解　説

開　発　管　理　技　術†

花　田　收　悦††

1. 開発管理技術の位置付け[1),6)]

ソフトウェア工学については多様な定義が試みられている．よく利用されているものとして，次の Boehm と Parnas のものがある．

（1） Boehm の定義

プログラムの設計製造，およびその開発・運用・保守に必要なドキュメント化に対し，科学的な知識を実際に応用すること．

（2） Parnas の定義

複数の人間が複数バージョンのプログラムを作成するための諸技術．

Boehm の定義は，ソフトウェア生産におけるドキュメント化の重要性を暗示している点において先見性を感ずる．

一方，Parnas の定義については理解するのに背景を認識することが必要である．すなわち，「複数の人間が複数バージョンのプログラムを作成する」という行動は，ソフトウェアの開発期間を少しでも短縮することにより，開発したソフトウェア製品の付加価値を高めようとする企業競争の実態を認識しなければならない．少人数でゆっくりと時間をかけてソフトウェアを開発する方が，よいソフトウェアを開発できることは明らかであるが，その製品の鮮度を維持できるという保証はない．

以上の二つの定義は，ソフトウェア工学のもつ二つの大きな技術分野である開発技術（Boehm）と開発管理技術（Parnas）のそれぞれ片面にスポットを当てた表現として解釈するのが妥当であろう（図-1）．

† Software Developing Management Technolgies by Shuets HANATA (NTT Software Production Technology Laboratories).

†† NTT ソフトウェア生産技術研究所

表-1　開発管理の基本パターン

開発管理のサイクル	主要な作業項目
計画	・プロジェクトの特色把握 ・目標の設定 ・チームの編成
実施	・データの収集 ・計画との Check & Action
評価	・計画との相違点の整理 ・分析 ・一般化 ・今後への提言

ここでは，後者の開発管理技術について述べる．

開発管理に限らず，一般に管理の基本は Plan-Do-Check-Action のフィードバックループの確立にある（表-1）．

フィードバック先は自プロジェクト内の場合（内側のループ）と，後続の他プロジェクトの場合（外側のループ）とがある．他プロジェクトにフィードバック

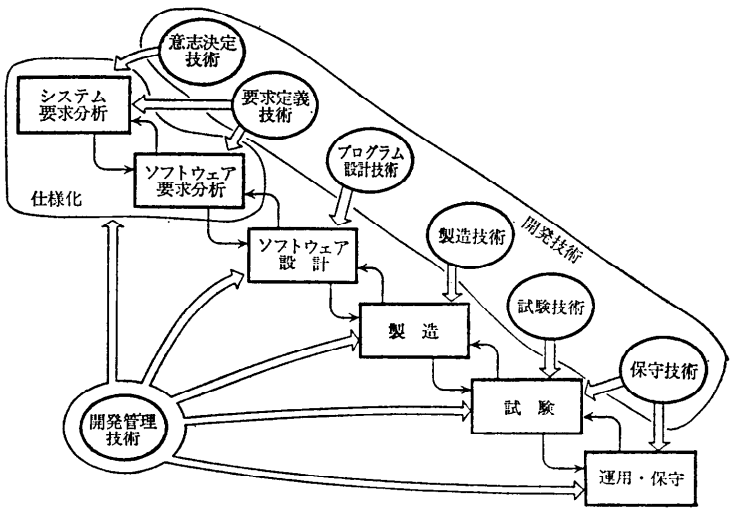


図-1　ライフサイクル・モデルと開発技術，開発管理技術との関連(例)

するためには，自プロジェクトの特殊事情を除去して一般化した形式に成果を集約しないと利活用が難しくなるので，プロジェクトリーダはプロジェクト終了の最後のしめくくりとして，各種の開発管理上のデータとともに，「今後への提言」をまとめる義務がある．

2. 開発計画[4]

管理の定石である Plan-Do-Check-Action のサイクルを実施するために，最初の出発点となる「計画(Plan)」についてしっかりとしたものを作成することがいかに重要かについては自明であろう．

計画が正しく立てられ，その内容が担当者に十分に周知されていると，仮りに計画どおりに進展しなくとも計画のどこの部分に狂いが生じたから，どこを補正すればよいというリカバリが自信をもって実施できる．計画時に十分考えていないとその因果関係の解明に時間がとられてタイミングのよいリカバリが立てられず，プロジェクトの混乱が増大する．

十分に練られた計画でも，プロジェクトがそのとおりに進行するのはまれである．したがって用意周到な計画でもズサンな計画でも，計画どおりに進行しないという側面は同じであるが，その後のリカバリを早く，的確に実施できるという点において差が生ずることになる．

一般に，計画立案においては過去の類似ソフトウェアの開発事例を参考として，自プロジェクトの特殊性を加味しながら補正・立案するのが通例である．この場合に開発規模の見積り，開発するソフトウェアの複雑性の予測，信頼性予測モデル，コストモデルなどの計量化（メトリクス）が行われるが，現在のところまだ決定的手法は見出されていない．

開発計画はプロジェクトの開始時に全体計画を策定しておき，さらに各工程の開始直前に工程別の詳細計画を作成する方法がよく用いられる．プロジェクトの終了時には開発完了報告の提出を義務づけ，表-1 の「評価」内容を十分に論議する．

全体計画の立案に必要な作業内容としては，表-2 に示すように次のような作業項目が主要なものである．

- 作業スケジューリング
- ニーズの分析
- 問題点の把握
- システムの構想
- ソフトウェア仕様[3]
- ハードウェア仕様
- ソフトウェア規模の見積り作業
- ドキュメンテーション

このうち，計画のベースとなるのは開発対象のソフトウェアの規模見積りである．一般には，これまでに開発した類似システムを何個か探し出して，それらの開発にかかわる種々のデータに対して，このプロジェクトの開発条件・開発環境・担当者のスキルなどの相違の程度に応じて補正して全体計画を作成する．

規模の見積りと実際の完成規模との計測事例を図-2 に示す．この事例はすべてコンパイラの開発事例であり，類似システムが存在しているケースである．過大見積りと過少見積りがあるが，比率的には機能の見落としなどに起因する過少見積りのケースが多い．

3. 工程管理

工程管理は開発に基づく作業の進行状況を把握し，もし問題点が発見されたらその対策をタイムリに措置することが最大のねらいである．進行状況の把握の方

表-2　システム開発計画書作成作業項目一覧[4]

No.	作業項目	作業内容
1	作業スケジューリング	作業項目の決定，作業量推定，作業レベル決定，要員／日程計画作成
2	ニーズの分析	要求仕様書の理解，資料収集，設計条件の把握
3	問題点の把握	実現のための技術レベル，難易度，信頼性，拡張性，安全性，オペレーション
4	システムの構想	目的，設計条件，対象業務，設計範囲，オペレーション，処理方法，機能分担，データフロー，性能
5	ソフトウェア仕様	機能構成，ファイル構成，入出力
6	ハードウェア仕様	コンフィギュレーション，機器仕様
7	見積り作業	要員工数，開発期間，システム構成と費用，ソフトウェア規模，ハードウェア設計／調達，テスト用設備／使用コンピュータ時間，教育／保守
8	ドキュメンテーション	システム開発計画書，内容／書式チェック，社内検討会

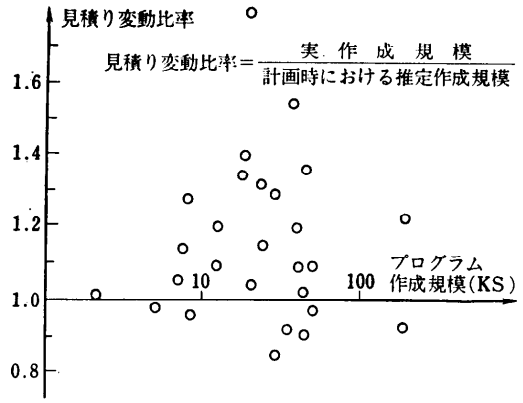


図-2　規模見積り変動比較[4]

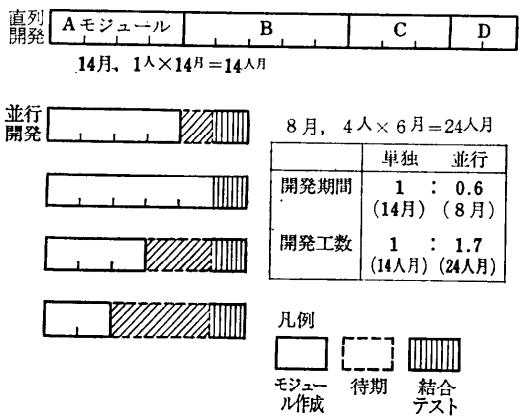


図-3　直列開発と並行開発[4]

法としては工程の終了時点，または一定周期（たとえば月単位，週単位）に作業状況を担当者から報告させる方法が最も多く用いられている．

工程管理の前提となる開発期間（工期）の設定は非常に難しい．開発期間を短縮するためには，一般に開発工数がかさむことが直感的に理解できるが，どの程度の期間の短縮がどの程度の工数増をまねくかの定量化の手法が確立していない．

図-3 に，工期の短縮が工数の増加を誘発する簡単なモデルを示す．つまり，並行開発の形態を採用すると担当者の作業にバラつきが発生し，その同期をとるために待機時間が発生するという仕組みになることが本質的な点である．現実の開発事例では，工期の短縮は多大の工数増加となるといわれている（表-3）．

ソフトウェアの並行開発の工数増加の主要な要因としては次のようなものが考えられる．

- モジュール間の調整作業の増加

表-3　開発期間と工数——SLIM の事例[4]——
開発規模：400 KS

	SLIM の出力	期間比	工数比
	年　人		
ケース 1	5.5× 25＝138	1	1
2	4.0×100＝400	0.7	3
3	3.4×200＝680	0.6	5

SLIM: Software Life-Cycle Management

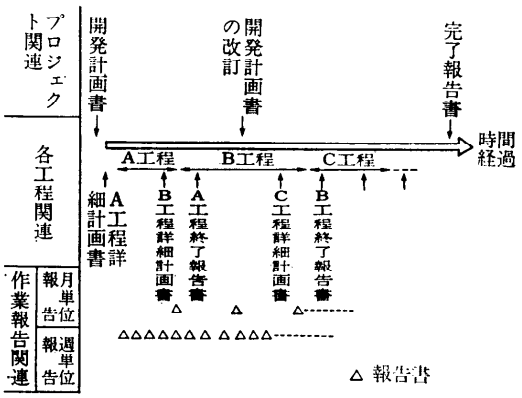


図-4　工程管理の体系例[4]

- モジュール間インタフェースの規定作業（直列開発より大）
- 修正にともなう他モジュールへの手戻り（直列開発より大）
- モジュール・テストにおけるスタブの作成（直列開発より大）

以上のように難しいことではあるが，開発期間の設定が済むと，開発計画に基づきソフトウェアの開発作業の進行状況を把握する．定期的に進捗管理を実施し，もし問題点が発見された場合にはタイムリな対処策を構ずる．必要ならば開発計画を見直すことになる．

一般には，プロジェクト，開発工程，月または週単位のレベルに分けて工程管理を行う．その事例を図-4 に示す．

週単位の報告は，重要なイベントが近づいたときとか作業の遅れが目立つような場合に実施する．通常は 2 週間単位，月単位の報告周期を採用している企業が多い．

4. 品 質 管 理

ソフトウェアの品質としてどのような特性要因を選択するのかまだ明確なコンセンサスは得られておらず，多様な提案がある．たとえば，Boehm は図-5 の

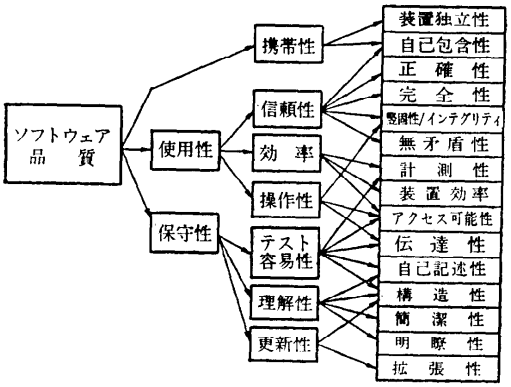


図-5　ソフトウェア品質の特性要因[1]

表-4　品質特性の重要感（アンケート）[2]

特性＼回答者	A	B	C	D	E	F	合計	平均
携帯性	5	5	1	4	2	3	20	3.3
信頼性	1	1	2	1	1	5	11	1.8
効率	3	3	7	7	7	1	28	4.7
操作性	2	2	5	2	3	2	16	2.7
テスト容易性	7	7	4	6	5	7	36	6.0
理解性	4	6	6	5	4	4	29	4.8
更新性	6	4	3	3	6	6	28	4.7

信頼性>操作性>携帯性>効率=更新性>理解性>テスト容易性

ように提案しており多数の特性要因を階層化したという点で高く評価される．しかし，特性要因数が多すぎて，どの特性要因を重要と考えるかについては個人差がある(表-4)．また，これらの特性要因には相反する性格のものがあるように思われる．

他方，ユーザの立場からすれば，目にみえないソフトウェアであればこそ，総合的な品質の評価がどのようなものであるかがいちばんの関心事である．したがって，家庭用品における JIS マークのように，購入しようとするソフトウェアに対して総合的な品質を保証するような規格の制定が望まれる．

近年，アメリカのマイコン用の流通ソフトウェア業界ではマイコン用ソフトウェアの評価を業とする人たち (evaluator) が存在し，流通されるソフトウェアにサイン入りの評価調書が添付されているという．我が国においても早くこのような仕組みが出現してくれることを期待する．

いずれにしても，ソフトウェアの品質としてどの特性要因に重点をおくかについては，そのソフトウェアの利用者の立場で考えて，プロジェクトの担当者全員に周知しなければならない．一般には，いわゆる魅力

表-5　当たり前の品質と魅力的品質の分類[1]

品質特性	当たり前の品質［主として担当者・経営者の立場からの評価］	魅力的品質［主としてユーザの立場からの評価］
携帯性		○
信頼性	○	
性能	○	
操作性		○
テスト容易性	○	
理解性		○
更新性		○

的品質がどの程度すぐれているかが商品価値を高める(表-5)ことに寄与する．しかし，現実にはソフトウェアの出荷当初はバグ（不良）が発生しユーザに迷惑をかけている事例が散見される．つまり，ユーザの思いどおりに動作しないという最も基本的レベルにおいて不十分であり，テスト技術などの開発・改良に対する期待も依然として大きい．

ソフトウェア品質の向上には，技術的側面だけからではなく，担当者の目的意識の高揚による効果も大きいことが認識されており，たとえば「バグ数の減少」，「使いやすいマニュアルの作成」，「ソフトウェア部分の利用率向上」などの身近なテーマを取り上げてボトムアップ的活動（これをソフトウェア QC と総称する）に展開している企業が増加しつつある．

5. 原価管理[5]

原価管理は，ソフトウェアの開発にかかわるすべてのコスト要因を的確に把握し，適正な運用が行われているかどうかをチェックするとともに，改善すべき事項があればそれに対するアクションをとらなければならない．

コストは少ないほど望ましいが，必要な経費を無理に削減するとどこかに歪みが生ずるので，原価管理はあくまでも適正なコストであるかを管理するのが基本である．

ソフトウェアのプロジェクトの運用における原価管理の対象としては，表-6 に示すようなものが主要な項目といえよう．原価管理の対象は大きく人件費(A)と物件費(B)に二分され，後者はさらにソフトウェアの開発に直接かかわる費用(C)とソフトウェアの開発にかかわる費用ではあるが，プロジェクトごとの経理が算出しにくい共通的経費(D)に区分される．

このうち，労働時間（工数）とコストの関係を推定するモデルとして COCOMO[6] (constructive cost

表-6　ソフトウェア開発における原価管理の対象[2),5)]

区分		内容
人件費（A）		①労働時間（ソフトウェアの開発時間） ②教育時間（文献調査，講習会含） ③会議などの時間 ④出張時間 ⑤社内検査，納入検査の立合 ⑥庶務的作業時間（就労管理，作業環境管理など） ⑦設備の運用費用
物件費（B）	ソフトウェア開発関連（C）	①ホストマシン設備使用量（コンピュータ，ファイル装置，入出力装置など） ②端末装置 ③通信設備 ④作業用机 ⑤消耗品
	共通的経費（D）	①局舎費用（光熱費など含む） ②ホストマシンなどの保守部品 ③厚生施設（休憩室，レクリエーション設備など） ④建物維持管理費 ⑤租税公課

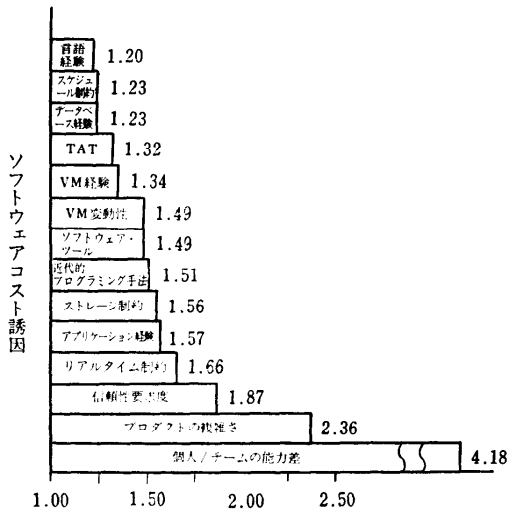


図-6　生産性レンジ

model）が著名である．COCOMO モデルの概要は以下のとおり（図-6，表-7）．

ステップ1　名目工数の算定

$P=3.2\ (KDSI)^{1.05}$

KDSI：Kilo Delivered Source Instruction

ステップ2　ソフトウェアコスト誘因の選定（15種）

ステップ3　努力係数の設定（5段階評価）

表-7　ソフトウェアツールの評価尺度と努力係数

評価尺度	ソフトウェアツール例	努力係数
非常に低い	・アセンブラ ・ベーシックリンカ	1.24
低い	・マクロアセンブラ ・オーバレイリンカ	1.10
基準値	・会話型デバグエイド ・会話型テキストエディタ	1.0
高い	・仮想メモリOS ・テストケースアナライザ	0.91
非常に高い	・統合ドキュメンテーションシステム ・プロジェクト管理システム	0.83

ステップ4　実開発工数　$P=P\cdot E_1\cdot E_2\cdots E_{14}$

$E_i(i=1\sim14)$：ソフトウェアコスト誘因別の努力係数

このモデルで使用される各種の係数，3.2，1.05，E_i などは各企業の開発環境や担当者のスキルに大きく依存しているため，このモデルを適用するためには自社の開発にかかわるデータによって，自社用の係数を求めること（キャリブレーション）が必須の手順である．すなわち，これらの数値をそのまま安易に流用するのは危険である．

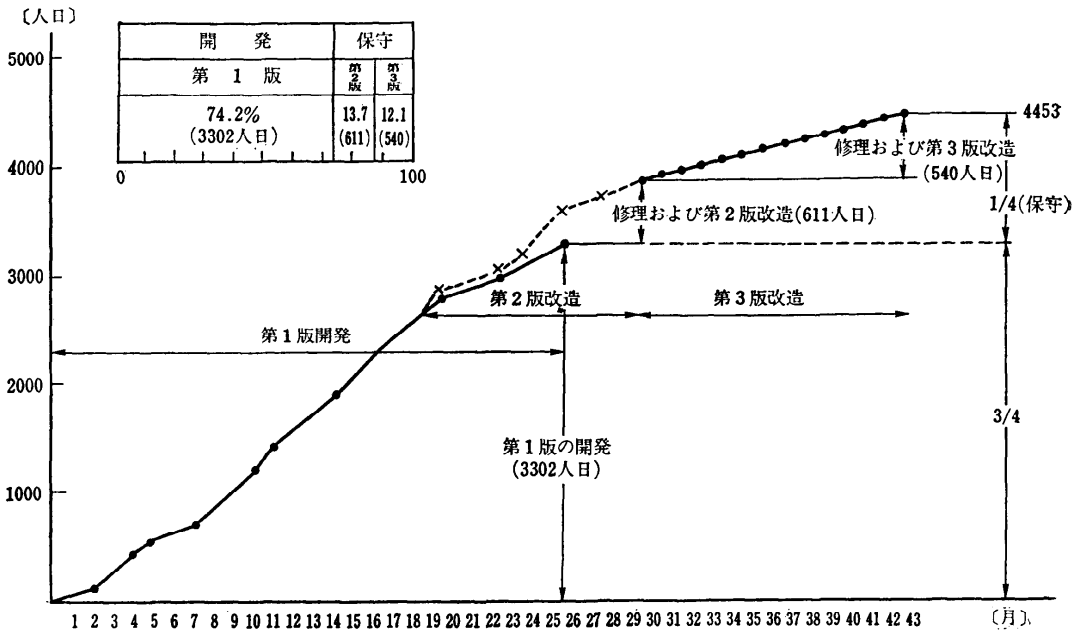
6. 構成管理[7)]

ソフトウェアのライフサイクルモデル（図-1）における運用・保守のフェーズにおいて，特に重要となるのが構成管理である．

Parnas の定義のところで述べたように，各企業では開発するソフトウェア製品を鮮度が損なわれないうちに市場に出荷するためには，継続して発生する顧客のニーズ（仕様）を一時的に凍結し，複数のバージョン（版）に分けて並行開発を進める．

その事例を図-7 に示す．この事例では第1版（初版）の完成前7カ月から第2版の開発（改造）に着手している様子がうかがえる．第2版の開発と並行して第1版の不良（バグ）の修理も行わねばならない．

このように複数バージョンを提供しなければならないもう一つの大きな背景は，ハードウェアの改良が急ピッチに行われている状況が関係している．コストパフォーマンスのすぐれたハードウェアが開発されるとそれをサポートするソフトウェアの開発が実施されるが，顧客側では旧式のハードウェアを捨てて新しい改良ハードウェアに乗り替えることはせずに，少なくとも償却期間が経過するまでは旧式ハードウェアを継続



注) FORTRAN コンパイラの第1版，第2版および第3版の実測値である

図-7　開発／保守工数の実測例（開発開始後43カ月までのデータ）

表-8　課題とその解決方法

課　題	課題の具体的内容	予想される問題	解　決　方　法
製品間の整合性情報の提供	個々のプログラムに製品の構成情報を付加する.	・情報の付加忘れ ・付加情報の設定ミス	・プログラムの構成を階層管理して開発者が取り扱うことができる単位を作成し，これに情報付加を行う.
変更間の整合性情報の提供	個々の変更に顧客先での変更を実施する順序を付加する.	同　上	・変更をとりまとめる単位を作成し，これに情報付加を行う. ・変更履歴の管理を行う.
提供物の検索	複数の版が存在する資源から，どの時点のどの資源を提供するかを判断する.	・変更資源の漏れ ・旧資源の提供 ・開発中資源の提供	・生産物と変更履歴の管理を行う. ・変更情報より提供物を自動生成する.
再利用による変更の同期	ある製品で行った変更を他の製品に反映する.	・変更資源の漏れ ・余分な変更の提供	・再利用プログラムの関係を管理する. ・再利用プログラムの変更を管理する.

使用する．場合によっては旧式と新改良式とを混在して使用することさえある．したがって，顧客に提供されるソフトウェアは顧客対応に異なる部分が存在し，どの顧客にどのソフトウェアを提供しているかを確実に管理しておかねば最新のファイルを定期的に送付するときに顧客の環境に合わないファイルを送りこむことになりかねない．

以上の事情により，ソフトウェア製品の構成管理の問題がクローズアップしてきた．解決せねばならない具体的課題としては以下のようなものがある（表-8）．

- 製品間の整合性情報の提供
- ソフトウェアの変更にともなう整合性情報の提供
- 提供物の検索
- 再利用による変更の同期

これらの管理を確実に遂行するためにはソフトウェア製品や顧客情報・変更履歴情報などをデータベース化して，支援ツールを準備し，顧客ごとにソフトウェア製品の出荷管理を行わねばならない．

7. 今後の課題

解決・解明すべき課題は多数存在するが，それらを一気に攻略するのは難しいと思われるので，重点化して取り組みたい．そのいくつかの候補をあげる．

（1） 開発データの標準化

ソフトウェアの開発を工学的あるいは科学的要素技術を適用しながら改善するには，大量かつ良質のソフトウェアの開発にかかわるデータの確保が必須条件である．このため，この種のデータを標準化し全企業が協力してデータを公開しあうことが急務である．それらのデータを過信することは危険であるけれどもデータに裏打ちされない開発管理技術は，たとえそれがすぐれた技術であっても普及速度がペースダウンしてしまう．アメリカの例では，この種のデータを開発費の10%程度の値段で買い取り，それを大学の先生方も動員して分析し，実践的技術の発掘に利用しようとしているという．

日本でもシグマプロジェクトにおいて種々の規約類の制定を検討していると聞くが，是非意慾的にチャレンジしてほしいものである．そして企業側では小異を捨て，シグマプロジェクトの規約類を積極的に採用すべきであろう(**表-9**)．

（2） 設計と製造の分離

表-9 ∑プロジェクトの規約類（仮）[9]

区　分	具体的規約例
管　理	・開発管理基準　・製品管理規約 ・品質管理規約　・保守管理規約 ・工程管理規約　・発注管理規約 ・費用管理規約　・購入管理規約
技　術	・開発作業基準 ・文書作成規約 ・レビュー規約 ・企画，計画作業規約 ・設計作業規約 ・製造作業規約 ・試験作業規約 ・保守作業規約
計算機設備	(a) 開発用計算機設備設置規約 (b) 開発用計算機設備運転・運用管理規約 (c) ∑センタ運転・運用管理規約

ハードウェアの世界では，ごく当たり前のこととなっている設計と製造フェーズの分離がソフトウェアの世界ではなぜできないのだろうか．この問題にチャレンジした企業がいくつかあると聞くが，あまりうまく進展していない模様である．

ソフトウェアの開発期間が2～3年に及ぶものも多数存在するので，設計・製造を分離できれば担当者はそれぞれの技術分野（設計，製造）について現在よりも約2倍の頻度で実践を積むことが可能になり技術の進展を加速させうる．

安くて良品質の工業製品を生産するには分業化し，分業化させた各フェーズの専門技術者が育つことが必須の条件ではないだろうか．

（3） ソフトウェアの再利用

類似のソフトウェアが多すぎるように思う．多少の不便を忍んでも既存のソフトウェアを再利用することを各企業とも積極的に取り組まねばならない．つまり，ソフトウェアの開発量を減らす努力である．オーダメイドに比較し既製服が安いことを知りながら，それでもオーダメイドの服しか着ないのは贅沢であり許されることではないと思う．

参考文献

1) Boehm, B. W.: Software Engineering, IEEE Trans. Comput. (Dec. 1976).
2) 藤野，花田：ソフトウェア生産技術，電子通信学会 (1985).
3) 花田編集責任：ソフトウェアの仕様化と設計，日科技連出版社 (1986).
4) 花田編集責任：ソフトウェアの計画と管理，日科技連出版社 (1987).
5) 花田：ソフトウェア生産性の評価と管理，情報処理，Vol. 21, No. 10, pp. 1057-1064 (1980).
6) 花田：開発管理技術の現状と課題，情報処理学会講習会テキスト，ソフトウェア工学の現状と動向，pp. 37-44 (1986).
7) 新田将人：構成管理ツール PRISM-X によるソフトウェア製品の品質管理，第5回ソフトウェア生産における品質管理シンポジウム発表報文集，pp. 131-138, 日科技連 (1985).
8) Boehm, B. W.: Software Engineering Economics, Prentice-Hall (1981).

(昭和62年4月7日受付)