解　説

ソフトウェア構成管理と保守管理†

松　尾　谷　徹††

1. はじめに

ソフトウェア開発や保守の生産管理技術としてソフトウェア構成管理 (Software Configuration Management：以下 SCM と称す) が見直され注目されている[1),2)].

SCM は不安定な構成要素をもつシステム，すなわち変更が発生する生産形態における管理技術である.

図-1 に構成管理の全体概要を示すが，SCM には二つの流れがある．一つは米国における大規模システム開発の購入者によって開発された，トップダウン的な管理手順である．二者間契約に基づいて，発注者と供給者が変更を合理的に取り扱うための規約である．図-1 の左側に相当するもので，これを規約 SCM と呼ぶことにする．すでに米国では ANSI/IEEE の標準として制定されており，現在，国際標準として ISO/IEC JTC 1 SC 7 において審議されている[3)~5)].

SCM は，品質保証の国際標準である ISO-9000 シリーズおよび，その認証制度の中でも，重要な役割として位置づけられている．今後，国内でも本格的に採用する企業が増えると考えられる[6)].

もう一つの SCM はプログラムの開発や変更を扱う，ボトムアップ的な技法である．これを技法 SCM と呼び，規約 SCM と区別する．技法 SCM は図-1 の右側に相当し，ライブラリ管理に端を発している．ソフトウェアは，多数の構成要素 (ソースモジュール，データ定義，マクロ，コンパイルオプション，など) を多数の作業者が関与して生産される．構成要素は多くの変更を受け，そのつどシステムの再組立が行われる．技法 SCM は，この混乱した生産活動を支援するツールと管理の枠組みである[7),8)].

規約 SCM は実現手段の一つとして技法 SCM の実施を義務づけており，両者は相反するものではない.

規約 SCM，技法 SCM 共に開発や保守のマネジメントにとって非常に有効なものである．ところが国内のソフトウェア開発現場に目を向けてみると，SCM の普及状況は思わしくない．この原因の一つは，我が国に米国の軍や NASA のような，SCM を推進する購入者が存在しなかったことがあげられる.

もう一つの原因として，管理者の変更に対する価値観の問題がある．つまり「仕様変更＝悪いこと，修正＝後戻り」であり，変更を合理的に取り扱う前に，変更をなくすべき，とする考え方が強い点である.

ソフトウェアの開発や保守におけるマネジメントは，大量生産における生産管理や品質管理とは異なったものである．ソフトウェアの生産形態において，変更をなくすことはできず，合理的に取り扱う必要があり，そのために SCM は重要な技術である.

本稿ではソフトウェアマネジメントの立場から SCM を紹介し，SCM が最も効力を発揮する保守管理についても概説する.

2. 規約 SCM

二者間契約によるシステム開発において，開発途中の変更は厄介な問題である．変更は品質の低下，納期や費用の増加を引き起こす．変更がもたらす価値と，悪影響を合理的に取り扱うため，供給者と購入者の間で約束ごとが必要である．このような背景から生まれたのが，規約としての SCM である.

† Software Configuration Management and Maintenance Management by Thoru MATSUODANI (1st C&C Systems Development Division NEC Corporation).

†† 日本電気(株)第一 C&C システム開発本部

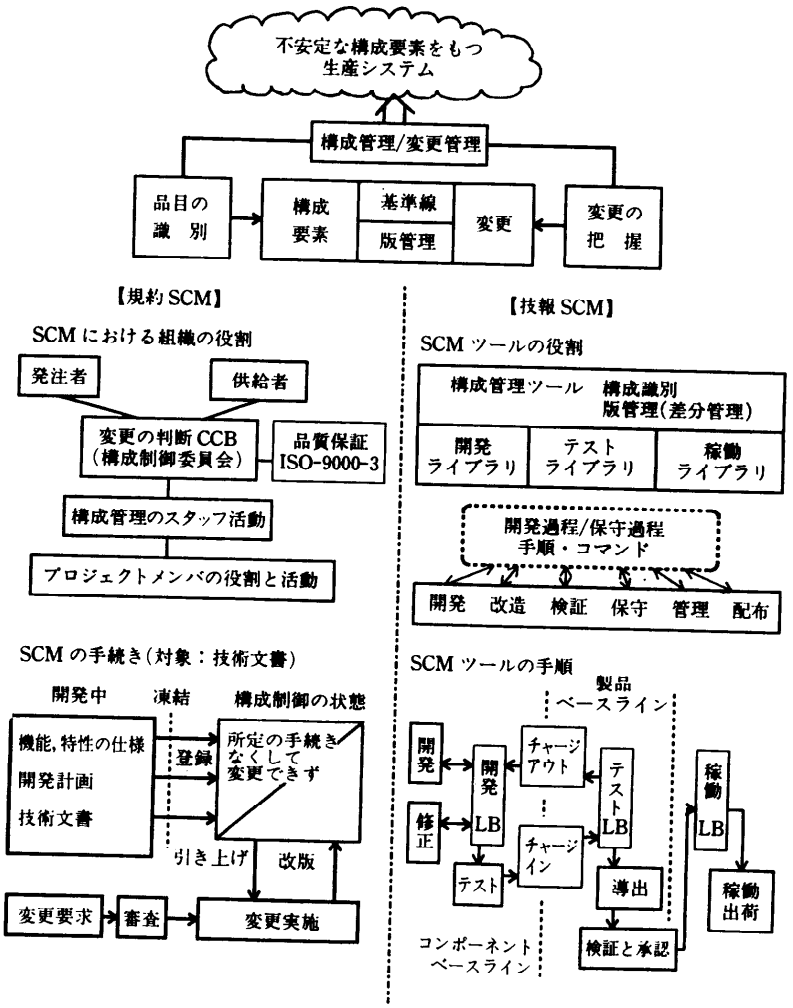

図-1 構成管理の全体概要

2.1 構成管理の歴史

まず構成管理の歴史的な背景を説明する．ハードを対象とするCM (Configuration Management) そのものが生まれたのは，東西冷戦状態にあった50年代の米国における兵器開発の分野であった．当時行われていた兵器開発のライフサイクルは，図-2に示すようなプロトタイピング (Prototyping) 型である．技術的な不安定要素を取り除くため，プロトタイピングを行い，変更の少ない安定した生産活動を行う方式である．この方式の問題はライフサイクルが長すぎる点である．どんなに急いでも一回の開発ライフサイクルに10数年を必要とした．

これでは「競合相手に追いつき，追い越す」という目標達成までに，何回も開発を行う必要がある．そこで一発逆転の開発方法として，図-3に示すような，開発を行いながら，かつ量産し，配備する方法がとられた．当然この方式は大量の変更が発生する．製品は生産される一方で変更され，製品一台一台が異なる構成と版をもつほど複雑な状況が発生する．

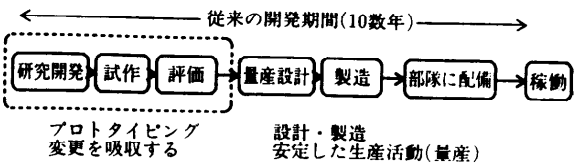

図-2 プロトタイピングによる開発ライフサイクル

この作りながら，変更を行う生産方式を支えたのがCMである．その後CMは民間にも広まり，新規性が高く，不安定な構成要素をもつが開

発期間が重要な分野の生産と保守の管理技法として，航空機やコンピュータの分野で広く使われている．

2.2 ソフトウェア変更の種類

ソフトウェアにおける変更は，ハードの場合と少し状況が異なる．図-4 に示すように生産活動の外側から発生する「外側の変更」と，内側から発生する「内側の変更」の2種類がある．

（1） 外側からの変更

ハードの CM が対象とする変更は，技術変更 (Engineering Change) と呼ばれるもので，設計時の技術的な不安定から生じている．ソフトウェアの変更は，要求自身の不安定からも生じており，要求仕様を初めあらゆる仕様の変更や追加が発生する．変更は開発費用や納期，システムの信頼性や安全性に大きな影響を与える．そのため供給者と購入者が変更を合理的に取り扱うための約束ごとが必要である．

（2） 内側からの変更

内側からの変更とはソフトウェア開発ライフサイクルに沿って大量に発生するバグ修正の類である．ソフトウェアの変更は直接的な損失，たとえば部品の廃棄や分解といった損失は少なく簡単に実施することができる．変更の問題は，変更前までに行ったテストの成果，すなわちテストとデバギングにより築き上げられたソフトウェアの信頼性を一瞬に崩す点である．この傾向はライフサイクルの後半ほど影響が大きく，変更をマネジメントの対象にする必要が生じる．供給者は，安全かつ確実にシステムを供給するため，内側からの変更を把握し制御しなければならない．

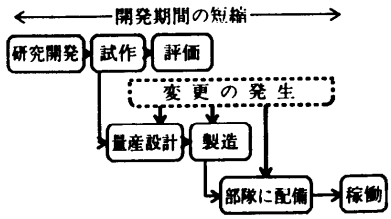

図-3 構成管理による開発期間の短縮

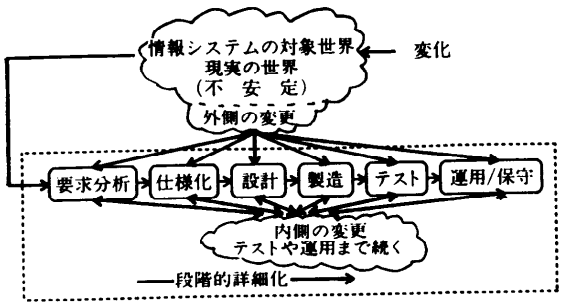

図-4 ソフトウェア開発における変更の種類

2.3 変更の把握

変更を把握するには，変更を受ける側の実体を把握する必要がある．これを SCM では品目 (Item) の識別 (Identification) と呼ぶ．ところが実体側も，開発ライフサイクルに沿って，仕様書からソースコードへと変わる．これでは，何が変更を受けているのか分からなくなる．図-1 の上部に示したように，基準線（ベースライン：Baseline）と版管理を用いて変更を把握する．

（1） 品目と要素 (Component)

品目の識別とは，対象とするソフトウェアが，どんな要素から構成され，何によってその特性やインタフェースが定義されているかを明らかにすることである．識別は階層構造で行われ，上位の単位を品目と呼び，品目を構成するものを要素と呼んでいる．具体的な品目には，プログラムやそれらを集めたサブシステムが対応する．そして仕様書やルーチンなどを要素と呼んでいる．品目と要素は識別名（または識別番号）により識別される．

（2） 基　準　線

品目は，開発ライフサイクルの進行とともに形態が変化する．たとえば機能仕様書→構造設計書→詳細設計書→モジュール→プログラムへと進んでいく．あたかも蝶が卵→幼虫→さなぎ→蝶へと変態していくように，工程により品目の形態と構成が大きく変化する．ライフサイクル上の進行と，変更を区別するため，ライフサイクル上の進行に対して管理上の区切りを設定する．この区切りを基準線と呼び，品目を構成する要素とその関係(インタフェースなど)を含め，公式に識別した塊として基準線を設定する．基準線が設定されると構成要素は第三者による構成管理の配下に置かれ，それ以降の修正は開発行為ではなく，変更として別の手続きにより管理される．

（3） 版 (Version)

版は基準線が設定された後に行われた変更の記録である．変更が許可され変更が実施されれば版を改版する．識別名（あるいは識別番号）

と版によりシステムの構成要素は一意に識別される.

2.4 SCM の実施

図-1 の左下,「構成管理の手続き」に示すように, SCM が実施されている状況を, 構成制御 (Configuration Control) された状態と呼んでいる.

(1) 構成制御

SCM で重要なことは, 構成制御された状態を維持することである. 構成制御された状態とは, 品目や構成要素が, それを直接担当する技術者の手から離れ, しかるべき管理下に移されることを意味する. この構成制御された状態に移すことを凍結 (Freeze) と呼ぶことがある. 凍結された状態とは変更を禁止することではない. 構成要素を修正するために, しかるべき変更の手続きが設定され, マネジメントの対象とすることである. 凍結された後は, 技術者が勝手に修正することはできない. これは変更をマネジメントするための手段である. 構成制御が行われている状態で, 変更管理 (Change Control) が可能となる.

(2) SCM 計画と組織

SCM は開発者や保守者に閉じたものではなく, 購入者やサブコントラクタなどシステム開発に従事する多くの人に関係する. SCM を確実に運用するためには, プロジェクトの初期段階で, SCM を行う役割と手続きを定義した SCM 計画が必要である. それぞれの組織やプロジェクトで個別に SCM 計画書を作成するのは, 産業界として大きな損失である. ANSI/IEEE の規格は, SCM の枠組みを SCM 計画書として提供する. ANSI/IEEE の SCM 計画書には以下の項目が含まれている.

- SCM を実施する組織と責任を明示する.
- 構成制御委員会 (Configuration Control Board : CCB) を設定し, 変更の審査を行う.
- 構成品目の範囲と基準線を決める.
- SCM 部門の業務計画を作る.
- 変更の承認に対する権限を決める.
- 用いる技法 SCM を決める.
- SCM に対する監査を行う.

などである.

図-1 の左側で紹介したように, 構成管理は CCB と SCM 部門によって運用される. SCM 部門は凍結された品目の構成制御を行い, CCB は変更を審査し, 変更の承認を行う.

3. 技法 SCM

規約としての SCM は, 開発の初期段階において, 契約や基準とともに SCM 計画が作られ, 運用される. 外側の変更を含め, 変更を大局的に捕らえ管理する枠組みである.

一方, 生産現場における変更の問題は, 仕様書の段階より, 作られたプログラムがより重要である.

3.1 プログラムの構成

プログラムは, 小さなソースモジュールに分割して作られる. 部品であるソースモジュールをコンパイラなどの開発ツールを用いてオブジェクトモジュールに変換し, 最終的なプログラムを組み立てる. 開発ツールを用いてソースモジュールからオブジェクトモジュールやロードモジュールを作り出すことを導出 (Derivation) と呼ぶ. 導出の過程は, 組立の過程でもある. プログラムの構成要素と導出の関係を**図-5** に示すが, さまざまな種類のものが存在している.

初期のソフトウェア開発においては, この組立の過程を作業者が個別に行っていた. しかし, 作業者の数が増えるとさまざまな問題を引き起こした. 修正したはずの修正がプログラムに反映されず, 古いものが組み込まれたりする問題である[8].

この問題を解決するため, ツールが作られ, ツールを基に構成管理の手続きが運用されるようになった. 図-5 に示した構成要素には次のような種類がある.

(1) 技術文書

開発の段階で作られたさまざまな技術文書類が含まれる. 計算機に格納された文章 (テキストファイル) もツールによる構成管理の対象に含まれる.

(2) ソースコード

プログラム言語で記述されたソースモジュール以外にも, さまざまな形態のソースが存在する. たとえばコンパイラに対するオプションや, データ構造体の定義を行うマクロソースなどがある.

(3) 開発ツール

開発ツール自身とその版も重要なプログラムの構成要素である. 開発ツールにはコンパイル時にインライン展開されるマクロや, 関数ライブラリなども含まれる.

（4） 導　出　物

ソースコードを基に，開発ツールによって作り出されたものが導出物である．導出物は対応するソースの版と対応がとれていなければならない．

（5） テストと品質保証

テストデータやテスト環境，およびテスト結果も構成要素である．変更が行われれば，それに対応してテストが行われなければならない．テスト結果も識別され版管理の対象となる．

3.2 プログラム修正手順

ライブラリは図-6 に示すように，通常3段階で構成され，開発ライブラリ，テストライブラリ，稼働ライブラリと呼ばれ，テストライブラリと稼働ライブラリが構成制御を受ける．開発ライブラリは開発者個人用の開発環境を提供するもので，同時に行われる他の開発者と競合しないよう設定される．

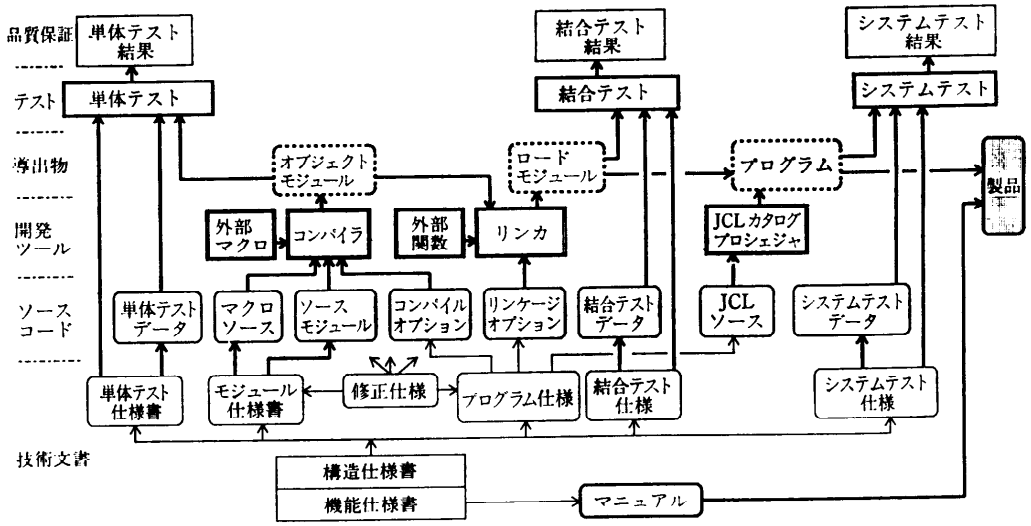

図-5　プログラムの構成例

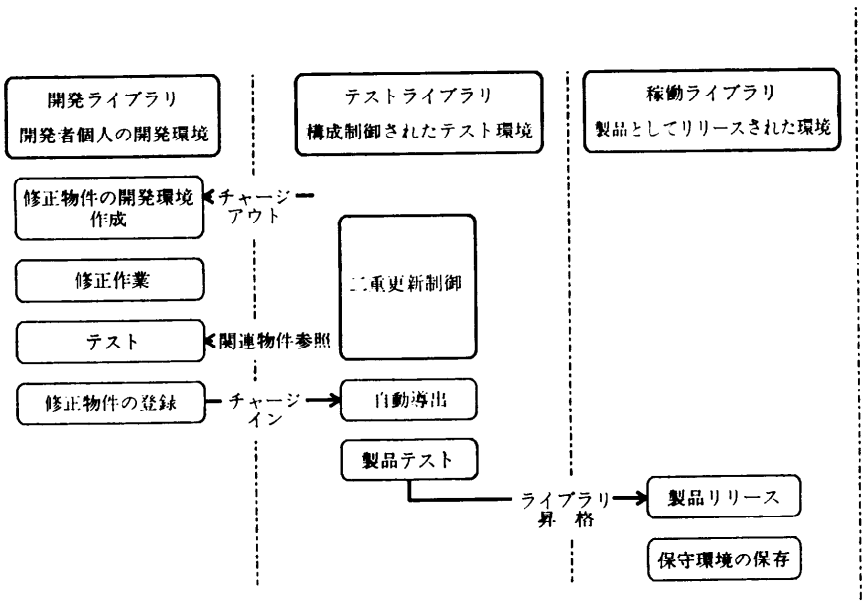

図-6　ライブラリ構造

（1） 修正の許可とチャージアウト (charge-out)

必要な変更が許可されると，担当者は当該構成要素（ソースモジュールなど）をテストライブラリから取り出し，開発ライブラリに複写する．取り出して修正が完成するまで，他の開発者が二重修正をしないようにライブラリからのコピーを禁止する．このような取り出しかたをチャージアウトと呼んでいる．

（2） 導出とテスト

しかるべき修正を行い，修正により再導出が必要なものを，一時的に開発ライブラリ上に再導出する．これはマクロを修正した場合に，それを引用しているソースモジュールを再コンパイルすることに相当する．導出したオブジェクトをテストし，修正の確認を行う．

（3） チャージイン (charge-in) と組立

開発ライブラリ上でテストに合格すれば，テストライブラリに戻入する．この戻入をチャージインと呼んでいる．チャージインは修正した構成要素についてのみ行う．その修正によって派生する再導出は，テストライブラリ上で行われ，その要素を含むベースラインが更新される．

（4） 稼働ライブラリへの昇格

テストライブラリ上では，別の変更と同期を取り，製品レベルのテストが行われる．テストの結果が公式に承認されれば，テストライブラリから稼働ライブラリへ複写される．マニュアルなど付帯する構成品目も含め製品ベースラインが設定される．

3.3 代表的なツール

技法 SCM を支えるのは，構成管理ツールである．古くから，いろいろなツールが提案され使用されている．

（1） make[8)~10)]

ソースモジュールを修正したならば，そのソースから導出されるオブジェクトモジュールやロードモジュールを再作成する必要が生じる．修正されたソースモジュールが，プログラム間で共用されるレコード定義であったなら，再導出すべき範囲は複雑で広範囲に及ぶことになる．再導出が必要なモジュールだけを人手によって選ぶことは難しい．make ツールはこの問題を自動的に解決してくれる．修正が行われたとき，make-file にあらかじめ記述された導出関係と，導出すべきファイルの修正日時（タイムスタンプ）を基に，必要な要素だけを再コンパイルあるいは再リンクする．

（2） SCCS/RCS[8),9),14),22)]

ソースモジュールの構成管理は，ライブラリに対してチャージイン/チャージアウトを行うことによって実現される．この代表的なツールとして SCCS (Source Code Control System) と RCS (Revision Control System) がよく知られている．これらのツールは，プログラムだけでなく，ドキュメントやデータも対象とする．導出されるオブジェクトやロードモジュールは対象とせず，make を起動することにより解決される．

SCCS や RCS は世代管理の機能をもっており，元本となる世代の情報と，その世代との差分情報によってライブラリの格納効率を向上させている．それぞれの版に対する経歴情報をもっており，チェックアウト後の二重更新ロックや，版の分岐の機能ももっている．

（3） LifeLine/LifeStudio[11),12)]

SCCS と make を用いた構成管理の欠点として，複数の作業者による同時更新と，任意の版の導出ができない点があげられる．この問題を解決するためにコンポジションと呼ぶ構成情報自身を記憶し，その版管理を行ったのが LifeLine である．この仕組みにより，複数作業者が同時に同一要素を変更する機能を提供している．

（4） Ada 言語システム[8),9),13)]

Ada言語システム (ALS : Ada Language System) は米国国防省が，プログラム言語 Ada を導入するために開発した構成管理機能を含む開発環境である．Ada によるプログラムはプログラムのインタフェースを記述するパッケージ仕様 (Package Specification) と処理を記述するパッケージ本体 (Package Body) に分かれている．Ada コンパイラは関係するパッケージ仕様間の厳密なチェックを行う機能を有している．Ada 言語システムはファイル構造をもっておりディレクトリ構造と世代管理の機能も含んでいる．このファイルシステムの特長は，ファイルに対するアソシエーション (association) と呼ばれるファイル間の関係情報を保持していることである．

4. 保守管理

ソフトウェア保守の目的は，システムの安定稼働や付加価値の向上，あるいはシステムの延命を行い，システム稼働によって得られる価値を高めることである．

ハードウェアと異なり，稼働中のシステムを変更しシステム価値を向上できることは，ソフトウェアの大きな特長である．しかし，その反面，ちょっとした保守上の誤りのため，大きな損失を発生させる危険性ももっている．

ソフトウェアの保守作業はマネジメント上も技術上もやっかいな問題である．保守作業の流れを単純化したものを図-7 に示す．保守作業の発生は大別して二つあり，一つは「事後保守」と呼ばれるもので，発生した障害に対応する作業である．もう一つは「計画保守」と呼ばれるもので，変更要求により変更や追加を計画的に行う，一種の開発作業である．

4.1 保守に必要な知識

保守が新規開発と比較してやっかいな理由は，対象となるソフトウェアについてあらかじめ理解しなければならない点である．保守のために必要な知識には，次の三つがある．

(1) プログラムのロジック構造

保守の対象となるプログラムの機能や内部構造や論理について理解する必要がある．

(2) プログラムの組立構造

図-5 に示したように，プログラムはソースコード以外にさまざまな構成要素から組み上げられている．これらの構成要素も保守に影響する．たとえば，使用したコンパイルオプションが不明であったり，用いたコンパイラ(開発ツール)の版が不明であったりすると，ソースプログラムを修正できても，組み立てたものが正しく動作しない．

昔，プログラムの修正はパッチと呼ぶ，プログラムのマシン語を直接修正する方法が用いられていた．この方法はプログラムの再組立を含まない保守方法である．そのため，修正の手間は大きいが，組立による誤りは生じない．

現在，保守をパッチで行うことはまれである．ソースコードで保守を行うために，再組立の環境を保存し，それを理解することが必須である．

(3) プログラムのテスト構造

保守に費やされる費用の中で，テスト，それも改造が他へ悪影響を与えないか否かのテストが，大きな割合を占めている．このテストをテストの流用として実施するか，あるいは改造ごとに新しくテストの設計から行うかでは大変な違いである．

テストを流用するには，テスト環境と以前のテスト結果を保存し，それを理解する必要がある．

4.2 保守における構成管理の役割

保守は，開発の出来映えに依存することは明らかである．開発物の何が保守に必要なのであろうか．直感的には欠陥が少なく理解しやすいプログラムであり，コメントの挿入など 4.1 の(1)のように思われるが，(2)や(3)も重要である．(2)や(3)の問題は構成管理が確実に実施されていなければ解決されない．確実に実施されるということは，マネジメントの配下で SCM が計画され，役割分担が行われ，コントロールされなければな

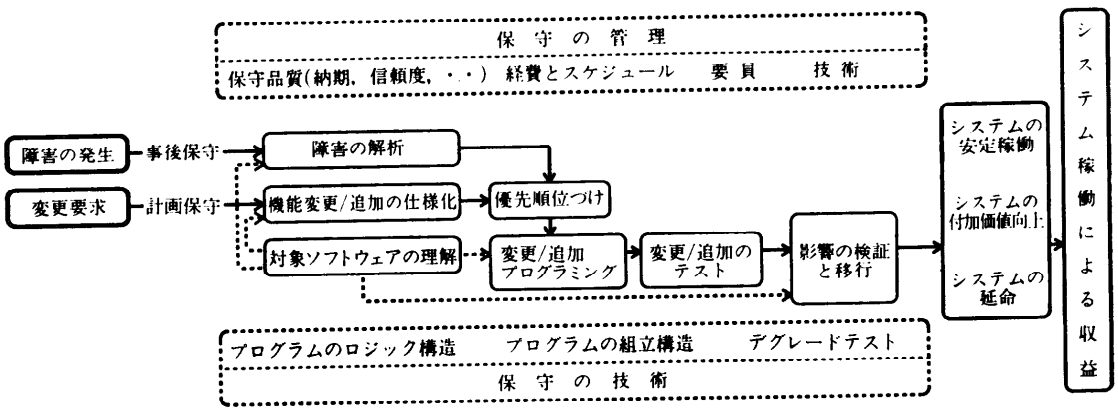

図-7 保守作業の流れ

らない.

開発と保守は，同一の組織が行うとは限らない．外部に発注されたソフトウェアの場合，購入者が保守を行わざるをえないのが現状である．そうすると，保守において重要な役割を果たす，開発時の構成管理のやり方が，まちまちであるのは購入者にとって大きな損失である．ANSI/IEEEの標準やISO/IECの国際標準として制定される必要性はここにもある．

5. おわりに

ソフトウェア構成管理について説明を行った．SCMは，変更を抑圧するためのものではなく，変更を合理的に取り扱うための管理技法である．

マネジメントの立場からみた構成管理の意義は，開発者と担当物件の分離，第三者による成果物の体系的管理，開発行為と変更行為の分離などを行うことである．そして，開発行為に対する管理以上に，変更に対し重点をおいた管理を行うことである．

そのための具体的な方法として，関係者間の手続きとしての機能（規約構成管理）と，プログラム開発におけるプロダクト管理の機能（技法構成管理）とがある．

SCMはソフトウェアにおける重要な問題，たとえばソフトウェアの購買や契約における変更の取扱い，分散開発環境における統制の問題，保守の問題，などの解決になくてはならないものである．しかし，SCMがソフトウェアマネジメントの基盤として定着するためには，まだまだ問題も多い．この分野における現場での活発な展開を期待したい．

参考文献

1) 森口繁一編：ソフトウェア品質管理ガイドブック，日本規格協会 (1990/7).
2) GE編 井上義裕他訳：ソフトウェア工学ハンドブック，マグロウヒル (1988).
3) ANSI/IEEE Std 828, IEEE Standard for Software Configuration Management Plans, IEEE (1984 および 1990).
東 基衛監修，ANSI/IEEE ソフトウェア規格集，日本規格協会 (1988).
4) ANSI/IEEE Std 1042, IEEE Guide to Software Configuration Management, IEEE (1988).
東 基衛監修，IEEE ソフトウェア規格集第2集，日本規格協会 (1991).
5) DoD 5200. 28-STD, Department of Defence Trusted Computer System Evaluation Criteria, Office of Standards and Products, Maryland (1985).
6) 飯塚悦功編：ソフトウェアの品質保証，日本規格協会 (1990).
7) Parikh, G.: Handbook of Software Maintenance, John Wiley & Sons. Inc (1986).
テクノピック訳，ソフトウェア・メンテナンス・ハンドブック，啓発出版 (1989).
8) Babich, W. A.: Software Configuration Management Coordination for Team Productivity, Addison-Wesley, New York (1986).
菊池豊彦訳，プログラムのチーム開発入門，CQ出版社 (1988).
9) Whitgift, D.: Methods and Tools for Software Configuration Management, John Wiley, Chichester (1991).
10) Feldman, S.: Make-A Program for Maintaining Computer Programs, Software Practice and Experience, 9(3), pp. 255-265 (1979).
11) 坪谷英昭，岸 知二他：構成・版管理ライブラリLifeLineを用いたCプログラム開発環境，情報処理学会，ソフトウェア工学研究会，SE-73 (14), pp. 107-114 (1990).
12) 池田健次郎，岸 知二他：共同開発時におけるインテグレーション支援機能，情報処理学会，ソフトウェア工学研究会，SE-79(3), pp. 1-8(1991).
13) Booch, G.: Software Engineering with Ada, Benjamin/Cummings, Redwood City, CA (1987).
14) Rochkind, M. J.: The Source Code Control System, IEEE Transactions on Software Engineering, SE-1(4), pp. 365-370(1975/12).
15) Knudsen, D. B., Barofsky, A. and Satz, L. R.: A Modification Request Control System, Proceeding of the 2nd International Conference on Software Engineering, pp. 187-192, IEEE and ACM, New York (1977).
16) Bersoff, E. H., Henderson, V. D. and Siegel, S. G.: Principles of Software Configuration Management, Prentice-Hall, Englewood Cliffs, NJ (1979).
17) Lampson, B. W. and Schmidt, E. E.: Organizing Software in a Distributed Environment, Proceeding of the SIGPLAN 83 Symposium on Programming Language Issues in Software Systems, San Francisco (1983).
18) ANSI and AJPO, Military Standard, Ada programming Language, ANSI/MIL-STD-1815 A (1983).
19) Kruskal, V.: Managing Multi-Version Programs with Editor, IBM Journal of Research and Development, 28(1), pp. 74-81 (1984).
20) Bersoff, E. H.: Elements of Software Configuration Management, IEEE Transaction on Software Engineering, Jan. 1984, pp. 79-87 (1984/1).
21) Goldfine, A. and Konig, P.: A Technical Overview of the Information Resource Dictionary

System, Computer and Standards, 1, pp. 153-208 (1985).
22) Tichy, W. F.: RCS—A System for Version Control, Software-Practice and Experience, 15 (7), pp. 637-654 (1985/7).
23) Tichy, W. F.: Smart Recompilation, ACM Transactions on Programming Languages and Systems, 8(3), pp. 273-291 (1986/8).
24) Dowson, M.: Integrated Project Support with IStar, IEEE Software 4(6), pp. 6-15 (1987).
25) Lablang, D. B. and Chase, R. P.: Parallel Software Configuration Management in a Network Environment, IEEE Software, 4(6), pp. 28-35 (1987/11).
26) Tichy, W. F.: Tools for Software Configuration Management, Proceeding of the International Workshop on Software Version and Configuration Control, pp. 1-20, Teubner Verlag, Stuttgart (1988).
27) Reps, T., Horwitz, S. and Prins, J.: Support for Integrating Program Variants in an Environment for Programming in the Large, Proceeding of the International Workshop on Software Version and Configuration Control, pp. 197-216, Teubner Verlag, Stuttgart (1988).
28) Clemm, G. M.: The Odie Specification Language, Proceeding of the International Workshop on Software Version and Configuration Control, pp. 144-158 (1988).
29) Montes, J. and Haque, T.: A Configuration Management System and More!, Proceeding of the International Workshop on Software Version and Configuration Control, pp. 217-227, Teubner Verlag, Stuttgart (1988).
30) Winkler, J. F. H. and Stoffel, C.: Program-Variations-in-the Small, Proceeding of the International Workshop on Software Version and Configuration Control, pp. 175-196, Teubner Verlag, Stuttgart (1988).
31) Sarnak, N., Bernestein, R. and Kruskal, V.: Creation and Maintenance of Multiple Versions, Proceeding of the International Workshop on Software Version and Configuration Control, pp. 264-275, Teubner Verlag, Stuttgart (1988).

(平成4年5月14日受付)

松尾谷 徹 (正会員)

1948年生. 1972年関西大学大学院修士課程修了. 同年, 日本電気(株)入社. 以来ハードウェアの開発を経て基本ソフトウェアの開発及び品質保証, 開発支援システム SEA/I の開発, 問題プロジェクトの支援など多くのソフトウェア開発の現場に従事. その間, システム統合技術や検証技術など下工程のソフトウェア工学に興味をもつ. 現在, 同社 C&C 第一システム開発本部ソフトウェア生産技術部技術課長, 電子情報通信学会, 経営情報学会各会員. 情報規格調査会 SC 7 委員.