海の研究（Oceanography in Japan），27(5)，175－188，2018，doi: 10.5928/kaiyou.27.5_175.

— 論　文 —

Git と Redmine を用いた気象研究所共用海洋モデル「MRI.COM」の開発管理*

坂本 圭[1]**・辻野 博之[1]・中野 英之[1]・浦川 昇吾[1]・山中 吾郎[1]

要　旨

著者らの海洋大循環モデル「気象研究所共用海洋モデル（MRI.COM）」は，開発が始まってから 20 年近くが経過し，気象研究所と気象庁の様々な部門で利用されるようになるとともに，ソースコードの大規模化・複雑化が進んだ。このような状況の下でも，バグの混入や意図しない影響を抑えながらモデルを効率的に開発するため，現代的なソフトウェア開発で用いられるツールと手法を取り入れ，開発管理体制を一新した。まず，ソースコードの開発履歴（バージョン）を管理する「Git（ギット）」を導入した。このツールにより，複数の開発者が複数の課題に同時に取り組む並行開発が可能になった。また，プロジェクト管理システム「Redmine（レッドマイン）」を導入し，開発状況を開発者全員で共有した。このシステムによってデータベースに逐一記録された開発過程が，他の開発者や次世代の開発者にとって財産となることが期待される。これらのツールを用い，さらに開発手順を明確にすることで，開発チーム内の情報共有と相互チェックを日常的に行う開発体制に移行することが可能となったことは，コード品質の向上に大きく寄与している。現在，気象庁では，MRI.COM だけでなく，気象研究所と気象庁で開発しているほぼ全てのモデルを Git（または SVN）と Redmine で一元的に管理するシステムを構築しており，モデルの開発管理及び共有化が大きく前進している。

キーワード：数値モデル，モデル開発，モデル共有，バージョン管理，プロジェクト管理

1.　はじめに

海洋大循環モデルは，この数 10 年の発展により，大気大循環モデルとともに，世界各国の地球科学研究における不可欠なツールとなっただけでなく，社会的にも重要なインパクトを持つようになった。日本では，気象庁が，1995 年にエルニーニョ監視，1999 年にエルニーニョ予測，2001 年に北西太平洋の海況監視予測に導入した海

＊　2018 年 1 月 25 日受領；2018 年 6 月 12 日受理

1　気象研究所海洋・地球化学研究部
〒 305－0052　茨城県つくば市長峰 1－1
＊＊　連絡著者：坂本 圭
e-mail：ksakamot@mri-jma.go.jp

洋大循環モデルが，産業，交通，防災をはじめとして，人々の生活の向上に寄与している。これら数値モデル予報の基盤を支える海洋大循環モデルの維持・開発は，気象研究所の著者らのグループの主要な業務である。また最近では，将来気候変動予測のための気候モデルの中心的なコンポーネントとしても重要性を増している。

一般に，モデルは，サブグリッドスケール現象のパラメタ化や支配方程式の離散化手法などに大きな不確実性を有している。これらの不確実性は，予測の誤差や不確実性に直結している。「気候変動に関する政府間パネル」等では多様なモデルの参画により評価された不確実性の幅は，気候変動に対する適応緩和策の立案において必須の情報である。各研究機関が，可能な限り，海洋大循環モデルを含めた気候モデルの不確実性を解消することは，人類社会に対する研究コミュニティの責務と言える。

しかしながら，モデルの維持・開発はより困難になってきているのが現状である。世界的にモデル開発者数は減少する（Jakob, 2011）一方で，モデルのソースコードは日毎に大規模化・複雑化している。このため，バグの混入や意図しない影響の抑制，つまりソフトウェア品質の維持は非常に難しくなっている。例えば，代表的な気候モデルでは，ソースコード 1000 行あたりのバグは 0.5–1.2 件と報告されている（Pipitone and Easterbrook, 2012）。これは 10–50 万行のモデル全体で 50–600 件ほどのバグがあることを意味する。多くのバグに対処しつつモデル開発を続けるには，精通した開発者が長い時間をかけなくてはならない。このため，これまで独自の海洋大循環モデルを維持してきた日本の大学の研究室でも，開発を中断しているケースがあると聞く。

気象研究所の海洋大循環モデル「気象研究所共用海洋モデル（MRI.COM）」でも開発状況は同じである。本モデルは，沿岸現象から全球循環まで対応する汎用性を持っている（Tsujino *et al.*, 2011; 坂本ほか, 2013; 辻野ほか, 2015）。このため，ソースコードは全体で 10 万行を超え，100 近いコンパイルオプションと数 100 に及ぶ実行時オプションがある。加えて，仕様を変えずにバグ修正だけが必要な現業目的と，次々に機能を追加する必要がある研究目的の両方に対応するため，安定版と開発版の 2 系統を維持しなければいけない。開発が始まってからおよそ 20 年が経過し，過去の絡み合ったソースコードの整理・再構成も必要である。これらソースコード自体の複雑さに加えて，その利用が多様化しているという問題もある。長い開発期間の結果として，MRI.COM は，日本沿岸海況監視，季節予報，将来気候予測など，気象庁と気象研究所の多様な現業と研究の基盤モデルに用いられ，それぞれの幅広い要望に対応して，同時並行的に開発が進められている。

このような複雑な状況の下で，ソフトウェア品質を維持しながら様々な機能を次々にソースコードに加えていくことが，少人数の開発チームに求められている。そこで著者らは，モデル開発を円滑に進める基盤として，チームとして協力して開発に取り組める体制を数年かけて構築した。これまでも著者らのグループでは，構成員が互いに協力して開発してきたが，開発作業は各開発担当者のそれぞれのやり方に委ねられていたため，情報共有や作成されたソースコードの品質にムラがあった。この状況を改め，開発プロセス自体をシステマティックに管理するため，ソフトウェア開発の現代的なツールと手法を取り入れた開発管理を導入した。

現代のソフトウェア開発においては，開発管理を支援する様々なツールと手法が標準的に使われている。最も重要なツールの 1 つが，ソースコードの開発履歴を管理するバージョン管理システムである。1990 年代には，ツール「CVS」が主に使われており，MRI.COM の開発でも使われていた。2000 年代以降には，より高機能な Subversion（以下，SVN とも呼ぶ），Git，Mercurial などのツールが広く普及している。もう 1 つが作業過程の記録と進捗状況の把握を一元的に行うためのプロジェクト管理システムである。代表的なものとしては，ウェブベースの Trac や Redmine などがある。これらのツールは主にオープンソース・ソフトウェアや商用ソフトウェアの開発で用いられてきたが，大規模化が進む学術的なソフトウェアの開発にも有用であり，多くの研究開発機関で導入が進んでいる。例えば海洋大循環モデルでは，ヨーロッパを中心に共同開発されている「NEMO」は Subversion と Trac で（NEMO development board, 2018），NOAA の地球流体力学研究所（GFDL）で開発されている「MOM」の最新版は Git ベースの共有ウェブサービス「github」で（NOAA-GMDL, 2018），バージョン管理とプロジェクト管理がおこなわれている。気

象庁においても，2013 年に全庁的にモデルの開発管理に Git, Subversion, Redmine を導入している（気象庁予報部数値予報課，2017）。

さらに，これらのツールを基盤に用いることで，開発者各人がそれぞれのやり方で開発する状況から脱して，システマティックな開発手順を導入することができる（例えば，Rasmusson, 2011）。つまり，開発の各ステップを，課題の設定，他メンバーへの説明，コーディング，レビュー，テスト，リリースなどに区分し，それぞれの作業を明確にすることができる。このように開発手順を標準化することは，モデルの開発速度の高速化とソフトウェア品質の向上の両立を実現するための不可欠な手段である。

本論文では，著者らが MRI.COM 開発管理のために導入した主なツールと，それらツールを用いて標準化した開発手順を紹介する。第 2 節では，バージョン管理システムとプロジェクト管理システムの紹介と，どのように活用されているかを説明する。第 3 節では標準化された開発手順の内容を，第 4 節では気象庁と気象研究所におけるモデル開発に関する情報共有の仕組みを述べる。最後に第 5 節で本論文をまとめ，今後の課題を示す。

2. 開発管理のためのツール

まず，MRI.COM の開発管理のために導入した Git（ギット）と Redmine（レッドマイン）の概要を説明する。詳細については公式サイト（Git Project, 2018; ファーエンドテクノロジー, 2018），入門書（濱野, 2009; 小川・阪井, 2010），気象庁技術報告（気象庁予報部数値予報課, 2017）を参照してほしい。

2.1. バージョン管理システム : Git

Git とは，ソースコードの開発履歴（バージョン）を記録，管理する「バージョン管理システム」と呼ばれるツールの 1 つである。特に，Linux など多数の人々が関わるオープンソース・プロジェクトの管理のために開発されたため，共同開発をサポートする強力な機能を持つ。ソフトウェア開発においては必須のインフラであり，このようなバージョン管理システムなしでは大規模なソフトウェアの集団的な開発は実質的に不可能である。

あるソフトウェアを複数人で開発する際には，一般に，管理者が持つオリジナルのソースコード（マスターブランチやトランクとも呼ばれるが本論文では幹と呼ぶ）を，各自の開発環境（ローカルと呼ぶ）にコピーしてから作業を始める。集団開発でまず必要なのは，幹のバージョンを一元的に管理し，散在させないことである。しかしながら，手動でバージョンを逐一記録することは煩雑な作業であり，データの紛失や混乱が起こりやすい。一方，Git ではソースコードを変更する場合は，常に，ソースコード全体のスナップショットを 1 つの単位として「リポジトリ」と呼ばれるデータベースに，逐次，記録する。例えば Fig. 1 に示すように，8 月 8 日，12 日，15 日とソースコードが一部変更される度に，その全体がリポジトリに記録される。手動の管理で起こりやすい，ある時期のバージョンの紛失などは，原理的に起こり得ない。また，幹の開発履歴が明示されることで，各開発者がローカルにコピーしたソースコードが最新のバージョンかどうかもすぐに判断できる。ローカルのソースコードを最新のバージョンに更新することや，記録されたある特定のバージョンを展開することもコマンド一つで可能である。このように Git を導入することで，幹のバージョンが自動的に一元的に記録され，バージョン更新に伴う混乱を防ぐことができる（ここで説明した機能はバージョン管理システムの基本的な機能であり，Git 以外でも SVN 等で実現できる）。

プログラムの共同開発では，幹が随時更新されていく状況でローカルの開発を行っていく。その際に重要なのは，ある開発者による幹の変更が他のメンバーでも理解できることである。その変更が理解されていないと，プログラムのその部分はブラックボックス化し，他の開発者による変更が難しくなる。Git では，1 回のソースコード変更を「コミット」と呼ぶ。変更は複数のファイルに及んでもよいが，1 つの意味内容であることが推奨される。例えば，Fig. 1 の（a）では，8 月 12 日のコミットはある 1 ヶ所のバグ修正であり，15 日はある 1 つの機能追加である。このように，ソースコード変更をコミット単位で記録することで，変更内容と目的が関連付けられ，開発者本人以外でも理解できるようになる。

Git では，コミットを柔軟に編集する様々な機能が提供されている。これを用いることで，ローカルで行った

コードの変更を整理した形で幹に還元できる。その1つの機能が「スカッシュ・マージ」である。プログラム開発において1回の作業で納得できるソースコードを作成できるのは稀であり，通常は様々な試行錯誤を行う。例えば，Fig. 1の（b）に示すように，仮組み，修正，コード整理などの，いくつかの段階を経て，開発は進む。しかし，途中経過のコミットは，開発時には必要だったものの，幹に反映させるには不要であり，むしろ他の開発者を混乱させてしまう。このような場合には，複数のコミットを1つにまとめるスカッシュ・マージの機能を使うことで，変更が分かりやすい完成稿のコミットを作ることができる。もう1つの機能が「リベース」である。ある時点の幹からローカルの開発を始めたが，作業中に幹が更新されることはしばしばある。このような場合，Fig. 1の（c）に示すように，リベースによってコミットの起点を最新バージョンに変更することで，あたかも最新バージョンを元にコード開発を行ったようにコミットを変更することができる。この機能によって，幹から分岐することなく，開発したコードを最新の幹に反映できる。これらの機能は，ソースコードの分岐を防ぎ，複数の開発者が複数の課題に同時に取り組みつつ成果を単一の幹に還元する「並行開発」を可能にする。

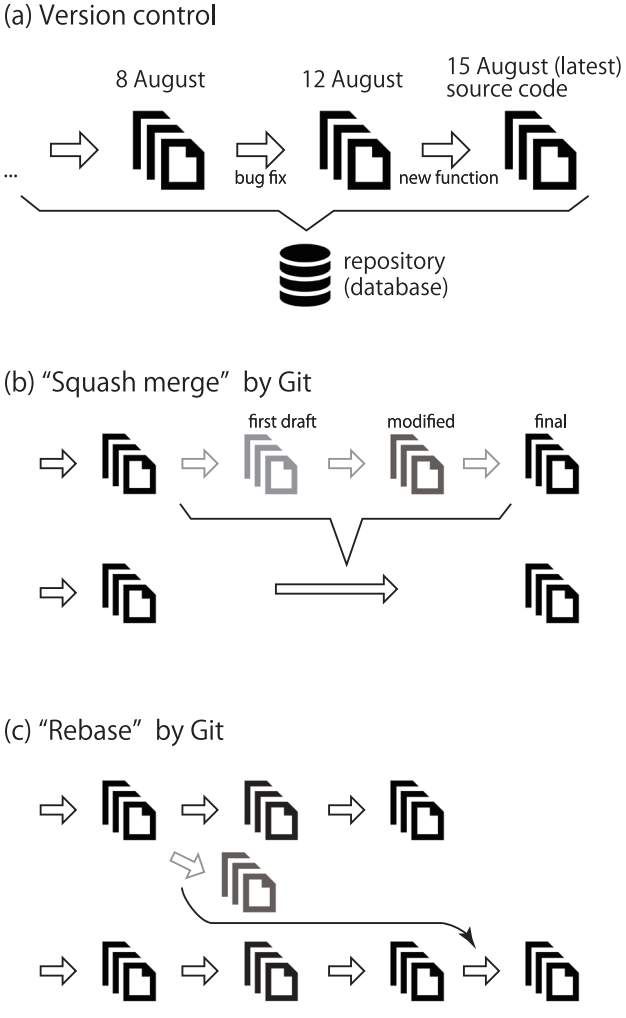

Fig. 1. Schematic diagram of basic functions of a version control system, Git. (a) Version control. Every time a part is changed, the entire source code is recorded in the repository (database). (b) "Squash merge" function. (c) "Rebase" function.

Gitの大きな特徴は，Fig. 2に示すように，リポジトリが単一ではなく，各開発者や共有サーバー上の複数のリポジトリが連携する「分散リポジトリ管理」を採用していることである。これにより，各開発者は各自のリポジトリ（ローカル・リポジトリ）で作業しつつ，共有サーバー上のリポジトリ（共有リポジトリ）は幹のコミットのみを取り込むという形で，個人の開発と共同開発をシームレスに両立させることができる。この特徴について具体的に説明するため，著者らの日常的な作業の例をFig. 2に示す。本例では，共有リポジトリ上にあるバージョン番号1.0のソースコードを元に，バージョン番号1.1のソースコードを作成する状況を考える。その際，開発者Aが主に開発し，開発者Bが相談役となる。一方，開発者Cも平行して開発しているが，今回のバージョン番号1.1のソースコードの作成には関わらないとする。そこで行う作業手順は，以下のようになる。

1）各開発者が開発の元となるバージョン1.0のソースコードを共有リポジトリからローカル・リポジトリへコピーする。
2）開発者Aがローカルでコミットし，バージョン1.1の案として「1.0.a」を作成する。
3）開発者Bは1.0.aを開発者Aのリポジトリから自分のリポジトリへコピーする。
4）開発者Bは1.0.aの変更を調べ，不十分だった点を修正するコミットを行い，1.0.a'を作成する。
5）開発者Aは1.0.a'を開発者Bのリポジトリから自分のリポジトリへコピーする。

6）開発者 A は 1.0.a' の変更を調べ，1.0.a より確かに良いことを確認した後，1.0.a' を 1.1 とすることを決める。そして，Fig. 1 の（b）に示したスカッシュ・マージにより，最終的なコミットとする。
7）開発者 A は 1.1 を自分のリポジトリから共有リポジトリへコピーする。幹の歴史が 1 つ進んだことになる。
8）開発者 C は，共有リポジトリ上の更新された幹を自分のリポジトリへコピーする。
9）開発者 C は，リベースによって自分の開発中のソースコード（1.0.b）を最新の幹を起点とするソースコード（1.1.b）に変更し，開発を続ける。

この手順は一見すると煩雑に見えるかもしれないが，リポジトリ間の連携はコマンド 1 つで行えるため，著者らのモデル開発で日常的に行われている。Fig. 2 が示すように，分散リポジトリ管理の仕組みにより，共有リポジトリには幹の歴史だけが残り，各開発者の試行錯誤はそれぞれのローカル・リポジトリのみで行われることに注目してほしい。これにより，個人の開発とグループの共同開発を区別しつつ円滑に連携させて開発を進めていくことができる。

バージョン管理システムの導入により，安定版と開発版といった複数の系統の維持も可能になる。MRI.COM では，内部で開発している開発版とは別に，現業や外部ユーザー向けに安定版の提供を行っている。開発版にはバグ修正，機能追加，変更，リファクタリング（機能は変えずにソースコードを整理すること）といった様々なコミットがなされるのに対し，安定版にはバグ修正のみがなされる。この 2 系統維持を手動で行おうとすると，通常の開発とは別に，安定版のバグ修正という作業が別

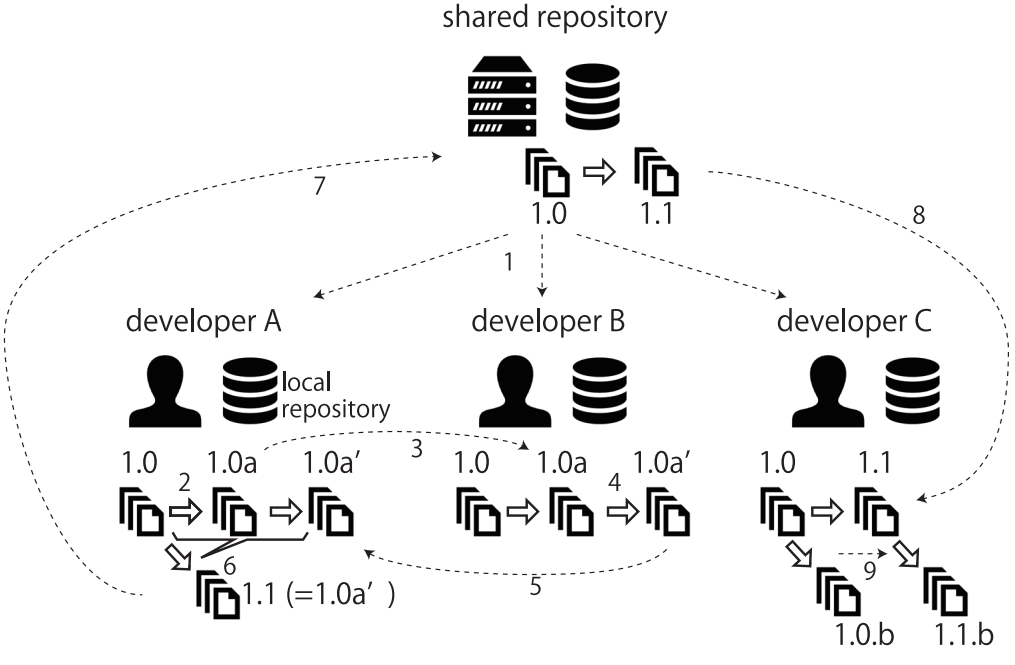

Fig. 2. An example of collaborative model development using the Git's distributed repository system. There is a trunk in the shared repository, and developers A, B and C develop the model in parallel in their respective repositories. The numbers attached to the code icons are version numbers. The broken or wide arrows with numerals indicate operating procedures as follows: (1) three developers, A, B, and C, get the latest trunk code, (2) developer A makes a draft for the next version, (3) developer B reviews the draft, (4) developer B modifies the draft, (5) developer A confirms the modification, (6) developer A makes the next version, (7) developer A publishes the next version, (8) developer C updates the trunk, and (9) developer C continues development based on the next version.

途必要になる。その際，開発版に対して行ったバグ修正の作業を，安定版で正確に再現する必要があり，負担が大きい。そのため，以前の MRI.COM では安定版は作成されなかった。一方，Git 導入後には，他の系統に行ったコミットを別の系統に取り込む「マージ」機能を用いることで，僅かな労力で安定版を維持することができる。Fig. 3 に 2 つの系統の開発履歴を示す。上方の安定版（stable Ver.1.0）では，Ver.1.0 を起点として，バグ修正のみがコミットされる。一方，下方の開発版（developing Ver.1.1）では，安定版にコミットされたバグ修正に加えて，機能追加やリファクタリングがコミットされる。このように開発することで，Ver.1.0 のバグ修正作業を 1 回行うだけで，安定版と開発版の両方に反映されるようになり，2 系統維持の負担が抑えられる。

2.2. プロジェクト管理システム：Redmine

これまで述べてきたように，海洋モデルのような大規模なソフトウェアでは，多くの開発者が様々なタスクを分担して開発を進める。このような並行開発を円滑に進めるためには，各タスクの作業過程を記録し，進捗状況を開発メンバー全員で共有することも必要である。プロジェクト管理システムとは，ソフトウェア開発における様々なタスクを一元的に管理するためのツールであり，MRI.COM の開発においても欠かせない基盤となっている。本節では，プロジェクト管理システムに用いている Redmine について紹介する。

Redmine では，タスクは「チケット」として登録され，ブラウザを通して管理される。MRI.COM 開発の 1 つのチケットを表示させた例を Fig. 4 に示す。この図には，以下の項目が示されている。

1）トラッカーと呼ばれるチケットの分類。MRI.COM では Redmine のデフォルト設定にならい，「バグ（バグ修正）」，「機能（機能追加）」，「サポート（その他）」の 3 種類を用いている。
2）チケット番号。登録時に発行される。
3）タイトル。タスクを簡潔に説明する。
4）進捗状況を示すステータス。MRI.COM では「新規」，「方針確認」，「進行中」，「解決」，「レビュー済」，「フィードバック」，「終了」の 7 種類から選ぶ。詳細は第 3 節で説明する。
5）開発を行う担当者。
6）開発が行われるリリース・バージョン番号。後述するように，リリース・バージョンは月毎に更新されるので実質的に期日を意味する。
7）変更について検証するレビュアー。レビューについては第 3 節で説明する。
8）タスクの背景，目的，インターフェイスや実装方針などの説明メモ。
9）関連付けられた Git のコミット。クリックすると，ブラウザでコード変更箇所（いわゆる diff）を閲覧でき

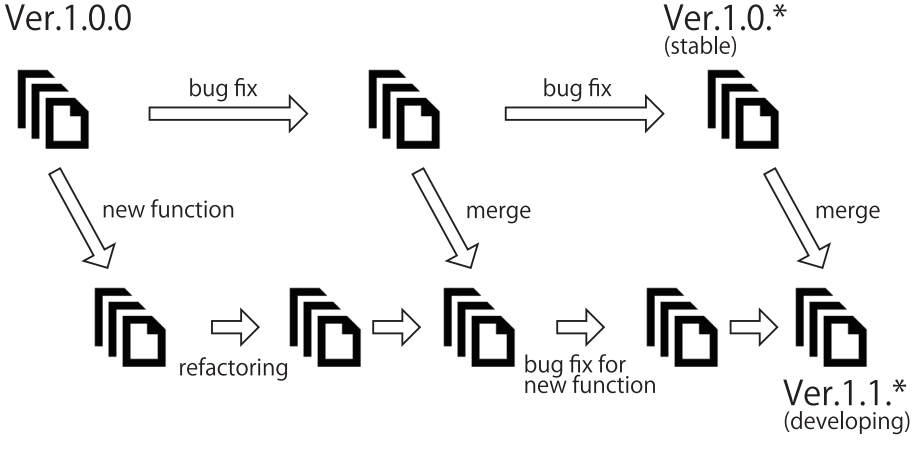

Fig. 3. Schematic diagram of commitments made to a stable version (Ver.1.0.*) and a development version (Ver.1.1.*) of the model. Git can record multiple lines with one repository.

る。

10) 開発に関するコメント。開発中の実装方針の変更やレビュアーによる意見など，作業時の議論をなるべく残す。

このように，チケットはタスク内容と進捗に関して必要な情報を全て記録し，ブラウザによってひと目で把握することができる。

チケットの入力項目は必要最小限としているものの，やはり，ある程度の追加入力作業が必要である。それでも，開発項目の全てがデータベース化されることのメリットは，追加入力作業に十分に見合うものである。具体的なメリットを以下に挙げる。

1) 他人に見られることを前提にしてタスク内容を明文化することで，開発方針が明確になる。

2) 担当者や期日が明示されることで，各タスクの優先度の誤認などを防ぐ。

Fig. 4. A Redmine ticket in Japanese made for MRI.COM. The items indicated by circled numbers are as follows: (1) classification (tracker), (2) serial number, (3) title, (4) status, (5) person in charge, (6) release version, (7) reviewer, (8) note, (9) git commit and (10) comments.

3) 実装方針など開発に関する議論が残されることで，他の開発者と将来の開発者はコード変更についての理解が容易になる。開発過程のデータベース化は，次世代の開発者にとって大きな財産となる。
4) 開発内容とソースコードが関連付けられ，どのような目的で，どのようにコードが変更されたかを誰もが見えるようになる。これにより，モデル利用者に対して，なぜそのような変更を行ったのかを説明する責任を果たすことになる。これは，現業使用や外部提供もされる MRI.COM にとって重要である。
5) Redmine でチケットを一覧表示させることで，モデル開発の全体の進捗を共有できる。

上記 5) で挙げた開発進捗の共有は特に重要であるので，実際の MRI.COM の開発について，チケットを一覧表示した Fig. 5 を例にして，より詳しく説明する。MRI.COM では 1 ヶ月単位のリリース・サイクルを採用しており，例えば 2014 年 5 月末に Ver.3.3.15, 2014 年 6 月末に Ver.3.3.16 をリリースした。Fig. 5 に示す Redmine でのチケット表示もリリース・バージョンに紐付けられ，Ver.3.3.15 では 16 件の開発事項があったことが分かる。MRI.COM ではこれらチケット内容をまとめることで，例えばバグ修正 4 件，機能追加 3 件，それ以外が 9 件などの「リリースノート」を，新しいバージョンのソースコードと同時に公開している（気象研究所・海洋研究開発機構, 2018)。また，Fig. 5 の時点において，Ver.3.3.16 の開発では 11 件のタスクが完了する一方で，予定されている 9 件のタスクが未完了であることも分かる。このように，プロジェクト管理システム導入によって，開発者は自分の作業だけでなく全体の開発の進捗状況も簡単に把握できる。これはチーム開発にとって必須の機能である。

3. MRI.COM 開発手順の標準化

Git と Redmine の導入後，著者らは，協力して系統的に開発する体制を整えるために開発手順の標準化を行った。具体的には，各開発者は以下の 7 つのステップに従って開発を進めることとした（Fig. 6)。

1) 開発者はまず Redmine チケットを作成し，Fig. 4 のように簡潔に開発内容を記す（いわゆるチケット駆動開発，小川・阪井, 2010)。
2) 開発者は他の開発メンバー 1 名を「レビュアー」に指名し，ソースコードを編集する前に，変更内容と実装方針について相談する（ただし，簡単なバグ修正であれば必須ではない)。
3) 相談結果を踏まえ，開発者がローカルでコード変更の作業を始める。
4) ローカルでコード変更を完成させ（Fig. 2 の手順 2 に相当)，レビュアーにレビューを依頼する。
5) レビュアーがコード変更のレビューとテストを行う。レビュー結果は Redmine のチケットのコメントや Git の修正コミットとして開発者にフィードバックされる（Fig. 2 の手順 3, 4, 5)。
6) コード変更を開発メンバー全員に公開し，チェックとテストを受ける（Fig. 2 の手順 7, 8)。
7) 検証が終わったコードを幹とする（Fig. 2 で Ver.1.1 を確定することに相当)。これで 1 つのチケットの開発は終了し，確定した歴史となる。

チケットが現在どのステップにあるかは，Redmine の「ステータス」で示され，他の開発者も現在の進捗を見ることができる（Fig. 4)。実際の開発では，いつもステップどおりに進むわけではない。しかし，ステップ 5 と 6 のコードレビューとテストは省略せず，幹に入る変更については，他の開発者によるチェックが必ず行われるようにしている。

開発チーム内の情報共有と相互チェックを日常的に行う体制を構築したことが，モデルのソフトウェア品質の改善に貢献している。まず，コードレビューにより複数の人の意見が反映されることで，ブラックボックス化したソースコード数が減るのに加えて，変数名やインターフェイスがより分かりやすくなるなど，可読性が向上している。開発メンバー間で既存のソースコードに対する理解が進むことで，コーディングの整合性も改善している。ステップ 6 があることで，テストの自動化も進み，幹に入る前にバグが修正されることも多い。さらには，これまで避けられてきたようなソースコードの構造を大きく変える変更にも，着実に取り組めるようになった。この原因としては，ツールによるサポートに加えて，開発者間の議論が進んだことで，そのモチベーションが生

3.3.15

2014/05/31

z*座標の導入に伴う作業、FX10対応

100%

16 チケット （16件完了 — 0件未完了）

関連するチケット

~~バグ #852~~: MLONLY オプションでコンパイル/実行ができない
~~バグ #880~~: vvdimp.F90 体積変数のtypo
~~バグ #883~~: offline nesting の親モデルがsr16000でコンパイルできない
~~バグ #884~~: margin__lpmax_[2,3]d における処理を同じにする
~~機能 #842~~: ssh異常値による終了の際に、標準出力が全てファイルに書き出されるようにする
~~機能 #853~~: l_global_local_cnsv_ssh = .true. の時、拡散フラックスのリスタートを読み込まなくても計算を開始できるようにする
~~機能 #866~~: bottom friction の係数をnamelist 指定できるようにする。
~~サポート #830~~: 標準出力 write(*,)と print *, をwrite(6,)に統一する
~~サポート #834~~: isopycnal 関係のzstar対応
~~サポート #835~~: cbnbio 関係のzstar 対応
~~サポート #836~~: オフライン関係のzstar 対応
~~サポート #863~~: ptrc のzstar 対応
~~サポート #867~~: 東大FX10用の設定ファイル作成
~~サポート #868~~: 体積・層厚関連で重複している変数を一本化する
~~サポート #869~~: 体積・層厚関連の旧変数を削除する
~~サポート #876~~: nonhydro および upc, utzq, quick の移流スキームのzstar 対応

3.3.16

期日まで 0日 (2014/06/30)

z*座標の導入を終了

91%

20 チケット （11件完了 — 9件未完了）

関連するチケット

~~バグ #893~~: ppm_adv.F90 の zstar への移行が完全ではない等
~~バグ #896~~: ZSTAR オプションにおける深度アノマリの計算ミス
~~バグ #901~~: cbnNPZD のzstar 対応が完全でない件
バグ #909: モデルの海水＋海氷体積をモニターする部分(total_vol_sigma (oghist2.F90)) のZSTAR向け修正が欠落
バグ #910: topo_vary.F90 に配列定義外参照がある
バグ #911: ppm_adv.F90 に配列定義外参照がある
バグ #915: SOMにおいてモーメント合成時に用いられている係数の修正
バグ #916: 浮力による乱流エネルギー生成・消滅の評価項の表現
~~機能 #847~~: オフライン・ネスティングとオンライン・ネスティングを同時に使用できるようにする
~~機能 #870~~: 海面淡水フラックスに対する z* 座標系における扱いの適用
機能 #871: z* 座標系の乱流混合モデルへの適用(仕様変更)
~~機能 #872~~: z* 座標系のBBLスキームへの適用
~~機能 #873~~: Leap-frog スキームを解く際、海面淡水フラックスに伴う、熱・海水の出入りを前半・後半部分のものでシリアルに加算する(仕様変更)
~~機能 #891~~: 水平フラックスを計算に乗じる断面積、鉛直粘性/拡散計算に使用する層厚の見直し(仕様変更)
~~機能 #897~~: solarangle.F90 における入射短波の減衰率計算に zstar　座標用のものを加える
機能 #905: オフライン・ネスティング子モデルでパッシヴ・トレーサーを独自に流せるようにする
~~サポート #902~~: NEMUROの鉄のパラメーター調節
サポート #903: オフライン・ネスティング子モデルにおける順圧成分出力の見栄えを良くする
サポート #904: namelist__print_blockに渡す変数iosに初期値を設定する
~~サポート #913~~: SOMの高速化

Fig. 5. Redmine tickets of MRI.COM shown as a list in Japanese. The content and progress of the tasks can be grasped in the list released every one month. Each list item indicates a ticket tracker, a serial number, and a title.

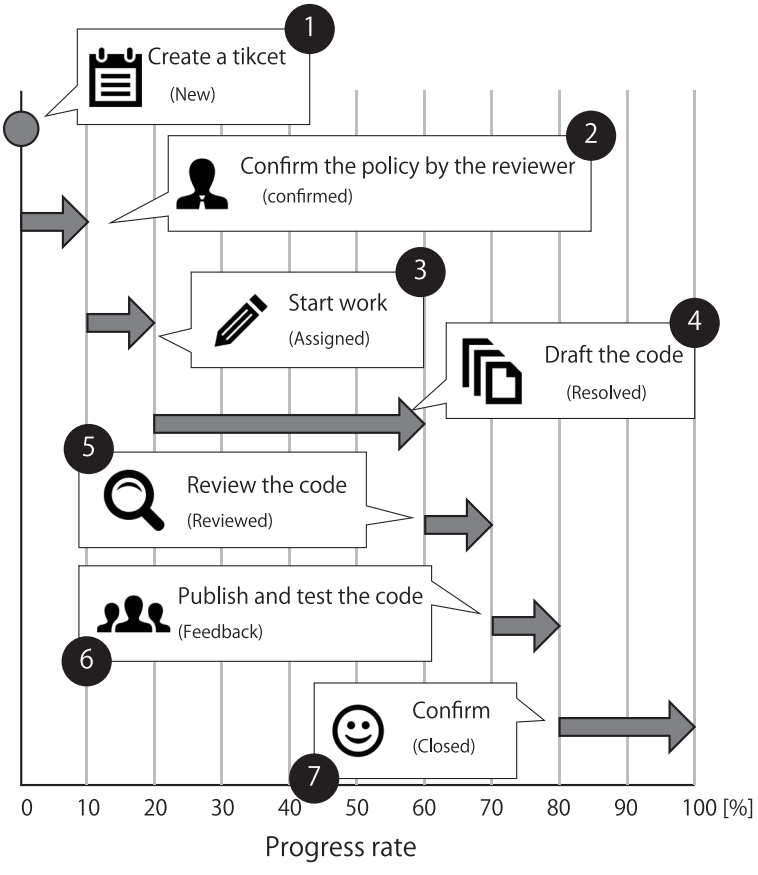

Fig. 6. Schematic diagram showing the operating procedure to develop MRI.COM by a Gantt chart. Start with Step 1 "Create a ticket", and end with Step 7 "Confirm". The parentheses in the steps are "status" shown in a Redmine ticket, and the horizontal axis is "progress rate" which is a rough guide and there is no strict meaning.

まれたことが考えられる。実際，MRI.COM の最近の開発では，鉛直座標系といった基幹となる力学フレームワークの変更や，モジュールの階層構造化といった大規模なリファクタリング（プログラム再構成）が取り組まれている。本節で説明した Git と Redmine を活用した開発手順はソフトウェア品質を維持，向上する基盤となっている。

4. 気象研究所と気象庁内のモデル共有

気象研究所と気象庁（以下「機関内」と呼ぶ）では，気象，気候，海洋，地球化学，地震，火山など様々な分野において 100 以上のモデル，システムが開発されている。その目的も，日々の現業から特定の研究まで幅広い。著者らが開発している海洋大循環モデルと同様に，これら多数のモデルも大規模化・複雑化が大きく進んでいる。現業では安定したモデル運用のために，研究・開発では実験の再現や効率的な開発のために，系統だったモデルの管理が必要不可欠となっている。また，1 つのモデルにより多くの開発者が関わるようになり，気象研究所と気象庁で共同開発するなど，部署を超えた開発も増えている。これらの課題を解決するために，気象庁はタスクチーム「開発管理調整グループ」を立ち上げて検討を進め，2013 年にモデル開発を一元的に管理する「開発管理サーバー」を導入した。実際，第 3 節で説明した MRI.COM の開発手順もこのサーバーを利用している。本節では，機関全体におけるモデル開発管理の取り組みを紹介する。

Fig. 7 に開発管理サーバーによるモデル管理の概念を示す。現在，機関内で開発しているモデルは，特別の事情がない限り，本サーバー上の Git もしくは Subversion（SVN）によってバージョン管理され，Redmine で開発管理されている。サーバーは常時稼働しており，機関内から自由にアクセスできる。本サーバーによりモデルの管理は大きく改善し，ソースコードの記録と共有，開発状況の把握のための不可欠なインフラとなっている。サーバーの管理は開発管理調整グループが担当する。これにより，リポジトリのバックアップやサーバーのメンテナンスといった必要だが複雑な作業の負担から，各モデル開発者を解放している。

開発管理サーバーによるプロジェクト管理とバージョン管理は，モデルのソースコードだけでなく，より広範に使われている。その 1 つが，モデルを動作させるためのミドルウェアや，前処理と後処理に使われる関連ツールである。例えば，気象庁独自のジョブスケジューラやデータフォーマットの変換ツールが開発管理サーバーで管理されており，開発グループ以外のユーザーも最新版のツールに簡単にアクセスできる。著者らも，MRI.COM 関連ツールをパッケージングした「MRI.COM 実験環境（MRI.COM eXperiment Environment, MXE と呼ぶ）」を開発・提供している。本パッケージには，モデル実行に必要な地形ファイルやフォーシングファイル等を作成する前処理ツール，モデル出力を必要に応じて編集する後処理ツール，積分・微分の計算や密度の計算などの基本的な解析ツールが含まれる。本パッケージは，MRI.COM の外部ユーザーにも提供されている（気象研究所・海洋研究開発機構，2018）。

モデルを用いた実験結果の共有にも Redmine がよく用いられている。実験課題をチケットとして登録し，解析結果を記録している。実験結果を踏まえてモデルの仕様を変更する場合にも，仕様変更のチケットと実験結果解析のチケットを関連付けることで，その理由を示すことができる。バージョン管理は文書作成にも有効である。例えば，MRI.COM 解説書（Tsujino *et al.*, 2017）の共同執筆は，モデル開発と同様に Redmine と Git を用いて進められた。ミーティングの議事録なども Redmine 上で保管されるケースが増えている。他にも，モデル開発以外の業務のタスク管理に使われる場合もある。2018 年現在，気象庁では，スーパーコンピューターのリプレースに向けて，多数のモデルの移行作業が進んでいる。移行作業

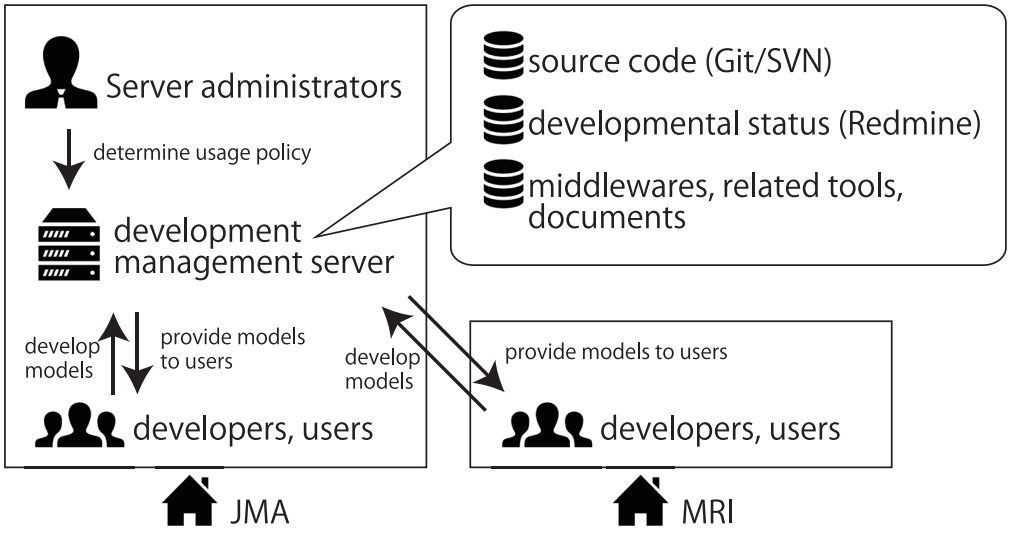

Fig. 7. Schematic diagram of model development management at JMA (Japan Meteorological Agency) and Meteorological Research Institute (MRI). Almost all the models of both institutions are controlled by Git, SVN and Redmine, which are running 24 hours a day on the development management server. We can access the model source codes and information in the two institutions at any time we want.

は数1000のRedmineチケットとして登録され，進捗が一元的に管理されている。このように，様々なケースで開発管理サーバーによるバージョン管理とプロジェクト管理が利用されている。これまでの情報共有の仕組みは，部署ごとにまちまちだったが，RedmineとGit/SVNが共通のインフラになりつつある。

5. まとめと課題

汎用海洋大循環モデル「気象研究所共用海洋モデル（MRI.COM）」のソースコードの大規模化・複雑化に対応するため，現代的なソフトウェア開発環境で用いられているツールと手法を取り入れ，開発管理体制を一新した。まず，ソースコードの開発履歴を管理するバージョン管理システム「Git（ギット）」と，開発のタスクを管理するプロジェクト管理システム「Redmine（レッドマイン）」を導入した。そのうえで，開発のステップを，課題の設定，他メンバーへの説明，コーディング，レビュー，テスト，リリースと区分し，そこで行うことを具体的に定める「開発手順の標準化」を行った。これらの取り組みにより，バグの混入や意図しない実験結果の改変を抑制しつつ，モデルの集団開発を効率的に進める体制が確立された。これによるメリットは多々あるが，とりわけ，複数の開発者による並行開発の円滑化，開発過程のデータベース化，ある変更を本体に取り込む前に他の開発者がチェックする「コードレビュー」の必須化，安定版と開発版の2系統維持，テスト自動化は，モデル開発において有益である。また，開発手順の標準化は，他部署や他機関の人が行ったバグ修正や開発も通常の手順で幹のソースコードに取り込む体制が確立されたことを意味する。実際にMRI.COMの開発に参加したい読者は，GitとRedmineの操作を説明した開発者向けドキュメント（気象研究所・海洋研究開発機構，2018）を参照し，著者らと相談しつつ，第3節のステップ1から始めてほしい。

GitとRedmineは気象庁内に置かれた「開発管理サーバー」で常時稼働しており，気象研究所内と気象庁内から自由にアクセスできる。気象庁では開発管理を検討するタスクチームの主導により，MRI.COMを含めて機関内で開発しているほぼ全てのモデルをGit（またはSVN）とRedmineで一元的に管理する体制が構築された。これによって開発者・ユーザーは，部課室の垣根を超えて自由に最新のソースコードを使うことができる。モデルに加えて，ミドルウェア，周辺ツール，ドキュメントも同じ枠組みで共有化が進んでおり，MRI.COMについても前処理・後処理・解析ツールをまとめたパッケージ「MRI.COM実験環境（MXE）」が開発・提供されている。

このように機関内のモデル共有は大きく進んできた。今後は，外部の機関や研究者との共同開発に向けて，開発管理手法をさらに拡張する必要がある。その背景には，モデルの大規模化が進むにつれ，それぞれの専門分野を持つ多数の開発者が開発に携わらなければならない状況がある。海洋モデリングでは，力学フレームワーク，乱流スキーム，海氷モデル，波浪モデル，ダウンスケーリング手法，高速化手法など，より幅広い領域で様々なスキームが提案され続けている。加えて，現在のモデリング研究では，大気モデルや生態系モデルなどの他分野のモデルとの結合もますます盛んにおこなわれており，海洋物理分野以外の研究者との共同開発も必要になると考えられる。このような状況の下，海外の主要なモデルはソースコードがインターネット上に公開され，広くコミュニティでなされる様々な改善をモデル本体に取り込む体制が確立されている。

MRI.COMについても，外部の機関やユーザーを含めた共同開発に移行することが検討されている。しかし，いくつかの解決すべき課題がある。1つ目は，開発管理を担っている開発管理サーバーに，セキュリティの制約により外部からアクセスできないことである。2013年から実施しているJAMSTEC，東京大学，九州大学とのモデル開発に関する共同研究では，JAMSTECのサーバーでGitとRedmineが稼働している。将来的には，より広い共同開発のために，海外主要モデルで行っているようにgithub等のインターネットでアクセスできるシステムを利用するのが望ましいように思われる。2つ目は，ソースコード公開の制約と著作権である。外部ユーザー向けの貸与制度を2012年に開始したのに加え，上記の共同研究の参加研究機関内ではソースコードを共有している。しかし，原則として，MRI.COMのソースコードは非公開である。日本の研究コミュニティの支持を得るには，抜本的な変更が必要になると考えられる。3つ目は，ユーザー向けサポートである。第4節で述べたMXEの作成

や，力学フレームワークと主要なオプションを解説したマニュアル（Tsujino *et al*., 2017）の公開などがおこなわれているが，MRI.COM 習熟のハードルは高く，よりきめ細かいサポートが必要である。これらの問題を一歩ずつ解決し，MRI.COM を広く日本の海洋研究コミュニティで支えられる体制に移行することによって，先人たちの努力を引き継ぎ，世界的な海洋モデル開発を先導したいと考えている。

謝 辞

本論文執筆にあたり，気象研究所豊田隆寛主任研究官，気象研究所海洋研究部第二研究室の皆様，および気象庁技術開発推進本部モデル技術開発部会開発管理調整グループの皆様には様々なご教示をいただいた。また，国立極地研究所の照井健志博士には論文執筆のきっかけを与えていただいた。心から感謝の意を表します。

References

ファーエンドテクノロジー (2018): redmine.jp. http://redmine.jp/（最終閲覧日2018年8月2日）

Git Project (2018): git. https://git-scm.com/（最終閲覧日2018年8月2日）

濱野純 (2009): 入門 Git，秀和システム．東京都，340 pp.

Jakob, C. (2011): From regional weather to global climate: Progress and challenges in improving models. Presentation at WCRP Open Science Conference: Climate Research in Service to Society. 24–28 October 2011. Denver, CO, USA., https://www.wcrp-climate.org/conference2011/orals/A4/Jakob_A4.pdf（最終閲覧日2018年8月2日）

気象研究所・海洋研究開発機構 (2018): MRI.COM ユーザーページ．http://synthesis.jamstec.go.jp/heroes/docs/html/（最終閲覧日2018年8月2日）

気象庁予報部数値予報課 (2017): 数値予報モデル開発のための基盤整備および開発管理．数値予報課報告別冊 **63**, 気象庁，108 pp.

NEMO development board (2018): NEMO users area. http://forge.ipsl.jussieu.fr/nemo/wiki/Users（最終閲覧日2018年8月2日）

NOAA-GMDL (2018): Modular Ocean Model. https://github.com/NOAA-GFDL/MOM6（最終閲覧日2018年8月2日）

小川明彦・阪井誠 (2010): Redmine によるタスクマネジメント実践技法，翔泳社．東京都，336 pp.

Pipitone, J. and S. Easterbrook (2012): Assessing climate model software quality: a defect density analysis of three models. *Geosci. Model Dev.*, **5**, 1009–1022.

Rasmusson, J. (2011): アジャイルサムライ−達人開発者への道．近藤修平・角掛拓未(訳)，オーム社．東京都，288 pp.

坂本圭・山中吾郎・辻野博之・中野英之・平原幹俊 (2013): 次世代日本近海予測モデル MRI.COM-JPN によるあびきの予測可能性．海と空，**88**, 15–28.

Tsujino, H., M. Hirabara, H. Nakano, T. Yasuda, T. Motoi, and G. Yamanaka (2011): Simulating present climate of the global ocean-ice system using the Meteorological Research Institute Community Ocean Model (MRI.COM): simulation characteristics and variability in the Pacific sector. *J. Oceanogr.*, **67**, 449–479.

Tsujino, H., H. Nakano, K. Sakamoto, S. Urakawa, M. Hirabara, H. Ishizaki, and G. Yamanaka (2017): Reference manual for the Meteorological Research Institute Community Ocean Model version 4 (MRI.COMv4). *Tech Rep.* **80**, Meteorological Research Institute, Japan.

辻野博之・坂本圭・碓氷典久 (2015): 気象庁気象研究所における沿岸モデル開発．沿岸海洋研究，**5**, 119–129.

Development management of Meteorological Research Institute Community Ocean Model (MRI.COM) using Git and Redmine

Kei Sakamoto[1]*, Hiroyuki Tsujino[1], Hideyuki Nakano[1], Shogo Urakawa[1], and Goro Yamanaka[1]

Abstract

Our ocean general circulation model, Meteorological Research Institute Community Ocean Model (MRI.COM), has been developed over almost 20 years; as a result, the source code has become larger and more complicated. Since it is now used for many projects, various departments and people of the Meteorological Research Institute and the Japan Meteorological Agency (JMA) are involved. To develop the model efficiently and maintain stability through widespread usage, we adopted tools and methods used in modern software development and redesigned our development management system. First, we introduced Git, which controls the development history (version) of the source code. This tool enables multiple developers to work on multiple tasks in parallel. Next, we introduced a project management system, Redmine, for all development members to share the development situation. This system records the development processes in a database one by one, which becomes a useful asset for other developers and next generation developers. By using these tools and clarifying the development procedure, we have achieved a new development system in which we can share information and perform mutual checks daily as a development team. This greatly contributes to improvement of code quality. JMA has built a system that centrally controls almost all models developed by Meteorological Research Institute and JMA based on Git (or SVN) and Redmine, which leads to great progress in development management and sharing of models.

Key words: numerical model, model development, model sharing, version control, project management

(Corresponding author's e-mail address: ksakamot@mri-jma.go.jp)
(Received on 25 January 2018; accepted on 12 June 2018)
(doi: 10.5928/kaiyou.27.5_175)

1 Meteorological Research Institute, Oceanography and Geochemistry Research Department, 1−1 Nagamine, Tsukuba, Ibaraki 305−0052, Japan
* Corresponding author: Kei Sakamoto
e-mail: ksakamot@mri-jma.go.jp