ソフトウェア工学 107－8
（1996．1．18）

ソフトウェア共同開発におけるバージョン管理支援

服部真穂　松尾朗　貫井春美
株式会社 東芝　研究開発センター
システム・ソフトウェア生産技術研究所　開発第三部
〒183 東京都府中市片町 3-22
maho@ssel.toshiba.co.jp

ソフトウェアの共同開発において、ソフトウェア成果物のバージョンの不一致を原因とする後戻り作業は、少なくない。その原因の中には、開発者間の成果物の影響範囲を管理できていなかったり、管理はできているが、連絡が徹底していなかったりといったことがある。

報告者等は、ソフトウェア共同開発支援システム D^2 (D Square: Distributed environment for software Distributed development) の適用を進める中で、バージョンが更新される際の一連の作業に着目し、これをモデル化することで、成果物のバージョンの整合性を保つといったアプローチを提案している。実際に社内複数部門の開発形態を分析し、バージョンの更新に着目した共同開発作業モデルを作成し、そのモデルに基づき、電子メールを応用したバージョン管理支援システムを開発し、適用している。

本報告では、共同開発モデルと支援システムに関して説明し、その適用評価を報告する。

Support for Version Control on Software Development

Maho Hattori, Akira Matsuo, Harumi Nukui
Systems & Software engineering. Laboratories,
R & D Center, Toshiba Corp.
3-20 Katamachi, Fuchu, Tokyo, 183, Japan

For the cooperative software development, it is important to maintain the same version of software between different sites. When a new version of software is released at one site, the other sites must be notified. Otherwise, there will be discrepancy among the versions of source code at the various sites.

To solve this problem, we try to apply D^2 (D Square: Distributed environment for software Distributed development)to cooperative software development. In this approach we focus the behavior when a new version of software is released, then it's behavior define as workflow model.

The version control support system that we developed is used electronic mail and perform version control based on that model.

In this report, we describe the following.

- A development model and a version control support system based on that model.
- A evaluation of the cooperative development applied to the version control support system.

1 はじめに

ソフトウェア共同開発において、ソフトウェア成果物のバージョン不一致を原因とする不具合や、それによる後戻り作業は少なくない。後戻り作業が発生すると、不具合を修正するために過去のバージョンに遡る作業が発生する。この時、成果物のバージョン管理が徹底していないと、過去バージョンの再現がスムーズに行われない[1][2][4]。

報告者等は、これらの問題を解決するために、バージョンの更新に伴う一連の作業の流れをモデル化し、そのモデルに従って支援することを提案している。バージョン管理支援システムは、D^2 (D Square: Distributed environment for software Distributed development) の一機能であるエージェントメールシステム[3]を応用している。エージェントメールシステムは、一定のルール (作業手順) をスクリプトに定義しておくことにより、ルールに従って電子メールメッセージの自動処理を行うシステムである。作業モデルをスクリプトでルール化することにより、一連の流れを支援する。

本稿では、ソフトウェア共同開発における開発モデル、それに基づいて作成したバージョン管理支援システムに関して説明し、システムの適用評価を報告する。

2 ソフトウェア共同開発の問題点

図1は、ソフトウェア共同開発における、問題点を示している。

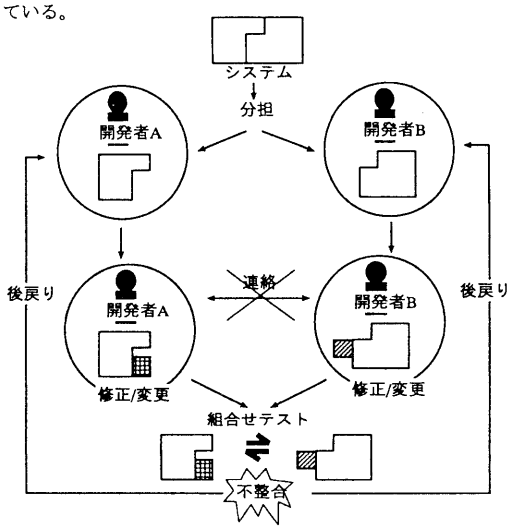

図 1: ソフトウェア共同開発の問題

(1) 変更の影響範囲が把握できないため、開発者 A が変更した内容が、開発者 B に伝わらない。

(2) 開発者 A が変更した内容を知らずに、開発者 B はそのまま作業を続けてしまう。

(3) 組合せテスト時に、各々の成果物のバージョンが合わず、再び製造工程に後戻りする。

(4) 後戻り作業が発生した場合、過去の作業環境を再現がスムーズに行われない。

これらの問題点は、成果物の複製が開発者ごとに存在していたり、開発者が、成果物の修正が相互に影響しあう範囲を、正確に把握できていないといったことに起因する。

3 解決へのアプローチ

報告者等は、前節に挙げた問題点を解決するため、以下のアプローチをとる。

- 成果物の一元管理
  開発者間の成果物を一箇所に集め、それをマスターファイルとして共通に管理する。開発者やテスト担当者が、製造作業やテスト作業を行う場合は、必ずマスターファイルを用いるようにする。

- 成果物の構成管理
  成果物間の相互の影響範囲と、それに関係する開発者の構成を管理する。

- 修正や変更と同期した通知
  ある成果物に修正や変更が発生した場合には、構成管理情報に従って、その影響範囲にある開発者への通知を行う。

- 変更履歴の管理
  修正や変更に関する情報を明確に記録し、バージョン管理を徹底する。

4 ソフトウェア共同開発のモデル化

ソフトウェア共同開発をバージョン管理に着目し、モデル化する。

4.1 基本モデル

問題解決のアプローチに沿った、最も基本的なモデルを図2に示す。

(1) サブシステムに分割し、各開発者や開発グループに分担する。分担決定後は、各開発者にて開発を進め、単体テストまで行う。

(2) 単体テストが終了した時点で、各開発者の成果物を共通のマスターファイルに登録する。

(3) マスターファイル登録後は、関係者に通知を行う。

(4) サブシステムがすべてマスター登録された段階で、できる範囲で組み合わせテストを開始する。

(5) 修正や変更が発生した場合には、マスターから成果物を取り出し、修正する。
(6) 修正の際には、関係者に通知を行う。
(7) 修正が終了したら、成果物をマスターファイルに再登録する。マスター登録と同時に、変更履歴を記録する。

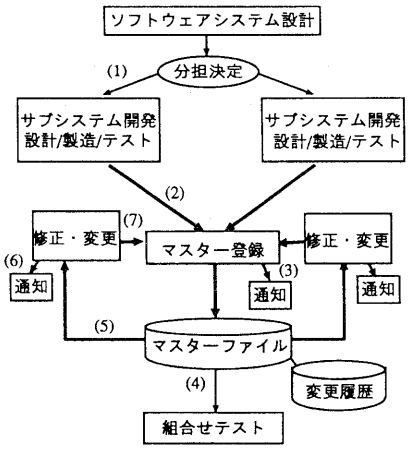

図 2: 共同開発モデル

ここで示した基本モデルは、

- 作業の流れと役割担当者
- 成果物の管理方式

に着目し、開発部門や対象製品にカスタマイズすることが必要となる。

以降に、報告者等がいくつかの開発形態を調査し、抽出したカスタマイズのパターンを説明する。

4.2 作業の流れと役割担当

作業の流れと役割担当に着目し、3種の開発フローのパターンを抽出した。

※ 文中のバージョン番号は、サブシステム単位に付ける番号を示し、リビジョン番号は、ファイル単位に付ける番号を示す。

4.2.1 パターン 1

以下のような開発形態には、図3に示す開発フローを適用する。

- 開発規模: サブシステムを複数人で担当する
- バージョン管理の単位: サブシステム単位
- 役割担当: テスト担当者が、版管理作業のすべてを行なう
- 組合せテスト: 組合せテスト中はファイルをロックし、修正を禁止する

図3のフローでは、

1. 開発者が修正 / 変更を行なうためにサブシステムを取り出し、作業する。
   この時、成果物にはロックが掛けられ、排他制御される。
2. テスト担当者に登録依頼をする。
3. テスト担当者が組合せテストを行なう。
4. テスト担当者がサブシステムをバージョン登録する。
   この時、成果物のロックがはずされる。

といった手順をとる。

ここでは、次の開発ルールを適用する。

- サブシステムに対し、複数の開発者が修正 / 変更を繰り返すことで、システム全体から見た場合、ファイル個々の管理は複雑である。従って、取り出しや修正の単位をサブシステム単位とする。
- 組合せテストや登録は、テスト担当者が、修正が発生した都度、必ず行なう。
- テスト担当者がサブシステム間の組合せテストを完了し、バージョン登録するまでは、開発者がそのサブシステムを修正することを禁止する。

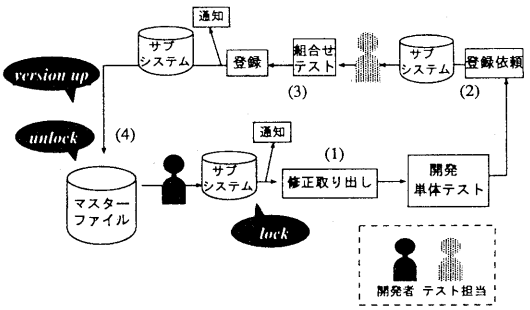

図 3: 開発フロー 1

【開発フロー 1 の説明】

(1) 取り出し作業
開発者は、サブシステム単位に共通のマスターファイルから成果物の取り出しを行う。
この時、マスターファイルにはロックをかけ、他開発者からの取り出しを排他する。
取り出したサブシステムを用いて、開発者は、開発・単体テストを行う。

(2) 登録依頼作業
開発者は修正した成果物を直接登録せず、テスト担当者に登録依頼を出す。

(3) テスト作業
登録依頼を受けたテスト担当者が、サブシステムごとに組合せテストを行う。

(4) 登録作業
テストの結果がOKのものだけ、テスト担当者がマスターファイルに登録を行う。
登録と同時にマスターファイルのロックをはずし、サブシステムのバージョンを更新する。

4.2.2 パターン2

以下のような開発形態には、図4に示す開発フローを適用する。

- 開発規模: サブシステムを少人数で担当する
- バージョン管理の単位: ファイル単位及びサブシステム単位
- 役割担当: 開発者は、ファイルのリビジョン管理を行ない、テスト担当者はサブシステムのバージョン管理を行なう
- 組合せテスト: 組合せテスト中は、ファイルをロックし、修正を禁止する。

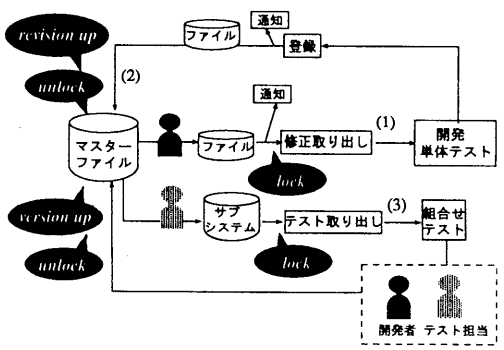

図 4: 開発フロー2

図4のフローでは、

1. 開発者が修正/変更を行なうためにファイルを取り出し、作業する。

   この時、成果物にはロックが掛けられ、排他制御される。

2. 開発者がファイルをリビジョン登録する。

   この時、成果物のロックがはずされる。

3. テスト担当者がサブシステムを取り出し、組合せテストを行なう。

   この時、成果物にはロックが掛けられ、排他制御される。

4. テスト担当者がサブシステムをバージョン登録する。

   この時、成果物のロックがはずされる。

といった手順をとる。
ここでは、次の開発ルールを適用する。

- 修正/変更時の取り出しや登録は、ファイル単位でできるようにする。
- 修正/変更の取り出し作業から、登録作業までは、開発者が続けて行なえるようにする。この時、登録されたファイルは個々にリビジョン更新していく。
- テスト担当者が組合せテストを完了するまでは、開発者がその関係成果物を修正することを禁止する。

【開発フロー2の説明】

(1) 取り出し作業
開発者は、共通のマスターファイルからファイルの取り出しを行う。
この時、マスターファイルにはロックをかけ、他開発者からの取り出しを排他する。

(2) 登録作業
開発者は、修正したファイルをマスターファイルにリビジョン登録する。
登録と同時にマスターファイルのロックをはずし、登録されたファイルのリビジョンを更新する。

(3) テスト作業
テスト担当者が、開発の状態を見て、マスターファイルからテストの単位でテスト取り出しを行なう。この時、マスターファイルにはロックをかけ、他開発者からの取り出しを排他する。

テスト結果がOKの場合、マスターファイルのロックをはずし、サブシステムのバージョン登録する。

4.2.3 パターン3

以下のような開発形態には、図5に示す開発フローを適用する。

- 開発規模: サブシステムを少人数で開発する
- バージョン管理の単位: ファイル単位及びサブシステム単位
- 役割担当: 開発者は、ファイルのリビジョン管理を行ない、テスト担当者はサブシステムのバージョン管理を行なう
- 組合せテスト: 組合せテスト中にも、結果をフィードバックし、修正を許可する

図5のフローでは、

1. 開発者が修正 / 変更を行なうためにファイルを取り出し、作業する。

   この時、成果物にはロックが掛けられ、排他制御される。

2. 開発者がファイルをリビジョン登録する。

   この時、成果物のロックがはずされる。

3. テスト担当者がサブシステムを取り出し、バージョン登録し、組合せテストを行なう。

といった手順をとる。

ここでは、以下の開発ルールを適用する。

- 修正 / 変更時の取り出しや登録の単位は、ファイル単位でできるようにする。
- 修正 / 変更の取り出し作業から、登録作業までは、開発者が続けて行なう。この時、登録されたファイルは個々にリビジョン更新していく。
- テスト担当者が組合せテストを行っている間も、結果をフィードバックし、開発者への修正を許可する。

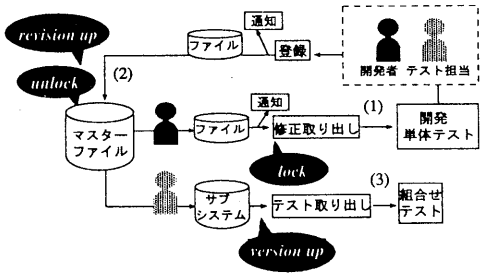

図 5: 開発フロー 3

【開発フロー 3 の説明】

(1) 取り出し作業

開発者は、共通のマスターファイルからファイルの取り出しを行う。

この時、マスターファイルにはロックをかけ、他開発者からの取り出しを排他する。

取り出したファイルを用いて、開発者は、開発・単体テストを行う。

(2) 登録作業

開発者は、修正したファイルをマスターファイルにリビジョン登録する。

登録と同時にマスターファイルのロックをはずし、登録されたファイルのリビジョンを更新する。

(3) テスト作業

テスト担当者が、開発の状態を見て、マスターファイルからテストの単位で取り出しを行う。この時、マスターファイルにはロックをかけず、他開発者からの取り出しが可能な状態とする。

バージョンの更新は、テスト取り出しの際に行なう。よって、テストで NG の場合も、バージョンだけは更新されていく。

4.3 成果物の管理方式

ソフトウェア共同開発において、インクルードファイル、ライブラリ、データ等、開発者やグループ間で共通するファイルは多くある。また、機種展開が多い製品のソフトウェアでは、機種間で共通するファイルも多く存在する。

本節では、複数開発者や複数機種間で共通するファイルの管理方法に関して述べる。

4.3.1 共通ファイルの管理

複数の開発ディレクトリで共通するファイルが多いと、図 6に示すように、一つのファイルを修正したことによる影響範囲は次第に広がっていく。

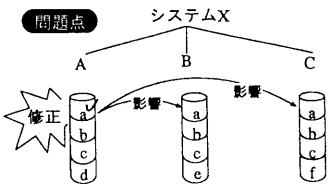

図 6: ファイルの相互関係

図 6の説明

あるシステム X には、A、B、C の開発ディレクトリが存在している。A のディレクトリ以下には、ファイル a, b, c, d が存在している。ファイル a は、B や C にも共通するファイルであり、それぞれのディレクトリ下にも存在している。

従って、A でバグが発見され、ファイル a を修正した場合、B や C のディレクトリ以下にあるファイル a を更新しなければならない。

しかし、実際には開発者や機種の数は多く、全影響範囲に修正を反映させることは大変な作業である。

このファイルの相互の影響範囲を管理するために、複数の開発ディレクトリで共通するファイルに関しては、その

実体を複数のディレクトリに分散させるのではなく、共通ファイル用のディレクトリ以下に置き、各開発ディレクトリからは共通ファイル用のディレクトリを参照するようにする (図 7)。

この方法により、一箇所でバグが発見され修正や変更を行う時、それが共通ファイルであるならば、共通ディレクトリ以下のファイルを修正することで、関係する全箇所から、そのファイルの修正が見えるようになる。

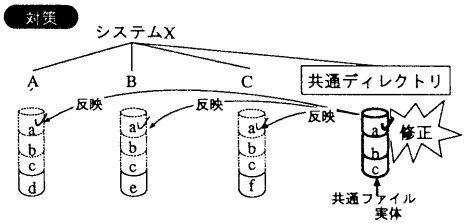

図 7: 共通ファイルの管理

図 7の説明

この図も、図 6と同じシステム X の例であり、ファイル a が、A、B、C に共通であることに変わりはない。しかし、共通するファイル a は、共通ディレクトリの下に置き、各ディレクトリ A、B、C 以下からは、共通ディレクトリにある a を参照する構成をとっている。
従って、A でバグが発見され、ファイル a を修正した場合、共通ディレクトリ下にあるファイル a を更新するだけで、全影響範囲に修正を反映させることができる。

4.3.2 枝分かれ管理

先に述べたように、複数の開発で共通のファイルは、共通ディレクトリ以下で管理するが、開発の進度に従って、図 8のように共通ファイルを枝分かれさせていく方法をとる。すなわち、ファイルを共通ディレクトリ以下で共通管理している間は、どこで誰が変更したものも反映させる。そして、出荷等に伴い、任意に共通ファイルを共通管理の枠からはずす必要が出た場合、それまでの共通ファイルから専用の枝を作成する。専用の枝が作成された後は、他の開発で発生した変更は反映させずに独立に管理していく。

5 バージョン管理支援システム

本節では、報告者等の開発したバージョン管理支援システムの概要とシステム構成に関して述べる。

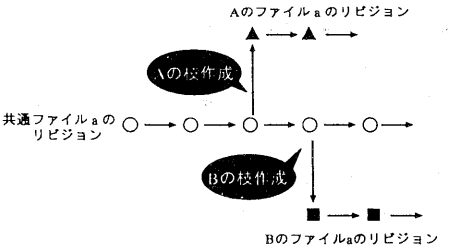

図 8: 共通ファイルの枝分かれ管理

5.1 バージョン管理支援システムの概要

成果物の管理やバージョン統一のためには、バージョン管理ツールが存在している。これらにメッセージ交換システムをリンクさせれば、通知の自動化が図れ、修正や変更と同期をとった通知が可能になる。その際、メッセージ交換システムに、開発フローに沿った動作をさせることが必要となる。

このような理由から、エージェントメールシステムと既存のバージョン管理ツール (RCS [1]) を組み合わせたバージョン管理支援システムを開発した。エージェントメールシステムとは、非同期型コミュニケーション支援を行なうシステムで、メールで交換するメッセージに対する諸操作を自動的にバックグラウンドで行なうものである。開発フローに相当する処理をスクリプトファイルに記述することにより、開発フローに沿った自動処理をコンピュータが実行していく。

開発フローに従ったルールをスクリプトファイルに記述しておくことにより、開発フローに沿った処理が可能である。

このバージョン管理支援システムでは、管理する対象成果物をソースコードとし、各々のソースコードを差分により、実体を管理している。

5.2 バージョン管理支援システム構成

図 9にバージョン管理支援システムの構成を示す。

開発者やテスト担当者は、何らかのトリガにより、ある成果物に修正や変更が発生した場合、バージョン管理サーバに成果物取り出し要求を電子メールで送り、成果物を取り出す。成果物を修正した後で、開発者やテスト担当者はバージョン管理サーバに成果物登録要求を電子メールで送る。

バージョン管理サーバは、取り出し要求や登録要求を受けた場合、エージェントメールシステムにより要求内容を解析する。更に、スクリプトファイルに記述された開発フローのルールに従って、バージョン管理ツールで、一元管

[1] Revision Control System(フリーソフトウェア)

理される成果物のマスターファイルから、必要な成果物を取り出したり、登録したりする。取り出された成果物に対しては排他制御を行ない、他の開発者やテスト担当者が同時に同一の成果物にアクセスすることを防ぐ。また、取り出しや登録が行なわれた成果物の関係者を構成情報から抽出し、該当者に電子メールで通知を送る。

更にバージョン管理サーバは、バージョン管理ツールにより蓄積される変更の履歴情報 (いつ、誰が、何を変更したか) を開発者の必要に応じて電子メールで送る機能も持っている。

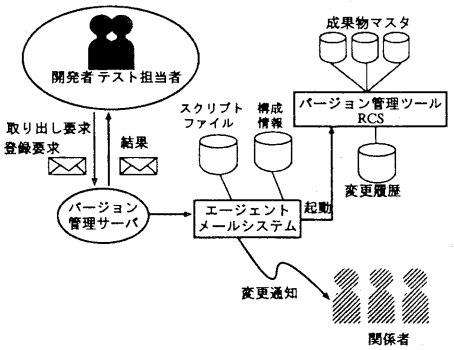

図 9: バージョン管理支援システム

6 適用評価

4節で述べたモデルを基に、バージョン管理支援システムを社内の複数部門に適用中である。

本節では、その一例の適用評価に関して述べる。

6.1 適用対象

対象部門の適用規模を表 1 に示す。

表 1: 適用規模

開発システム	制御ソフトウェア	
開発者数	31 名	
適用対象	サブシステム	56 個
	インクルードファイル	55 個
	ライブラリ	12 個
	データ	28 個
	計測テキスト	40 個
開発ステップ数	582K ステップ	
開発言語	C 言語	

6.2 定量評価

上記の適用対象部門で、バージョン管理支援システムを運用した、約 1 年半の期間中に得られた評価を述べる。

1. 不具合発生頻度の削減

   システム運用中の 5ヵ月間に発生した不具合件数と不具合の要因をバグ収集シートより調査した。その結果、バージョンの不一致 / 仕様確認洩れ / 変更後未登録 / センタ登録ミスを要因とする不具合が、バージョン管理支援システムの適用により根絶したことがわかった。

   適用前、適用後の不具合件数に関する詳細の数値は以下の通りである。

   支援システム適用前：
   不具合全体の **17 %**がバージョンの不一致等の不具合
   支援システム適用後：
   不具合全体の **0 %**がバージョンの不一致等の不具合

   これは、ソースファイルやデータファイルの取り出しや登録までの開発手順が明確になったことと、その手順が開発者全体に浸透したことにより、共同開発者間の作業状況の把握が容易になり、サブシステム間の整合がとりやすくなったことによる効果であると考えられる。

2. 開発作業の効率化と変更洩れの削減

   バージョン管理支援システム適用期間中の 5ヵ月間において自動的に発信された登録通知メールの履歴をシステムのログより収集した。それによると、

   300 回の登録で自動発信された通知メールは **2254 通**

   人間系の作業により、電話や FAX を使って登録通知が行われていた適用前より、作業効率を向上させたことがわかる。

6.3 定性評価

適用対象者全員に対して、システム導入後のバージョン管理運用に関するアンケート調査を行った。アンケートの結果、以下のような効果を確認することができた。

- 成果物の一元管理が可能になり、成果物のバージョン管理が確実に行えるようになった。
- 変更履歴の取得が可能になったため、グループリーダーやテスト担当者の作業がやりやすくなった。
- 成果物の取り出しや登録時の通知により、共同開発者の作業の進捗を把握できるようになった。

- 成果物の取り出しから登録までのルーチンが確立した。

7 考察

7.1 適用による効果

前節で示したように、バージョン管理支援システムの適用により、以下の効果が確認された。

1. 開発工数の削減
   - 不具合の中からバージョン不一致等による不具合を根絶し、後戻り作業を削減した。
   - 登録通知発信の自動化により、従来人間の手作業で行なっていた登録に要する作業工数を削減した。
2. 運用面での効果
   - 作業ルーチンが明確になり、それが開発者全体に定着したことで作業の煩雑化、後戻りを削減した。
   - 通知や管理の徹底により、バージョン履歴の取得が容易になり、テスト担当者やグループリーダがバージョン状態の把握や進捗の把握を行いやすくなった。
   - ファイルの状態の取得が容易になり、共同開発者間での作業状況が透過になり、開発しやすくなった。

7.2 今後の課題

今後の課題として以下を進めていく予定である。

1. 多機種展開開発の支援機能
   機種展開の多い製品のソフトウェア開発においては、機種間で共通するファイルが多く、共通ファイルとして管理する必要が出てくる。またこの場合、開発の途中まで共通で管理していたファイルを、開発の進度に従って共通管理から切り離し、別の枝で管理していくといった管理方法も考慮していかなければならない。本稿の中で、成果物の管理方式を述べたが、この妥当性を今後評価し、多機種展開開発の支援機能として確立したい。

2. ドキュメントやテストデータの管理機能
   現在、バージョン管理支援システムによる支援対象は、ソースコードとしている。今後、支援対象をドキュメントやテストデータ等まで拡張し、ドキュメントやテストデータ等で発生した変更と、ソースコードの変更との相互関係を管理する機能を拡張していきたい。

8 おわりに

ソフトウェア共同開発における後戻り作業を削減するために、ソースコードのバージョン管理に付帯する作業の流れをモデル化した。そして、開発の基本モデルを、作業の流れと役割担当者、成果物の管理方式に着目して、パターン化した。各パターンに沿って、スクリプトを記述し、エージェントメールシステムを応用したバージョン管理支援システムを開発した。

現在、バージョン管理支援システムは、社内の複数部門に適用している。

適用対象の1部門において評価を行った結果、バージョンの不一致による不具合の根絶による後戻り作業の削減といった、開発工数の削減や、開発ルーチンが徹底して開発しやすくなった等の、運用面での効果が確認された。

今後、他部門での評価も進めていく。また、多機種展開開発支援機能の確立と、ドキュメントやテストデータの管理機能等の拡張を急ぎたい。

参考文献

[1] 服部, 松尾, 貫井: ワークフローに基づくソフトウェア共同開発支援～遠隔地間でのバージョン管理支援, 第14回ソフトウェア生産における品質管理シンポジウム報文集 pp.191-198(1994)

[2] 服部, 松尾, 貫井: エージェントメールを適用したバージョン管理支援システム, 第47回全国大会講演論文集 (5)pp.203-204(1993)

[3] 松尾, 貫井, 中村: 電子メールにおけるエージェントシステム, 第43回全国大会講演論文集 (5)pp.296-297(1991)

[4] Watts S.Humphrey : ソフトウェアプロセス成熟度の改善, 日科技連 pp.125(1991)