易度項目得点表の例

	項　　目	A	B	C	D
1	技術的新規性	3	2	1	0
2	性能面の特別要求	3	2	1	0
3	障害対策の特別要求	3	2	1	0
4	規模大	3	2	1	0
5	開発期間短縮要求	3	2	1	0
6	品質面の特別要求	3	2	1	0

図2．4に難易度係数の例をしめす。

図2．4　難易度係数の例

	難易度項目得点	難易度係数
1	0－4	0.9
2	5－9	1.0
3	10－14	1.3
4	15－19	1.5
5	20以上	1.7

例（難易度係数の実習）

難易度項目得点表の例

	項　　目	A	B	C	D
1	技術的新規性	3	2	1	0
2	性能面の特別要求	3	2	1	0
3	障害対策の特別要求	3	2	1	0
4	規模大	3	2	1	0
5	開発期間短縮要求あり	3	2	1	0
6	品質面の特別要求	3	2	1	0

とすると総難易度項目得点＝2＋3＋3＋2＋2＋2＝14である。

したがって難易度係数は図2．4より1.3となる

例（新規作成分、再利用分、標準部品の実習）

ただし、手直し係数は

再利用分　　$\beta=0.4$

標準部品係数　　$\gamma=0.1$

とする。また、

S1＝10k、S2＝20k、S3＝20k（C言語）

P＝500ステップ／人月である時、

開発工数

＝〔ステップ数／ステップ生産性〕＊難易度係数

＝(10*1000/500＋20*1000/500*0.4＋20*1000/500*0.1)*1.3

＝40*1.3

＝52人月

3　ソフトウエア開発の必要な工数／期間の予測方法

―COCOMOモデルによる予測―

3．1　はじめに

COCOMOモデルは、米国TRW社のソフトウエア工学研究者のB. W. Boehmが1981年に発表したソフトウエア開発工数と期間に関するモデルである。63のプロジェクトデータを分析した結果、得られたモデルでもある[2][6][7]。

DOTYモデル[4]が比較的小規模なシステム開発に適したモデルに対して、COCOMOモデルは中規模ないしは大規模なシステム開発に適したモデルといわれている。また、PUTNAMモデルはこれに対して、大規模または超大規模なシステムに適したモデルといわれている[3][5]。

COCOMOとは、「Constructive Cost Model」の略で、「積み上げ方式によるコスト・モデル」という意味である。

3．2　COCOMOモデルの特徴

COCOMOモデルには次に挙げる3つの特徴があり、広く認められた手法である。

その1）基本見積りから詳細見積もりまで可能である。

基本COCOMOモデル、中間COCOMOモデル、詳細COCOMOモデルの3つのモデルにより、基本見積りから詳細見積りまでを行うことが出来る。

その2）多様な開発形態に適用可能（3つのモード）である。

組織モード、半組み込みモード、制約組み込みモードの3つもモードがある。

その3）ソフトウエア開発の生産性を考慮（15のコスト・ドライバー）している。

COCOMOモデルでは、ソフトウエア開発の生産性を考慮するために、15のコスト・ドライブを考えている。

3．3　3つのモード（開発形態）

①組織モード（Organic Mode）

比較的小さなチームで、熟知している業務に関する自社ソフトウエアの開発を行う場合を「組織モード（Organic Mode）という。50KLを超えるプロジェクトはまれである。企業における情報システム部門のメインテナンス（システムの改良改善）開発などがこれに相当する。

〈特徴〉

- ✧ システムに関する理解が深いために、お互いのコミュニケーションのオーバヘッドが少なくてすむ。
- ✧ システムと利用者の要求との食い違いの調整が比較的容易である。
- ✧ 開発環境が一定しており、新しいハードウエアや操作手順のための開発が少なくて済む。
- ✧ 納期時期への制約が比較的ゆるやかである。
- ✧ サイズが比較的小さい（50KDSI程度）

②半組み込みモード（Semidetached Mode）

「半組込モード」とは、「組織モード」と次の「制約組み込みモード」の特徴を半々に持つ、または、特徴が中間的な場合を云う。

〈特徴〉

- ✧ チームメンバーはシステムに対して中間的なレベルの経験を持つ。
- ✧ チームメンバーの中には、経験の深い人から、浅い人まで巾広くいる。
- ✧ チームメンバーはシステムのある部分に関しては詳しいが、他の部分については知らない。
- ✧ 半組み込みモードのプロジェクトは、300KDSIぐらいまでの範囲に達する。

〈例〉

幾つかの厳密なインターフェース（受発注処理）と幾つかの自由度の高いインタフェース（オペレータ・メッセイジ）などを必要とするトランザクション処理システムが代表的な例である。

③制約組み込みモード（Embeded Mode）

大きな制約のもとで、ソフトウエア開発を行う場合は「制約組み込みモード（Embeded Mode）という。

〈特徴〉

- ✧ プログラマーは多くの変更を吸収しなければならない。
- ✧ 開発が遅れると時代遅れとなるために、開発が急がれる。

〈例〉

電子為替決済システムや空調制御システムのような、ハードウエア、ソフトウエア、規則などの複雑な要因が強く絡み合った状況下で動作するシステム開発が、これにあたる。

3．4　用語の定義

〈ステップ数〉

- ✧ ステップ数はKDSIという単位であらわす。
- ✧ KDSIとは「Kilo Delivered Source Instruction」を意味する。
- ✧ 「Delivered」とは、「利用者に供給される」という意味で、テスト・ドライバーなどの支援ソフトウエアを除くことを意味する。
- ✧ 「Source Instruction」は、コメント行は除く。

〈見積りの工数を含めるフェーズと作業範囲〉

- ✧ フェーズは設計からテスト工程を含む
- ✧ ユーザ教育などは含まない。
- ✧ プロジェクト管理者やライブラリアンのコストは含む。

〈人・月〉

- ✧ COCOMOモデルでの人・月は19人・日（152時間）を意味する。
- ✧ 1年→12ヶ月
  - ➢ 1ヶ月→実働19日
  - ➢ 1　日→実働8時間

を意味する。

3．5　モデルの種類

作業内容による分類。

①開発作業用COCOMO：COCOMOといえば主にこれを指す（マネジメントやドキュメンテーションを含む。設計からテストまでの範囲）。

②保守作業用COCOMO

③コンバージョン用COCOMO

三つのバージョン

基本COCOMO：見積り規模のみで予測する。

中間COCOMO：見積り規模＋15の影響要因で見積もる。

詳細COCOMO：上記＋工程毎に影響要因を調整する。

一般には、簡便でしかも信頼性の高い予測値が得られる中間COCOMOモデルが良く使われる。

3．6　COCOMOモデルの見積り式

①工数見積り式

$$MM = a(KDSL)^b$$

②開発期間の見積り式

$$T_{DEV} = c(MM)^d$$

中間COCOMOでは

$$MM = a(KDSL)^b \cdot \Pi A$$

Aは15の影響要因を表わす

図3．1　中間COCOMOモデルの見積り式の係数表

	a	b	c	d
組織	3.2	1.05	2.5	0.38
半組込	3.0	1.12	2.5	0.35
組込み	2.8	1.20	2.5	0.32

3．7　サイズと工数の関係

図3．2に開発規模（ステップ数）と工数（人月）の関係を示す。

同じ開発規模（ステップ数）でも開発形態によって工数は大幅に変わることがわかる。また、開発規模（ステップ数）が増大するにつれて工数の増大する率が大きくなることがわかる。

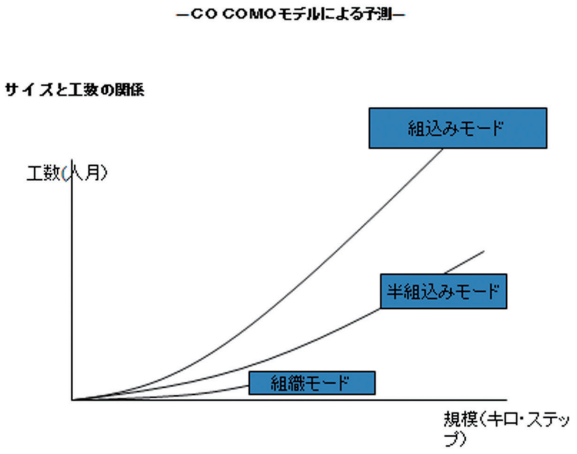


図3．2　サイズと工数の関係

3．8　コスト・ドライバーについて

ソフトウエア開発の生産性を考慮するために、COCOMOモデルでは、15の

コスト・ドライバーを使用することが出来る。コスト・ドライバーを考慮することによって、ソフトウエア毎に異なる生産性を見積りに反映させることが出来る。

各コスト・ドライバーは、多くは［very low］、［low］、［nominal］、［high］、［very high］、［extra high］の6つのランクをとる。

コスト・ドライバーは次の4つのカテゴリーに属する15の要因からなる。

（1）ソフトウエア製品に関する属性

RELY：ソフトウエアに要求される信頼性

DATA：データベースのサイズ

CPLX：ソフトウエア製品の複雑性

（2）計算機に関する属性

TIME：システムに課せられた実行時間の制約

STOR：主記憶の制約

VIRT：OS、ハードの変更頻度

TURN：プログラム開発時のターンアラウンドタイム

（3）要員に関する属性

ACAP：アナリストの能力

AEXP：同様なアプリケーションの経験度合い

PCAP：プログラマーの能力

VEXP：OS、ハードの経験度合い

LEXP：使用されるプログラム言語の経験度合い

（4）プロジェクトに関する属性

MODP：最近のソフトウエア開発技法の使用度合い

TOOL：ソフトウエア開発ツールの使用度合い

SCED：要求される開発期間の制約

3．9　COCOMOモデルの適用例

①　物流システムの例に中間COCOMOモデルを適用する例

（1）規模の見積り

COBOLで100kステップ

（2）モードの選定

半組み込みモード

(3) 工数の見積り

＝$3.0 * 100^{1.12}$

≒521（人月）

(4) コスト・ドライバーの見積り

ΠA＝1.11*1.06＊1.00…・・＝1.17とする。

(5) 調整後の工数

＝521*1.17≒610（人月）

(6) 期間の見積り

＝$2.5 * 610^{0.35}$≒24（月）

(7) 要員数と生産性の計算

平均要員数＝工数／期間＝610／24≒26人

生産性＝100＊1000／610＝163（ステップ数／人月）

② 測量システムの例に中間COCOMOモデルを適用する例

(1) 規模の見積り

C言語で10kステップ

(2) モードの選定

組み込みモード

(3) 工数の見積り

＝$2.8 * 10^{1.20}$

≒45（人月）

(4) コスト・ドライバーの見積り

ΠA＝1.11*1.06＊1.00…・・＝1.17とする。

(5) 調整後の工数

＝45*1.17＝53（人月）

(6) 期間の見積り

＝$2.5 * 53^{0.32}$≒9（月）

(7) 要員数と生産性の計算（演習1）

平均要員数＝工数／期間＝53/9≒6人

生産性＝10*1000/53≒189（ステップ数／人月）

③ 制御システムの例に中間COCOMOモデルを適用する例

（1）規模の見積り

C言語で50kステップ

（2）モードの選定

組み込みモード

（3）工数の見積り

$=2.8*50^{1.20}\fallingdotseq306$（人月）

（4）コスト・ドライバーの見積り

ΠA＝1.11*1.06＊1.00…・・＝1.17とする。

（5）調整後の工数

＝306*1.17≒358（人月）

（6）期間の見積り

$=2.50*358^{0.32}\fallingdotseq17$（月）

（7）要員数と生産性の計算

平均要員数＝工数／期間＝358/17≒21人

生産性＝50*1000/358≒140（ステップ数／人月）

3．10　Dotyの計算式 [4]

COCOMOモデルは中規模ないしは大規模なシステム開発に適してモデルといわれているのに対して、DOTYモデルは比較的小規模なシステム開発に適したモデルである。

演算式は次のようになる。

$$MM=a\,I^{b}$$

ここで、MM：総開発工数（人月、分析から総合テストまで））

I：プログラム規模（Kステップ）

a，b：定数（図3．3）

図3．3　Dotyモデルの係数表

アプリケーション	オブジェクトコード		ソースコード	
	a	b	a	b
全体（総平均）	4.790	0.991	5.258	1.057
指揮管制	4.573	1.228	4.089	1.263
科学技術計算	4.495	1.068	7.054	1.019
事務処理	2.895	0.784	4.495	0.781
ユーティリティ	12.039	0.719	10.078	0.811

開発期間（D）の積算アルゴリズムは次の式のようになる。

$$D=1{,}000I/(99.25+I^{0.667})$$

ここで　D：開発期間（月）

I：プログラム規模（オブジェクトコード　K語）

4　ソフトウエア開発に必要な開発工数／期間の予測方法―Function Point法―

4．1　はじめに

この方法は、1979年にIBM社のA. ALBRECHTによって開発された方法論であり、米国でかなり普及している[8][9]。

「ファンクションポイント」は、規模と工数の予測の問題に対して、ソフトウエア内部で使用されるデータからソフトウエアの機能を規定する方法論である。この方法論によれば、ソフトウエアの機能は、基本的には、ファンクションポイントとして、入力、出力、マスターファイルおよび照会の数などの加重平均から求めることが出来る。

フンクションポイントに基づく作業工数の推定は、早期の基本的な要求事項から、ファンクションポイントを比較的、容易に求めることが出来る。

この点において、プログラムステップ数SLOC（Source Lines of Code）による推定よりも一層有効であると考えられる。

4．2　フンクションポイント法の特徴

フンクションポイント法の特徴は、前述したように、使用言語、作成者の技術、開発環境、4GLやアプリケーション・パッケイジを用いた場合などといったシステムの作り方とは直接関ること無く、一貫した尺度でシステムの価値を評価するところにある。

COCOMOモデルやPUTNAMモデルや標準値法は、あくまでもソフトウエアを開発する立場からの原価評価であるが、このファンクションポイント法は、ソフトウエアまたはシステムの機能価値の評価をするものである。これにより、旧来からのプログラム数やソース・ステップ数によるアプローチの限界を打開しょうとしている。

したがって、利用する側の立場、つまり顧客から見れば、システムの機能に対して代価を支払いう方が合理的で納得しやすい面がある。この意味で、現在、再度、ソフトウエアの見積り手法として注目を集めつつある。

4．3　基本的な算定手順

基本的な算定手順は下記のとおりである。

（手順1）システムの提供する機能（入出力、マスターファイル、インターフェースなど）の数を把握した上で、個々の機能を分類（単純、平均、複雑）して、重み係数を乗じ、全体を合計することで、調整前の初期ファンクションポイントを求める。

（手順2）対象システムの特性に応じて、調整用のシステム特性係数を求める。

（手順3）初期ファンクションポイントにシステム特性係数を乗じて、最終値を得る。

4．4　5つのファンクションについて

システムの機能は下記の5つから構成される。

①ユーザからの入力処理機能

- 画面入力処理
- 帳票入力処理
- ファイルメインテナンス（ファイル入力）処理

②ユーザに対する出力処理機能

- 画面出力処理
- 帳票出力処理
- ファイルメインテナンス（ファイル出力）処理

③ユーザからの照会処理機能

④内部論理ファイル機能（保守の責任を伴う）

⑤外部インターフェースファイル機能（参照のみ）

4．5　ファンクションポイント

図4．1に示すように、5つの機能それぞれに対して、数と複雑度を評価して、ファンクション数と重み付け係数を評価する。

図4．1　ファンクション数＊重み付け

複雑度 / 機　能	ファンクション数＊重み付け係数			計
	単純	平均	複雑	
外部入力処理(画面／帳票／ファイル)	×3＝	×4＝	×6＝	
外部出力処理(画面／帳票／ファイル)	×4＝	×5＝	×7＝	
内部論理ファイル	×7＝	×10＝	×15＝	
外部インターフェースファイル	×5＝	×7＝	×10＝	
外部照会処理	×3＝	×4＝	×6＝	
	調整前ファンクションポイント(A)			

複雑度は、扱うデータ項目数によって、分類する。

たとえば、入出力の平均は“10〈データ項目数＜15”

内部論理ファイルの平均は“30＜データ項目数＜50”

等の判定基準を設ける。

この表の計算によって、5つの機能のファンクション・ポイント（調整前）を求める。

4．6　システムの特性係数の求め方

下表に示すように、14個のシステム特性の評価をする。

この14の特性各々に対して、0から5までのポイントを評価する。

0：該当しない

1：ほとんど無い

2：適度にあり

3：平均的

4：重要

5：特に重要

この結果、システム特性値の合計 ＝ Bとする時、

システム特性係数C＝0.65＋(0.01＊B)を求める。

図4．2　システム特性係数の算定表

ID	観　　点	特性値
1	データ通信機能あり	
2	分散処理あるか	
3	レスポンスやスループットの制約あるか	
4	コンピュータ資源の制約が厳しいか	
5	トランザクション量はどうか	
6	オンラインデータ入力はあるか	
7	エンドユーザ処理の充実(画面処理の複雑性)	
8	DBのオンライン更新あるか	
9	処理の複雑性はどうか	
10	流用性を考慮して開発する必要が高いか	
11	導入移行性を考慮する必要があるか	
12	オペラーションのしやすさを考慮しているか	
13	複数個所での使用を考慮して開発するか	
14	変更容易性を配慮するか	
	システム特性値の合計(調整用)B	

4．7　全体のファンクションポイントの計算

調整前のファンクションポイントとシステム特性を使って、全体のファンクションポイントを次の式で求める。

全体のファンクションポイント

＝5つのファンクションポイントの合計＊{0.65＋(調整ポイント＊0.01)}

＝A＊C

4．8　見積りへの適用

見積りに適用するにあたっては、

①新規か／改定か

②使用言語は、アセンブラーか高級言語か

③パッケイジ適用か／否か

等によって、ファンクションポイントあたりの作業工数の傾向にバラツキが生じるため、見積りに適用する場合は、実績データ収集と分析及び基準値設定にこうした面での考慮が必要である。

4．9　ステップ数による評価とF.Pによる評価について

ステップ数による評価にはいくつかの課題がある。そのひとつは「Physical LineとLogical Line」の問題である。

2番目は「Lineのタイプ」によって評価が異なるところである。

たとえば、

- Executable Lines
- Data Definition
- Comments
- Blank Lines

等によって評価が異なる。

3番目は

「Reusable Code」の問題である。

4番目は

「言語の問題」である。

Cobol、C、Basic、Adaなどによって、評価が異なる。

下図は、ファンクションポイントによる評価を基準にした場合の言語の違いよる生産性

の違いを示す例である。Jonesによると[32]

図4．3　言語の違いよる生産性の違いを示す例

言語	ソースコード数/FP
アセンブラー	320
C	128
BASIC	128
FORTRAN	128
COBOL	105
PL/1	80

である。

4．10　例

〔例〕下表に示すような平均的なプロジェクトを考える。すなわち、5つの機能が

①10入力処理機能

②10出力処理機能

③10照会処理機能

④1内部論理ファイル機能

⑤1外部インターフェースファイル機能

を持つシステムを考える。

図4．4　例

複雑度 / 機　能	ファンクション数＊重み付け係数			計
	単純	平均	複雑	
入力処理(画面/帳票/ファイル)	×3＝	10×4＝	×6＝	40
出力処理(画面/帳票/ファイル)	×4＝	10×5＝	×7＝	50
内部論理ファイル	×7＝	1×10＝	×15＝	10
外部インターフェースファイル	×5＝	1×7＝	×10＝	7
照会	×3＝	10×4＝	×6＝	40
	調整前ファンクションポイント(A)			147

このとき、図4．5に示すように、システムの特定値の総計が40であるとする。

図4．5　システムの特定値の総計（例）

ID	観　　点	特性値
1	データ通信機能あり	0
2	分散処理あるか	0
3	レスポンスやスループットの制約あるか	4
4	コンピュータ資源の制約が厳しいか	3
5	トランザクション量はどうか	3
6	オンラインデータ入力はあるか	4
7	エンドユーザ処理の充実(画面処理の複雑性)	4
8	DBのオンライン更新あるか	2
9	処理の複雑性はどうか	3
10	流用性を考慮して開発する必要が高いか	0
11	導入移行性を考慮する必要があるか	4
12	オペラーションのしやすさを考慮しているか	4
13	複数個所での使用を考慮して開発するか	5
14	変更容易性を配慮するか	3
	システム特性値の合計(調整用)B	40

システムの特性係数の総計は

40＊0.01＋0.65＝1.05（Complexity Multiplier）

したがって、

最終的なF. P＝147（Unjustified Total）＊1.05〔Complexity Multiplier〕

≒154〔Adjusted Function Point〕

となる。

4．11　ファンクションポイントから工数、月数、要員数への変換

Albrechtの換算式では、次のとおりである[9]。

PH＝54*FP－13390

ここでPH：人時

ただし、500＜FP＜2000とする。

一人月160人時とすると、2.96FP／人月に相当する。この時代での生産性は、低かったと推定する。

国土交通省の2005年度の経済調査会のデータではCSシステムの生産性基準は

11－15FP／人月と報告されている[8]。

Caper Jonesによると、MIS（ビジネス系の情報システム）では、2000年で15FP／人月と推定している[33]。いづれにしても、自部門の生産性基準値を持つ必要があるが、このような「世間相場」を参考にしながら、自部門の生産性データの分析を推進してゆくことが大切である。この2つの調査による世間相場としては、10FP／人月の生産性からスタートして分析することを推奨したい。

要求定義から顧客へ引き渡す前までの概略期間に関する一般的な経験則は、

ファンクションポイント値の0.4乗≒開発スケジュール（月数）

である。Caper Jonesの分析によると分野によって0.32乗～0.45乗以上まで変動している。MISソフトウエア開発分野では0.39乗、軍需ソフトウエア開発分野では0.43乗の分析結果が得られているので、一般的には0.4乗を中心に単純なプロジェクトはより小さく大きなプロジェクトではより大きく見積もることが妥当である。

ソフトウエア開発要員数の見積もりに関する経験則は、

ファンクションポイント／150≒開発要員数（人／月）

である。

また、ソフトウエア開発工数の見積もりに関する経験則は

開発期間（月数）＊開発要員数≒開発工数（人月）

となる。

たとえば、1,000FPのプロジェクトでは、開発期間（月数）は

開発スケジュール（月数）≒$1000^{0.4}$≒16か月

であり、開発要員数は

開発要員数≒1000/150≒6.6人／月

の要員が必要となる。したがって

開発工数≒16か月＊6.6人≒106人月

となる。

これによると

1000FP/106人月≒9.5FP/人月

となり、世間相場とほぼ一致していることがわかる。

5　ソフトウエア開発に必要な工数と期間の予測方法

―PUTNAMモデルによる予測―

5．1　はじめに

1977年にPUTNAMがNorden/Rayleighのマンパワー理論式を活用して提案したモデルである。別名、3者の頭文字をとって“PNRモデル”とも云う[10][11][12]。

PUTNAMモデルの特徴は、他のモデルが工数予測を主体としているのに対して、最適な要員配置を示す“マンパワー曲線”をモデル化した点にある。この曲線の式をもとに、さまざまなプロジェクトの特性を検討することが出来る。Putnamモデルには、次の2つの式が用意されている。

①ステップ数と工数（コスト）、開発期間の関係を示す“ソフトウエアの式”

②工数が与えられている時に、時間による要員数の変化を示す“Norden/Rayleighのマンパワー曲線”の式の2つである。

いずれもプロジェクトデータから経験的に成り立つことが示されている。

5．2　ソフトウエアの式

ステップ数と工数（コスト）、開発期間の関係を示す“ソフトウエアの式”を示す。

$$S=EK^{1/3}t_d^{4/3} \quad \text{または} \quad K=S^3/(E^3*t_d^4)$$

ここで　S：ステップ数

E：環境ファクタ

t_d：開発期間（年）

K：総工数（人・年）

Eは技術レベルを示す定数で、チーム構成、設計技法、マシン環境、テスト技法、ツールの活用能力などのプロジェクトの力を反映するものである。

（Ex）

貧しい開発環境：E＝6,500

良いソフトウエア環境：E＝10,000

卓越した環境：E＝12,500

総工数KはSを3乗、開発期間を4乗した値を使っている。

開発期間や技術レベルやステップ数をわずかに改善する事によって、大幅なコスト改善の余地が得られることを示す。

5．3　Norden/Rayleighのパワー曲線

$m(t)=2K*a*t*\exp(-at^2)$

m：要員数

t ：時間（年、月）

K：コスト総計（総工数：人年、人月）

a ：加速度係数（要員追加の投入の対応能力）

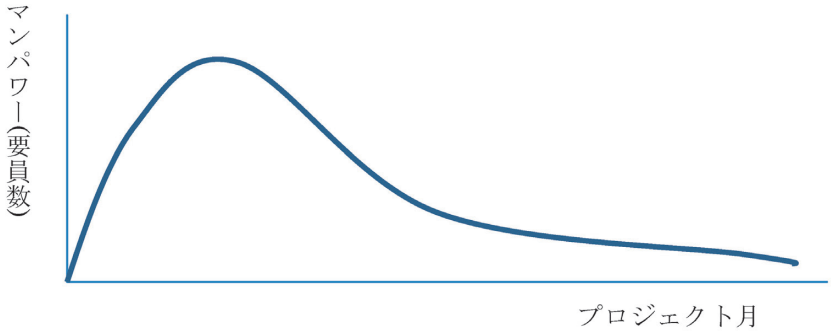
時刻tまでの累積マンパワーC(t)とすると

$$C(t)=\int_0^t m(t)\,dt=K[1-\exp(-at^2)]$$

（注）総工数K：保守作業を含む。

（注）大型プロジェクトではマンパワー曲線のピーク付近が開発完了時点をしめす。中小のプロジェクトではそれがピークの右側にくる。

図5．1　マンパワー曲線



t_d：開発期間とすると

$M'(t_d)=0$より

$$(td)^2=1/(2a)$$

従って、この時

加速度係数$a=1/(2(td)^2)$で

マンパワー曲線m(t)をt_dを使って表わし、

$M(t)=K/(td)^2 t\ \exp(-t^2/(2(td)^2))$

ピーク時のマンパワー　$m_o=m(t_d)=K/(t_d\sqrt{e})$

である。

5．4　例題

コストKが600人・年、開発期間が3年6ヶ月の大規模プロジェクトを考える。

(a) ピーク時の要員数を計算することができる。

$$m_o = m(t_d) = K/(t_d\sqrt{e})$$

$$= 600/(3.5\sqrt{e}) = 104$$

(b) 1年2ヶ月後の累積マンパワーはどうなっているかも計算できる。

加速度係数$a = 1/(2(t_d)^2)$でマンパワー累計$C(t)$をt_dを使って表わし、

$$C(1.17) = 600*[1-\exp(-\frac{1}{2}(\frac{1.17}{3.5})^2) = 600*(1-0.9457) = 600*.0543 = 32.58$$

$$= 32.6\text{（人・年）}$$

（注）$e = 2.7$, $\sqrt{e} = 1.64$

6　ソフトウエア開発のプロセス設計（プロセスモデル）
―ソフトウエアはどのように作られるのか―

6．1　ソフトウエアはどのように作られるのか

ソフトウエア開発のスケジュールマネジメントを行うにあたっては、まずどのようなソフトウエア開発のプロセスモデル（ライフサイクル）を選ぶべきかを決定する必要がある。

ここで、過去、どのようなプロセスモデルが開発され、活用されてきたかを考察しておく。

ソフトウエアのプロセスモデルは大きく分けて、下記の3つのモデルがある。

①ウォータファール・モデル

②スパイラルモデル

③プロトタイピングモデル

それぞれに特徴があり、スケジュール管理をするにあたっては、開発する情報システムに適合したプロセスモデルを選択し、デザインする必要がある。以下詳細に考察する。

6．2　ウォータフォール・モデル [13] [14]

基本となる古典的なプロセスモデルである。要求定義からはじまり、外部設計、内部設計、プログラミング、テストの各工程を一つずつ完成し、滝から水が流れ落ちるように逆戻りが無いように順次確実に開発を進める。

要求事項が明確に出来る事が前提で、開発期間に余裕があり高い信頼性が要求される大規模なシステム開発に適している。

ウォータフォール・モデルのステップは下記のとおりである。この手順に従って、作業を分割し、開発のスケジュールを設計する必要がある。

- ✧ 要求定義：分析：基本計画：要件定義とも言う。主な作業項目は下記にしめす。
  - ➢ 経営課題や業務課題の把握
  - ➢ システムの目的と達成目標の確立
  - ➢ システム機能の分析と決定
  - ➢ 費用対効果
  - ➢ 開発スケジュールの作成
- ✧ 外部設計：概要設計とも言う。　主な作業項目は下記にしめす。
  - ➢ システムの外部から見える部分の設計
  - ➢ 入力設計／出力設計が主
    - ✧ 画面／帳票／レポートの設計
  - ➢ 機能面からの設計
    - ✧ データベースの論理設計
    - ✧ 業務オペレーションの設計
- ✧ 内部設計：詳細設計ともいう。主な作業項目は下記にしめす。
  - ➢ システムの機能をどのように実現するか
  - ➢ システム内部の設計
    - ✧ 機能を実現する為の「仕組み」をプログラミングの観点から設計
    - ✧ ソフトウエア開発者を中心に進められる
      - ● プログラム構成と詳細機能設計
      - ● データベースの物理的設計
      - ● モジュール構造の設計
      - ● テスト計画の詳細化
- ✧ プログラミング

  主な作業項目は下記にしめす。
  - ➢ プログラム設計＋プログラミング工程
    - ✧ モジュール仕様の作成
    - ✧ コーディング
    - ✧ 単体テスト

✧　テスト工程

主な作業項目は下記にしめす。

➢　結合テスト

➢　システムテスト

➢　運用テスト

詳しくは、共通フレーム98における企画開発プロセス（WFLによる共通フレーム、通産資料調査会）を参考にするとよい[16]。

6．3　ウォータフォール・モデルのメリットと課題

✧　メリット

➢　開発の進捗管理がしやすく、

➢　多人数の共同作業となる大規模システムの開発に適し、

➢　信頼性の高いシステムを構築できる。

✧　欠点

➢　実際は机上の世界だけでは要求事項を全て明確にする事は簡単ではない。

➢　大量のドキュメントが必要となる。

➢　開発に時間がかかる。などがあげられる。

✧　まとめ

➢　時間的にも余裕があり、開発を的確に着実に進めたほうがよい大規模な開発に有効である。

➢　柔軟性に欠け、重いプロセスモデルといえる。

一方、システムに対する要求が複雑化・高機能化する現在では、従来のWFLモデルでは対応しきれない限界も見え始めている。たとえば、工程は上流から下流への一方向で、やり直しが効かないという弱みもあり、WFLの限界が言われるようになってきている。

開発面のWFLの限界としては

➢　初期段階からシステムの全てを見通す事の無理

➢　手戻り作業が工程に組み込まれていない

➢　新規の要求は開発途中に組み込めない

➢　完全な要求仕様によるシステム開発

などが考えられる。

利用者側から見たWFLの限界としては、

- ➢ エンドユーザと開発部門の分離による弊害
  - ➢ 参画意識の低さ
  - ➢ ユーザニーズの反映されないシステム作りとなりがちである。
- ➢ 利用部門による新システムの確認テストが最終段階にしか出来ない。

などが考えられる。

6．4　さまざまなプロセスモデル

そこで、以上述べたWFLモデルの欠点を補うために様々なプロセスモデルが考案されている。以下にしめす。

- ✧ 成長モデル
  - ➢ 最初は小さなソフトウエアの開発からはじめ、徐々に成長させていくプロセスモデルである。
- ✧ 柔軟性に富むが、進捗やコストの管理が難しい
- ✧ プロトタイピングモデル
  - ➢ 要求定義の段階で簡単な試作ソフトウエア（プロトタイプ）を作成し、ユーザの評価を繰り返し、確実にユーザの要求を把握するのが狙いである。
    - ✧ 使い捨て型
    - ✧ 骨格型
    - ✧ 拡張型などがある。
- ✧ スパイラルモデル [15]
  - ➢ 成長モデル＋WFL
  - ➢ サブシステムごとにWFLモデルを適用し成長させてゆく
- ✧ 発展的プロトタイピングモデル
  - ➢ 重要な部分からプロトタイプを作成
  - ➢ 徐々に進化させる
  - ➢ 未知の業務分野のシステム開発に適する
  - ➢ 管理上は困難多し
- ✧ 段階的配布モデル
  - ➢ 重要な部分から開発し、順次リリースする
- ✧ インクレメンタル・モデルとも云う
  - ➢ 発展的配布モデル

最初のバージョンリリースから発展的にバージョンアップをリリースする

7　作業分析―作業分析・作業計画―

7．1　はじめに

第6章で考察したように、開発対象の情報システムの特性に応じて、開発プロセスモデルをデザインしなければならない。そのうえで、デザインした開発プロセスモデルに適応した作業の分析を行う必要がある。

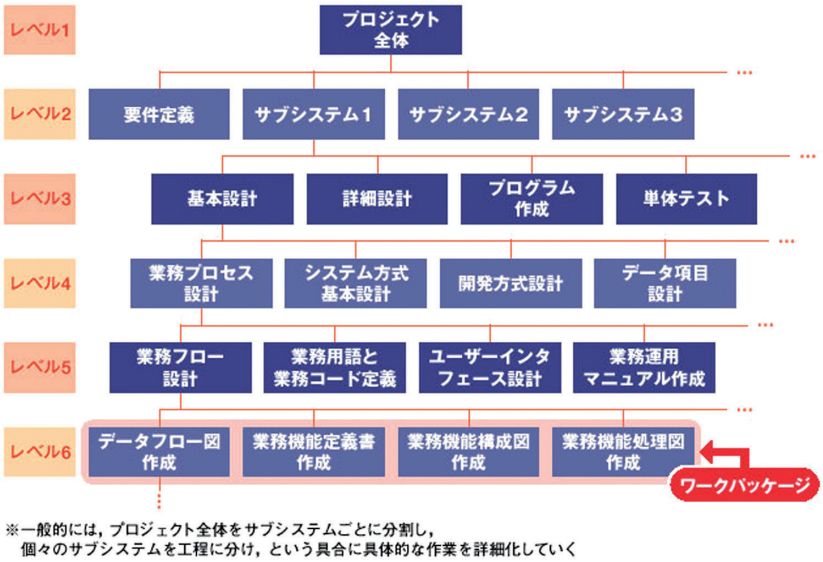
7．2　WBS（Work Breakdown Structure）

作業分析（要素分解）図のことである。スケジュールを管理するにあたっては、必要な作業を分析し、構成することが最も大切なこととなる。

主要な要素成果物をより小さな、よりマネジメントしやすい構成要素に分解することである。

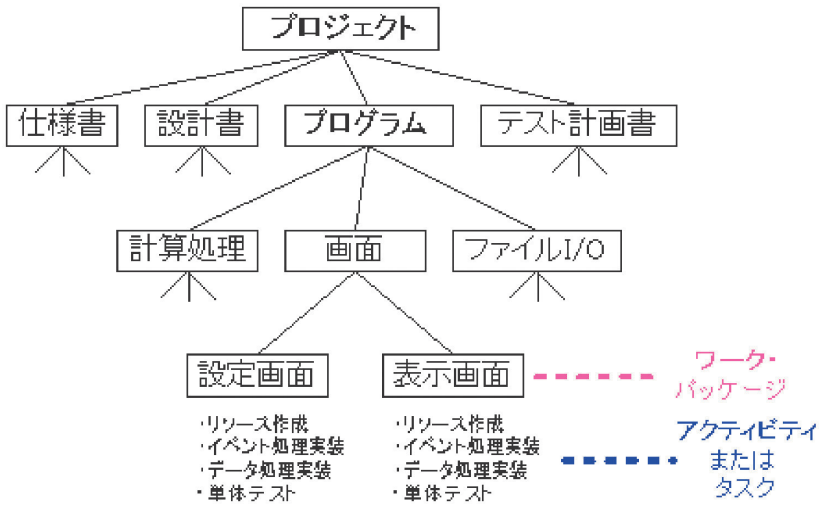
- プロジェクトの主要な要素構成である成果物を特定する。
- 適切なコストと所要時間の見積りが可能になるまでさらに詳細に要素を分解する。
- WBSの構成要素は実績測定を容易にするために、有形かつ検証可能な成果によって記述すべきである。

図7．1　WBSによる作業分割の例 [17]



WBSとは、プロジェクト全体を、成果物を生み出す作業単位に分割すること。大日程計画では、WBSの最小レベルである「ワークパッケージ」まで分割し、中日程計画や小日程計画では成果物の有無を考慮しない作業単位である「アクティビティ」まで分割する。

図7．2　開発フェーズによるWBSの例 [18]



	A	B	C	D	E	F
1	WBS (Work Breakdown Structure)					
2						
3	大項目	中項目	小項目	タスク		
4	仕様書					
36	設計書					
65	プログラム					
66		計算処理				
81		画面				
82			設定画面			
83				リソース作成		
84				イベント処理実装		
85				データ処理実装		
86				単体テスト		
87			表示画面			
88				リソース作成		
89				イベント処理実装		
90				データ処理実装		
91				単体テスト		
92		ファイルI/O				
103	テスト計画書					

多くの適用分野には、テンプレートとして利用できる標準WBSやそれに準じたWBSがある。

（例）システム開発取引の共通フレーム（SLCP-JCF2007、ISO/IEC 12207、JIS X0169）[16]

（例）DODの国防調達品用の標準WBS [19]

プロジェクトに関わる作業を全て拾い出し、作業内容を明確にすることが肝心である。作業分解図又は系統的作業一覧表のことをいう。

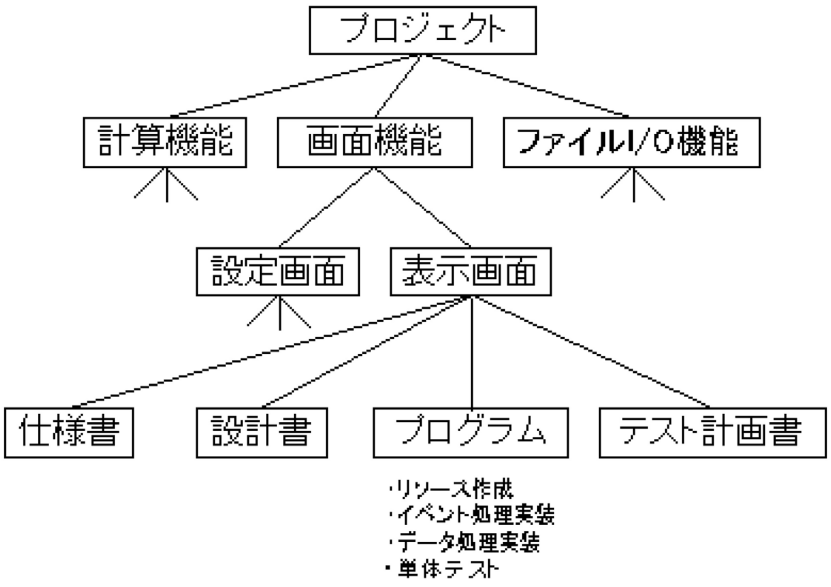
7．3　分解方法

- ✧　機能（または成果物〕による分類を優先する場合と
- ✧　開発フェーズ（又はプロセス）による分類を優先する場合

がある。結果的にはどちらでも最終の作業単位では同じとなる。

一般的には、プロジェクトの成果物（納品物）を全て洗い出す事から始めると良い。

図7．3　機能によるWBSの例 [20]



8　日程計画と進捗管理

8．1　アクティビティ定義

- ✧　WBSをベースにプロジェクトのワークパッケージをより小さく、よりマネジメントしやすい要素のアクティビティに分割する。
- ✧　アクティビティ定義の最終アウトプットは成果物ではなく、あくまでも、アクティ

ビティ（活動、又は作業）であり、プロセスである事に留意する事。

- ✧ WBSの第3レベルのワークパッケージで通常は十分であることが多いが、さらに出来るだけ詳細に細分化する方が管理しやすくなる。
- ✧ そのプロジェクトにおいて実行する全てのアクティビティを記述しなければならない。
  - ➢ 分類の主な基準
    - ✧ 機能（または成果物、納品物、構成要素など）による分解
    - ✧ 開発フェーズによる分解
  - ➢ 大切な事はそのプロジェクトにおいて実行する全てのアクティビティを記述しなければならないと言う事である。

8．2　アクティビティ順序設定

- ✧ アクティビティ相互の論理的関係を明確にし、文書化すること。
  - ➢ ここでは、期日や期間が確定していなくても良い。これは「スケジュール作成」の段階で確定すればよい。
  - ➢ あくまでも、アクティビティ間の論理的関係が分析できていれば良い。
- ✧ Tools
  - ➢ アローダイアグラム法（PERT図法）[20]
- ✧ Outputs
  - ➢ プロジェクト・ネットワーク図

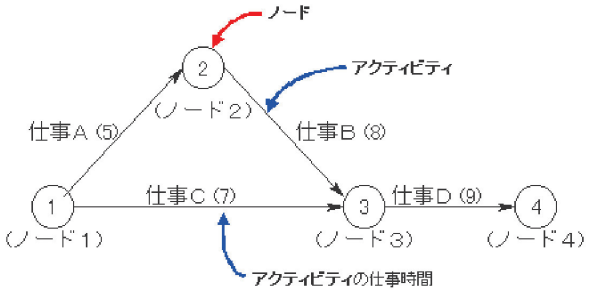
8．3　PERT（Program Evaluation and Review Technique）図法の説明 [20] [21]

PERTとは、製品開発の日程計画を立てる時に用いられる技法である。複雑な仕事の処理順序の関係をネットワークの形でアローダイアグラムによって表現し、プロジェクトの開始から終了に至るまでの仕事の処理時間に余裕のない経路（クリティカルパス）を明確にして、予定工期までにプロジェクトを完成できるかどうかの計画の実行可能性を検討し、管理の重点を明らかにする手法である。

〈ネットワークの表現方法〉

矢印（アクティビティ）……仕事

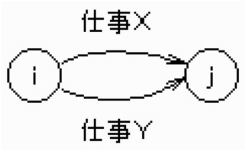
丸印（ノード）……仕事の着手または完了ポイント



注意点

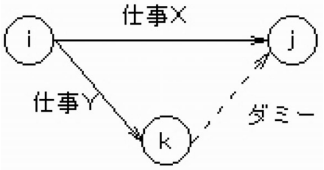
①2つのノード間に含まれる仕事は1つに限る（仕事の一意性）。よって、下図8．1のような表現は不可能である。

図8．1



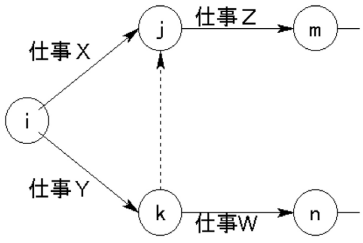
②図8．1に示す仕事X、Yのように、同じ結合点から出て同じ結合点に入るような平行して行う仕事がある場合は、点線で所要時間が0の架空の仕事（ダミーアクティビティ）を作り、図8．2のように示す。

図8．2



③図8．3のように、仕事Zの先行する仕事がX、Y両方であるが、仕事Wの先行する仕事がYのみである場合には、ダミーを用いないと正しい先行関係を表現できない。

図8．3



④あるノードに入ってくる仕事（または出て行く仕事）は、全て共通な後続する仕事（または先行する仕事）を持っている

⑤プロジェクトの開始と終了をそれぞれ1つのノードにまとめる

⑥矢印の方向に従って、ノード番号を大きくする

〈日程の計算方法〉

①プロジェクトに含まれる複数の仕事の先行関係を、アクティビティとノードとダミーで表現する。

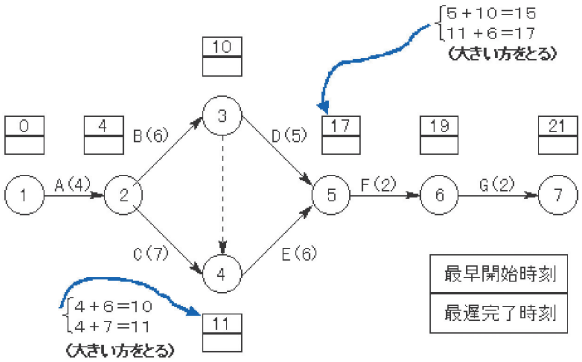
②仕事であるアクティビティ（矢印）の上に、仕事の番号とその時間値を記入する。

③最早開始時刻を計算する（図8．4）

最早開始時刻

あるノードから出る仕事を最も早く開始できる時刻

図8．4　最早開始時刻の計算

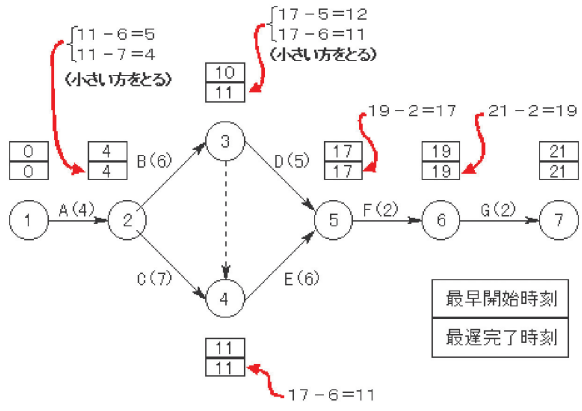


④最遅完了時刻を計算する

最遅完了時刻

プロジェクト全体を予定期日に完成させるために、各ノードで遅くとも仕事が完了していなければならない時刻

図8．5　最遅完了時刻を計算



8．4　アクティビティ所要期間見積り

- Tools
  - 専門家の判断
  - 類似見積り
  - 定量ベース所要期間
    - （例）図面枚数＊生産性
    - （例）開発ステップ数＊生産性
  - 予備時間
    - リスクに応じた予備時間、コンティンジェンシー、バッファー

8．5　スケジュール作成

- プロジェクトにおける各アクティビティの開始日および終了日を決定する事である。
- Tools
  - クリティカルパス法
    - 各アクティビティに対し単一の確定的最早・最遅開始日および終了日を計算する。

〈クリティカルパスの例〉

最早開始時刻と最遅完了時刻が等しいノードを始点から終点まで結んだ経路をクリティカルパスという。上の図では①→②→④→⑤→⑥→⑦がクリティカルパスである。

プロジェクト・スケジュールをネットワーク図で表現したとき、プロジェクト全体の作業開始から終了までをつなぐ、まったく遊び時間のない経路が少なくとも1本できる。この最長経路がクリティカルパスである。

クリティカルパス上にない作業は、遅れが出ても余裕（フロート）の範囲内であればプロジェクト全体のスケジュールには影響しない。しかしクリティカルパス上の作業が遅延すると、プロジェクト全体の納期を遅らせてしまうことになる。逆にクリティカルパスが短縮できるとプロジェクト期間も短縮できる。このため、生産管理／プロジェクト管理においては特に重要な管理対象である。

〈例題〉

次の図8．6の引越しについての作業表をもとにして、PERT図の作成と、PERT計算を行なう。

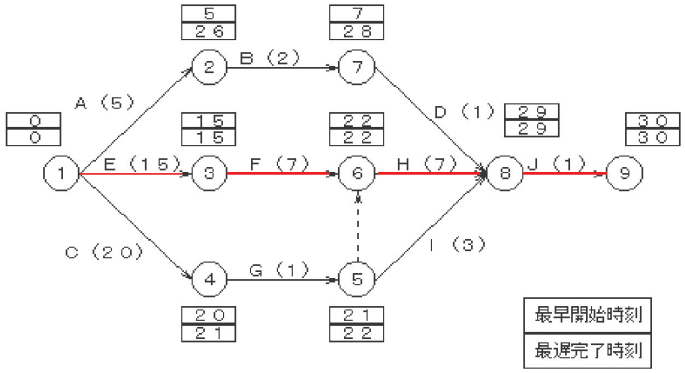
図8．6　作業表

	仕事項目名	所要日数	先行関係
A	業者決定	5	—
B	見積もり	2	A
C	新居決定	20	—
D	業者と打ち合わせ	1	B
E	荷物の整理	15	—
F	ダンボール箱調達	7	E
G	新居下見	1	C
H	荷造り	7	F，G
I	家具購入	3	G
J	引越し	1	D，H，I

PERT図

図8．6に基づいて、PERT図を作成すると次のようになる

図8．7　PERT作成例



赤線（太線）がクリティカルパスである。

8．4　スケジュールコントロール

次にスケジュールを調整する手順を示す。

- (a) 納得できる変更となるように、スケジュールの変更要因に働きかける事
- (b) スケジュールが変更された事を確定すること

事が大切である。

- Inputとしては
  - 実績報告
  - 要求変更

- ✧ Outputsとしては
  - ➢ スケジュール更新版
  - ➢ 是正処置
  - ➢ 教訓

を残すことが大切である。

9　構成管理システムによるソフトウエア開発マネジメント [34]

9．1　構成管理とは

構成管理（Configuration Management，以降CMと称す）とは、製品の構成要素を識別し、これに対する変更を管理する手法である。ハードウエアの構成管理は、第2次世界大戦の初期からロッキードなどで手作業であるが開始された。当時、米国国家基準局などがいくつかの規定を作成しているが、コンピュータ・プログラムの構成管理についてはほとんど何も書かれていなかった。

1970年代になると、ソフトウエアも含むシステムとしての構成管理の重要性への認識が高まり、1980年代には、米国のDOD規格にソフトウエア構成管理（Software Configuration Management，以降SCMと称する）が組み込まれ、米国の産業界に多くの影響をもたらしている。

9．2　現状と課題

分野に限らず、多くのソフトウエア開発担当者や管理者が経験するように、この研究活動においても下記のような問題を抱えていた（図9．1）[22]。

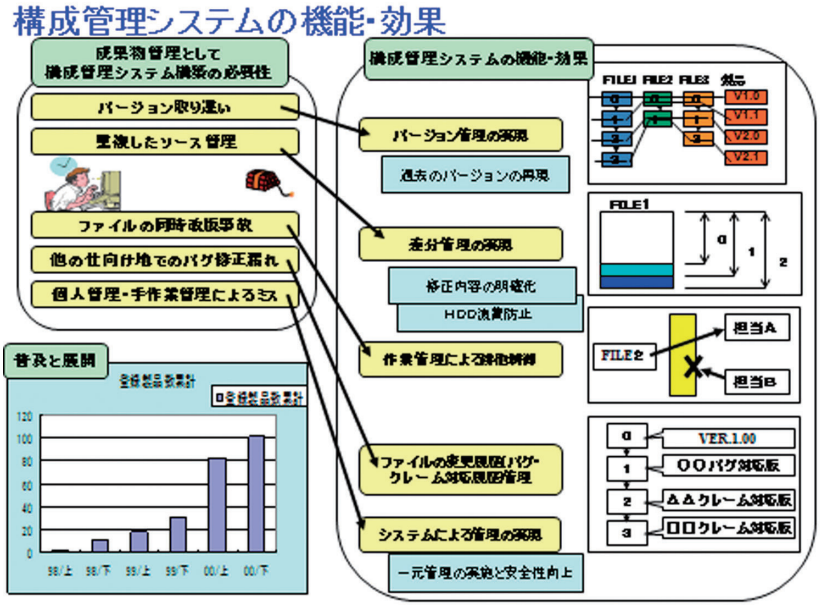
1. ひとつのファイルを複数の開発担当者が同時に改版してしまい、ファイルの整合性が取れなくなってしまった（ファイルの同時改版事故）。
2. 思い込みや記憶違いで、誤ったバージョン（古いバージョン）のファイルをコピーして修正し、新しいバージョンとして適用してしまった（バージョンの取り違い現象）。
3. 必要なファイルであるにも関わらず、誤って消去してしまい、いつの間にか消失してしまった（ファイルの誤消去事件）。
4. 開発が進むうちに完成ソフトウエアのファイルが認識できなくなり、どのファイルが完成ファイルなのかわからなくなった。
5. バージョンごとに一式のファイルを保管し、重複したソース管理となってしまい、

膨大な保管容量となってしまった。

6. 異なるバージョンで別々に同一バグを修正せざるを得なくなってしまい、修正ミスや修正漏れが発生してしまった。

などなどである。

図9．1　構成管理システムの機能と効果 [22]



9．3　ソフトウエア構成管理システムの導入

そこで、下記の機能を持つソフトウエア構成管理システムの導入を検討し、適用することによって、上記に記載した開発担当者や管理者が抱える諸問題の解決に対応することとした（図9．2)。

(A) 作業領域環境管理による排他制御機能により、現在の改版作業者が明確になるシステムを実現することによって、ファイルの同時改版事故を防止できる。

(B) ファイルのリビジョン管理と製品のバージョン管理が可能なシステムにより、誤コピーによるバージョンの取り違いの阻止。

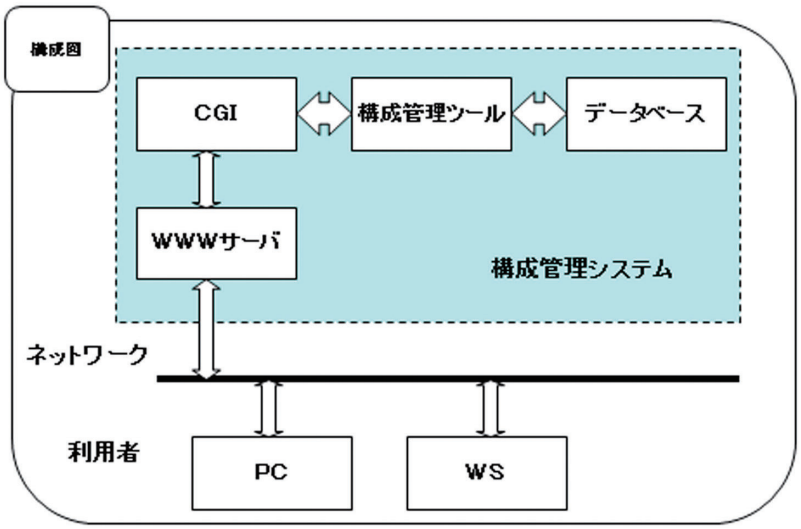
(C) ファイル構成を可視化することにより、完成ソフトウエアのファイルがわからなくなることを防止できるシステム。

(D) ファイルの差分管理を可能とし、バージョンごとに一式のファイルを持つことを必要としないシステム。

(E) ひとつの変更内容がどのバージョンに反映すればよいのか容易になり、異なるバージョンで別々に同一修正をする必要がないシステム。

(F) バックアップをセンターサーバで保証することにより、万が一でのファイル消失を防止できるシステム。

図9．2　構成管理システムの構成図



9．4　ソフトウエア構成管理システムによる開発プロセスの分析と改善手法についての提言

すべての企業は、ソフトウエア構成管理システムを導入・構築し、ソフトウエアの資産管理と変更管理を徹底し、ソフトウエア資産の安全性、信頼性、有効性を高め、健全性を保証することが不可欠である。

と同時に、ソフトウエア開発プロセスの分析とその更なる改善としての新たな取り組みとして、下記の構想を提言する。

構成管理システムを単に、ソフトウエア資産の保管と、変更管理の活用にとどめるのではなく、構成管理を必要とするソフトウエア開発プロセス全範囲において、構成管理システムに管理されるデータならびに構成管理をアクセスするプロセス〔過程〕自体の情報を意図的に取り入れ、効率的に活用し、ソフトウエア開発プロセス自体を目視化し、分析可能にすることによって、ソフトウエア開発プロセスの改善につながる手法を提言するものである。

従来の開発者による定性的なまたは不正確な（残念ながら）情報による分析から成果物

情報とプロセス情報の累積を基に、正確なかつ静的な（スタティックな）または動的な（ダイナミックな）情報による適確な分析を可能にするものである。

そして、ソフトウエア開発にかかわる品質改善、工期の短縮、開発コストの低減に貢献できることを狙いとするものである[23][24][25]。

9．5　基本構想

基本構想としては次の3つのプロセス分析と改善の仕組みを作ることを狙いとする。

①成果物を管理する構成管理システムを発展させ、これにプロセス（ソフトウエア開発過程）に関する情報を加え、開発プロセスを可視化し、分析できる仕組みを作る。

②開発プロセスの可視化と現状分析により、開発過程のプロセスに対して、まず、動的な生産状況の現状把握から着手し、順次、ソフトウエアまたはシステムの品質特性の把握（品質管理)、工数実績の把握（工数管理)、工程の進捗状況の把握（工程管理）へと段階的に改善を強化する仕組みを作る。

③また、これらの現状分析データを中長期的に蓄積することによって、製品別、部門別の統計的なデータを分析する仕組みを作り、製品別、組織別の基準値を見出せることを可能とし、また製品別、部門別の問題点を分析すると同時に全組織での問題点を把握する仕組みを作る。

9．6　開発プロセス分析システムの仕組み

構成管理システムと開発プロセス分析システムの関連を図9．3に示す。

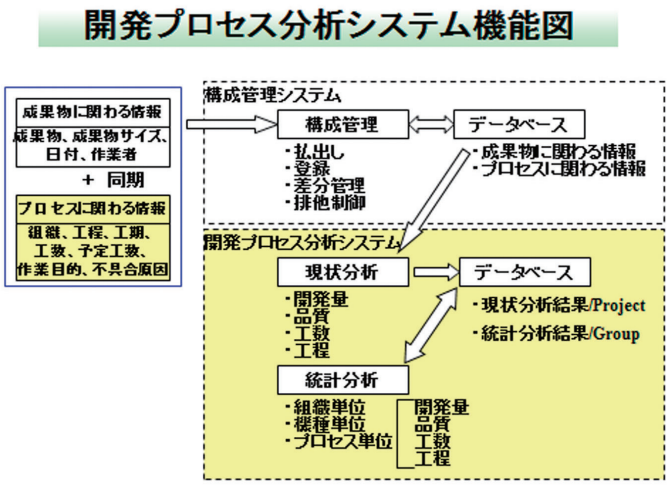
構成管理システムの対象はシステム開発プロセスのシステム設計、ソフトウエア設計、プログラム設計、検査に関する設計情報、ならびにソースコードとなる。

また、プロセス情報としては、作業目的、変更要因、開発工程、日時、担当者名、所属部署名、工数、予定工期、予定工数などプロセス分析に必要なプロセスデータで構成される。

これらのプロセス情報を入力する機能は市販の構成管理システムではまだ標準化されておらず機能も十分ではない。　したがって、この研究では、wwwを利用したカスタマイズの容易さと使いやすさを考慮したデータマネージャーを開発し適用している[26][27]。

構成管理システムはプログラムやドキュメントなどの成果物を登録し、その変更履歴を管理する。格納される情報は成果物、成果物サイズ、登録者名、登録日時などであり、これを成果物情報という。

図9．3　開発プロセス分析システム機能図

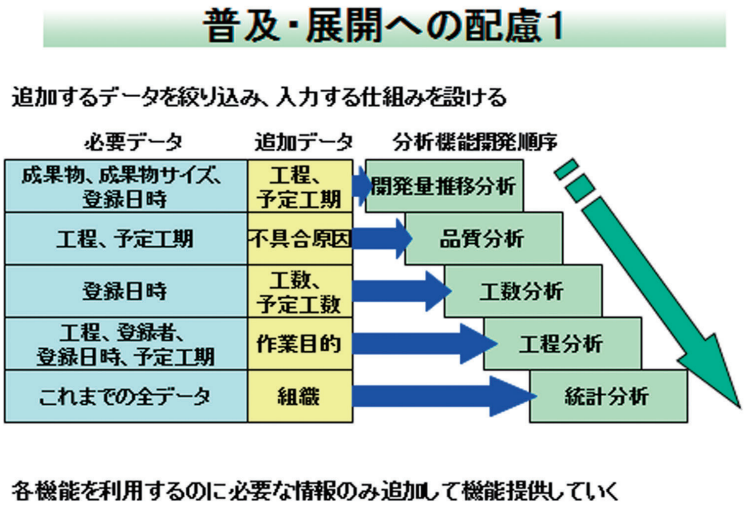


9．7　段階的な成熟レベルアップへのアプローチ

開発プロセスを分析するためには、構成管理システムが持つ本来の成果物情報に加え、分析に必要なプロセス情報を加味しなければならない。しかし一度にすべてのプロセス情報を入力することを求めることは、開発技術者にとっても管理者にとっても大きな負担となる。そこで、この研究では、開発プロセスとして下記のように段階的に分析システムを発展させていく成熟レベルアップのアプローチを考案した。

構成管理システムが持つ本来の成果物情報に加えて、各機能を実現するために必要にして最小限のプロセス情報を追加してゆくことによって、下記の分析機能を段階的に実現しながら提供してゆくことが可能になる（図9．4）。

図9．4　段階的な成熟レベルアップへのアプローチ



①各プロジェクトのプロセス毎の動的な生産状況のモニタにより、プロセスの状況を把握する。例えば

（1）開発着手の遅れ期間の状況。

（1）開発が進んでいるのか。停滞しているのか。

（2）開発量や修正量が増え続け、予定通りに開発が終わりそうにない。

（3）リリース後も安定していない。

（3）予定通りの開発量が出来ているのか。

（4）累積開発量と開発期間による静的生産性の分析。

②変更内容や修正量などの分析による品質分析を可能とする。例えば

（1）開発プロセスにおいて開発量が安定しない。変動が激しい。仕様の見直しや追加が繰り返されている。

（2）テスト期間に入ってからの変更量が多い（バグが多い）。

（3）テスト期間での変更量が安定しない（テストが収束しない）。

（4）出荷後の修正量が多い（出荷後のバグ修正が多い）。

（5）特定のモジュールに修正が集中している。

③変更日時による工数推定や実績工数の入力により、コスト分析を可能とする。例えば

（1）開発プロセスにおける生産性（ステップ／日）が基準値よりも低い／高い。

（2）テストプロセスにおける工数比（修正ステップ／総ステップ）が基準値よりも低い／高い。

（3）保守プロセスにおける工数比（修正ステップ／総ステップ）が基準値よりも低い／高い。

（4）各プロセスの工数比率の適正を分析できる。

④予定工期と実績工期の分析により工程（工期）分析を可能とする。例えば、

（1）システム設計／プログラム設計／テスト／保守プロセス等の工程（工期）分析により進捗状況とその要因の把握が可能となる。

（2）システム設計／プログラム設計／テスト／保守プロセスの工期比率の適正性分析を可能とする。

⑤現状データを積み上げ統計分析することにより、企業全体、部門、機種、工程別の基準値を得ることが可能となる。たとえば

（1）開発期間に対する開発量の統計分析により、生産性の基準値を得ることができる。

これにより、より精度の高い開発コストの予測や開発工程計画の立案が可能となる。

(2) 品質阻害要因を分析し、品質改善活動の重点項目を設定することができる。

(3) 工程ごとの工数と期間を統計分析することにより、基準となるマンパワー曲線を得ることができる。

9．8　実施事例

構成管理システムによる開発プロセス分析システムについての実施事例の一例として、「開発量推移分析」を示す。出力例を図9．5に示す。

成果物のサイズと開発量のサイズが開発プロセスの進捗に応じて、きめ細かく観察される。たとえば

①修正等の作業が差分として見て取れる。

②開発プロセスでは仕様の変更や設計のやり直しが多いかどうかが観察できる。

③テストプロセスではバグの修正が観察できる。

④保守プロセスではクレームの頻度と修正時期や修正量が観察できる。

ことが期待できる。

(a) 具体的な事例として図9．5の例で見られるような開発プロセスでの停滞状況（stagnation）が観察された。これは、開発プロセス中に生じた仕様の変更のため、一時開発を停止せざるを得ないことを示していた。

(b) 急激なソースコードの増大（Rapid Increasing）が観察された。これは多くの場合、すでに開発されソースコードの再利用が実施されたことを示す。どの程度、ソースコードの再利用率が達成されているか観測できると同時に、再利用ソースコードの修正度合も観測することができる。また、再利用率と開発工数との相関関係を観察することによってより適切な開発工数の見積もり手法も確立できる。

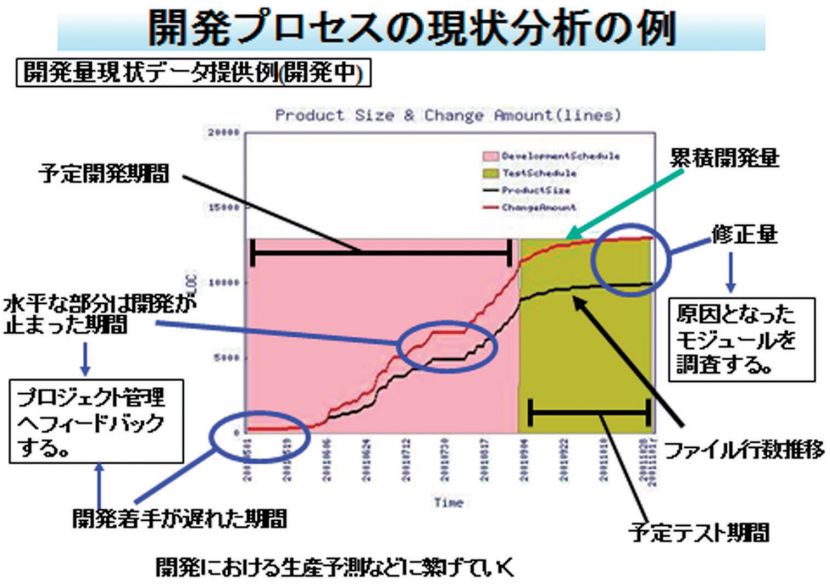
(c) 開発ソースコード量とプロダクトコード量との差の急激な変化が観測された。たとえば、

・開発プロセスにおいては仕様の変更要求が入ることによる急激な変化が観察される場合、

・デバッグプロセスやテスト（検査）プロセスにおけるソースコードの修正作業が頻繁におこなわれる場合、

・保守プロセスにおけるクレーム対応における修正が頻繁に行われる場合
・または、テスト期間に投入されていたテスト用のコードを最終製品にするときに除去した場合などの現象が観測されている。

図9．5　開発プロセスの現状分析の例



9．9　オフラインからオンラインへ

ソフトウエア開発プロジェクトまたはソフトウエア開発プロセスにおいてプロセスの実態を見る努力と見せる仕組みを築き上げていく努力は管理面においても技術面においても大変重要で意義あることである。

しかしながら、ソフトウエア工学の研究とその実践が求められた1960年代の「ソフトウエア危機」の時代から今日まで、実データに基づいた見せる努力と見る努力が十分にかつ具体的になされてきたかというとまだまだ満足できるものではない。

オフライン的にまたはバッチ的にデータを収集してデータ分析するケースはまま見受けられるにしても、特に、オンラインで構成管理の仕組みやエディターの仕組みを利活用し、開発プロセスの現状把握と分析をしかも組織的に取り組み実現しているケースはまだまだ少ないといえる。

今後、オンラインでのプロセスデータの分析機能による、プロジェクト推進中でのコスト管理、品質管理、工程管理などへのフィードバックの仕組みやいくつかのプロジェクト

結果における統計的データの分析結果を、次期ソフトウエア開発プロジェクトへのフィードフォアード的な管理へ活用する仕組みをさらに追及すべきと思える。

10　最後に

現在の高度情報化社会ではあらゆる分野にコンピュータが適用され、今や社会のインフラになっているので、ひとたび故障すると大きな社会問題となる危険性が高い。

規模的にもコスト的にもハードウエアに比してソフトウエアの占める割合が急増している。ハードウエア自体はLSI化が進み品質はますます向上している中、コンピュータを動かすソフトウエアも高品質でなければならない。

また、ソフトウエアシステムの社会的責任が拡大しつつある。その背景にはシステムの大規模化と複雑化がある。直接的または間接的にもその影響度は高くなりつつあり、かつ広範囲にわたる傾向が強い。

このように、ソフトウエアマネジメントの必要性は高まるばかりであるにもかかわらず、ソフトウエア開発の品質、コスト、開発期間そして満足度などに関して、十分にマネジメントされているとは言い難い。むしろますます大きな問題を抱えつつある。

その大きな要因はソフトウエア開発のマネジメント教育が十分いきわたっていない事にある。また、「ソフトウエアを開発する者は開発状況を見せる工夫や努力が十分といえず、ソフトウエア開発を管理する者は見ようとする努力が十分ではない。」ところに課題があるといえる。

ソフトウエア開発をマネジメントすべきことは品質、コスト、スケジュールと多々あるが、ここではソフトウエア開発のコストマネジメントとスケジュールマネジメントについて考察してきた。

コストマネジメントにおいては、まず、開発プロセスに応じたコスト予測の手法を選択することが大切であることが分かった。次に工数予測にあって、大切なことは基準値法を用いるにしても、COCOMOモデル法やファンクションポイント法を用いるにしても、自部門の生産基準値を持つ事が重要となる。一つの開発プロジェクトが終わる毎にそのプロジェクトの生産基準値を収集し、次回の開発プロジェクトへ反映していくことをたゆまなく継続することが大切なこととなる。まさに継続こそ力なりである。

スケジュールマネジメントにおいては、最も重要なことはその開発プロジェクトに必要な作業（Works）を適切にブレイクダウン（Breakdown）できるかどうかにかかっている。

そのうえで必要にして欠かすことのできない技法はPERT図技法である。スケジュールを計画管理するにはPERT技法をしっかりと身につけることを怠ってはならない。そして進捗の管理にさらに不可欠なのは、進捗に応じて達成しなければならないマイルストーンを明確にすることにある。作業分解 (Works Breakdown structure) 時に作業の結果得られる成果物をマイルストーンとして設定することにある。

最後に第9章の構成管理システムによるソフトウエア開発マネジメントの例にて考察したように、開発プロセスが進むにつれてその進捗状況が管理者にも開発者にも共有できる支援システムを構築することが大切である。ぜひ参考にしていただきたい。

参考文献

［１］ソフトウエア開発プロジェクト管理、グローバルナレッジネットワークインク、1998

［２］Bert Boehm : Software Cost Estimation with COCOMO II、Prentice Hall； 1版 2000

［３］情報システム概論第4回、www.myu.ac.jp/~infosys/infosysdesign1/lesson4.ppt

［４］ドティ・アソシエイツ編、上条史彦監訳：ソフトウエアのコスト積算、1981

［５］Meyers, Ware Putnam : Measures For Excellence : Reliable Software On Time, Within Budget、Lawrence H.1991

［６］B. W. Boehm : Software Engineering Economics, Prince Hall, Inc., 1981

［７］真野俊樹、誉田直美：見積もりの方法、日科技連出版社、1998

［８］児玉公信：ファンクションポイント法、日経能率協会マネジメントセンター、2006

［９］Albrecht : Software Function, Source Lines of Code ,and Development Effort Prediction, IEEE Transaction on Software Engineering, Vol.SE-9 (6), Nov.1983

［10］L. Putnam : Software Cost Estimating and life Cycle Control, IEEE Computer Society Press, 1980

［11］L. Putnam : A general empirical solution to the macro software sizing and estimating problem, IEEE Trans. Software Engineering, Vol.SE-4, No4, 1978

［12］L. Putnam : Measurement data to support sizing, estimating and control of the software life cycle, Proc. COPCON, 1978

［13］森口繁一（監修）：ソフトウエア品質管理ハンドブック、日本規格協会、1990

［14］B. W. Boehm : Software Engineering, IEEE Trans. Software Eng., Vol. C-25, 1976

［15］B. W. Boehm : A Spiral model of software development and enhancement, ACM Software Eng. Notes, Vol.11, 1986

［16］『共通フレーム2007――経営者、業務部門が参画するシステム開発および取引のために』情報処理推進機構 ソフトウェア・エンジニアリング・センター＝編／オーム社／2007年10月 (SLCP-JCF2007、ISO/IEC 12207、JIS X0169)

［17］http://itpro.nikkeibp.co.jp

［18］http://www.aerith.net/design/wbs/wbs2-j.html
［19］MIL-STD-881C、防衛標準の運輸省ディフェンス資材項目の作業分解構造（WBS）（03-OCT-2011）［後継、MIL-HDBK-881A］
［20］http://lab.mgmt.waseda.ac.jp/prod_b/bpr/third/tejyun1.htm
［21］大村　平：ORのはなし、日科技連
［22］吉崎・大木他：ソフトウエア構成管理プロセス改善一手法(2)、情報処理学会第60回全国大会、2000.
［23］吉崎・大木他：構成管理データに基づく開発プロセス分析の一手法、情報処理学会第64回全国大会、2002
［24］吉崎・会沢他：『構成管理システムを活用したソフトウエア開発プロセス改善への取り組み事例、第66回情報処理学会全国大会、2003
［25］Aizawa : An Approach to Improvement of Software Project Management by Utilizing Data from SCM System, 3rd World Congress for Software Quality, 2005
［26］東陽テクニカ：CASEフェスタ'01　ソフトウエア構成管理の高度な実践方法、2001
［27］吉崎・会沢他：フトウエア構成管理プロセス改善一手法（1)、情報処理学会第60回全国大会、2000
［28］徳田弘昭著：『ソフトウエア構成管理～SCMでソフトウエア開発の革新を～、ソフト・リサーチ・センター、1999
［29］徳田弘昭著：ソフトウエア構成管理ハンドブック～ソフトウエア資産管理のキィテクノロジー～、ソフト・リサーチ・センター、2004
［30］（社）情報サービス産業協会：SCM（ソフトウエア構成管理）に関する調査研究報告、平成9年
［31］http://itpro.nikkeibp.co.jp/article/COLUMN/20070417/268532/
［32］Caper Jones : Estimating Software Costs, McGraw-Hills, 1998
［33］Caper Jones : Applied Software Measurement, McGraw-Hills,1996
［34］吉崎浩二：ソフトウエア構成管理システムによるプロセス分析と改善手法についての考察、上武大学経営情報学部紀要　第29号、2006年12月