ラショナル統一プロセス ——ソフトウェア開発のベストプラクティス

Rational Unified Process—Best Practice for Software Development

野田勝彦，呉　暁星
安竹由起夫，岡村敦彦，荒井玲子

要　約　ソフトウェア開発の道しるべとしてのソフトウェア開発プロセスが，近年世界的に研究されている．数ある開発プロセスの中でも，開発ライフサイクルの管理，ベストプラクティスの効果的な適用，カスタマイズの容易さから，RUP（Rational Unified Process）が注目されている．

RUP は実際の開発で有効だった経験を反映しつつ成長した実際的なプロセスである．本稿では，RUP の歴史的背景から，RUP の概要，適用事例を紹介し，現在の RUP が抱える課題および将来動向について述べる．

Abstract　This paper reports on the Rational Unified Process through the history, the characteristic and the future.

In addition to describing a case study, it explains an implementation example of RUP.

RUP is a software development process that it has lately attracted software industry 's attention.

It excels the others in providing many guidelines for managing software projects, the effective way of implementing the best practices and customizable templates, especially.

1. はじめに

本稿はソフトウェア開発の有用な開発プロセスとして推奨している統一プロセス（以下，「RUP」）について述べたものである．RUP は実際の開発で有効だった経験を反映しつつ成長した実際的なプロセスである．本稿では RUP の歴史的背景から，RUP の概要，適用事例を紹介し，現在の RUP が抱える課題および将来動向について述べる．

2. RUP の歴史，背景

ここでは，RUP の歴史的背景について述べる．RUP は，多くの企業および開発に携わる人々の経験，実践を反映しつつ，改訂を加えて現在の RUP に至る．RUP の系譜を図 1 に示す．図中，中央の枠に囲まれている表記は RUP のバージョンを表し，左右に記述してあるのは，そのバージョンに追加された技術領域，対応する統一モデリング言語（Unified Modeling Language，以下「UML」）のバージョンを表す．

RUP の原点は，現在「コンポーネント開発」と呼ばれるエリクソン・アプローチである．エリクソン・アプローチは，1967 年 Ivar Jacobson により開発された．エリクソン・アプローチの特徴は，ブロックを使用してシステムをモデリングし，ブロックのインタフェースを定義していくということである．そのため，同じインタフェースが定義されているブロックを置き換えることにより，システム構成を変更すること

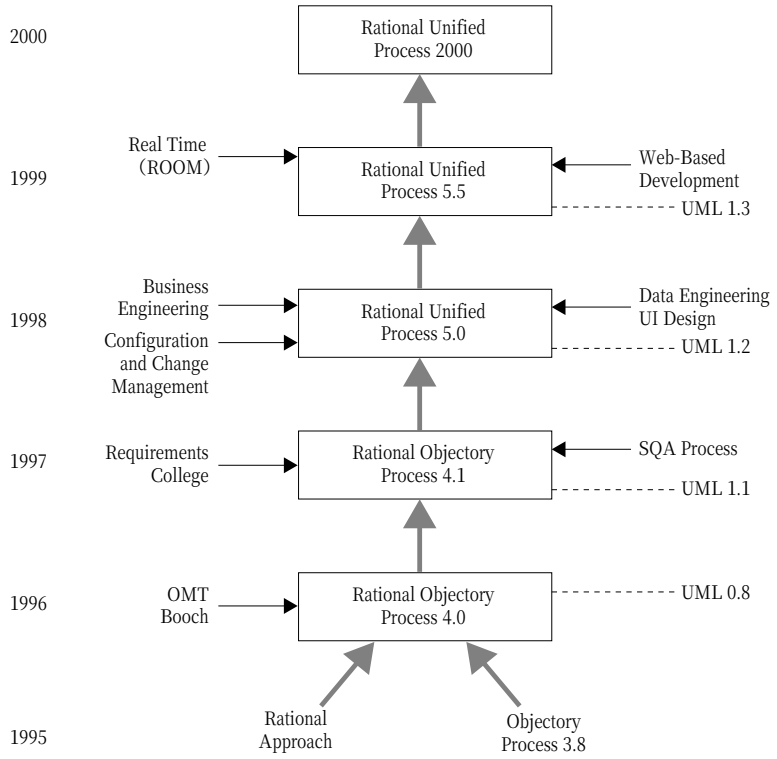

図 1　RUP の系譜

ができる．その後，Ivar Jacobson は，Objectory アプローチを開発する．Objectory アプローチは，「ユースケース駆動開発」と呼ばれ，モデル間の追跡可能性の実現を目的としている．Objectory アプローチの特徴のひとつに，Objectory プロセス自身をシステムとみなしていたという点があげられる．Objectory アプローチは，システムをメンテナンスすることと同じように Objectory のプロセスに改良を加え，進化させるという形で開発されている．

Objectory アプローチの中心であるユースケース概念と，オブジェクト指向設計方法論に，ラショナル社の経験と実践原則であるラショナル・アプローチが加えられ，1996 年に Rational Objectory Process 4.0 が，1997 年に Rational Objectory Process 4.1 が形成された．

ラショナル・アプローチの特徴は，アーキテクチャに重点をおいていることと，反復開発である．ラショナル・アプローチは，「アーキテクチャ駆動反復開発」と呼ばれている．また，反復開発における進捗を管理するために，四つのフェーズが考案された．四つのフェーズとは，方向づけ（inception），推敲（elaboration），作成（construction），移行（transition）である．Rational Objectory Process では，アーキテクチャをシステム構成の重要な部分として扱っている．また反復開発は，アーキテクチャを前提とした，リスク駆動アプローチへと進化した．なお，UML は，この頃開発され，Rational Objectory Process のモデリング言語として採用された．

その後，Rational Objectory Process には，要求管理や設計，テストといったプロ

セスの専門知識が組み込まれ，拡張された．具体的には，Requisite　Inc 社からは要求管理プロセス，SQA Inc 社からはテストプロセス，Pure－Atria 社からは構成管理の専門知識，Performance　Awareness 社からはパフォーマンステストとロードテストの専門知識，Vigortech 社からはデータエンジニアリングに関する専門知識が組み込まれている．また，ビジネスモデリング向けのワークフローも導入された．

1998 年，Rational Objectory Process 4.1 は，ソフトウェア開発のライフサイクル全体をサポートできるプロセスへと成長し，ラショナル社はこのプロセスをまとめ，製品化し Rational Unified Process 5.0 として発表した．これが現在 RUP と呼ばれるものである．RUP にはさまざまなソフトウェア開発分野における研究成果，開発経験が反映されたものであり，今日なお改良が続けられている．

3. RUP の概要

ここでは，RUP の概要について述べる．

3.1 ラショナル統一プロセスとは

ラショナル統一プロセス RUP とは，ソフトウェア開発プロセスであり，開発組織でのアクティビティと責務割り当てを行う際の系統的なアプローチを提供するものである．目的は，エンドユーザのニーズに応じた高品質のソフトウェアをスケジュールどおり納期遅れなしに提供することである．

また，RUP はプロセス製品でもある．RUP はラショナル社によって開発およびメンテナンスされており，ラショナル社が提供する開発ツールと統合されている．RUP 開発チームは，社内コンサルティング組織をとおして，顧客要求および経験を反映し，RUP を継続的に改良している．従って，既存の書籍ベースの方法論とは異なり，新技術がすぐに反映され，「生きた」プロセスとして随時アップデートされて顧客に提供される．

RUP は，開発全般に対するガイドライン，テンプレート，ツール・メンター[*1] のガイドラインといった知識ベースに簡単にアクセスすることによって，チーム開発を促進する．チームメンバーが同じ知識ベースにアクセスすることにより，要求，設計，テスト，プロジェクト管理，構成管理に関する問題はなくなり，チームメンバーが共通の言語，プロセス，ソフトウェア開発のビューを共有することができる．

RUP はモデルベースで開発を行っていくという特徴がある．各種仕様書といった大量のドキュメントに焦点をあてるのではなく，RUP はモデルをメンテナンスしていくことに重点をおいているため，開発中のソフトウェアをより正確に表現することができる．

また，RUP は UML を効果的に使う方法を提示している．UML は，要求，アーキテクチャ，設計を明確に扱うことのできる業界標準言語である．UML はラショナルソフトウェアで開発され，現在標準化団体である OMG（Object Management Group）でメンテナンスされている．

RUP はプロセスの大部分を自動化するツールによって支援されている．これらツールはソフトウェア開発プロセスにおける，様々な成果物，特にモデルを生成し，保守するために使用される．ツールには，ビジュアルモデリングに関するもの，プログ

ラミングに関するもの，テストに関するものなどがある．これらツールは，変更管理，構成管理などについても，すべての反復で関連付けされる．

RUP はソフトウェア開発のベストプラクティスを基本にして開発されている．したがって，RUP を使用することによって，ソフトウェア開発のキーアドバンテージの利点を得ることができる．ベストプラクティスについては，次節で詳しく述べる．

3.2 RUP の特色

RUP は次のような特色をもっている．

① ベストプラクティス
② ユースケース駆動開発
③ プロセスのカスタマイズ
④ ツールによるサポート

3.2.1 ベストプラクティス

ソフトウェア開発のベストプラクティスとは，それらを組み合わせて適用することにより，ソフトウェア開発上の問題の根本原因を解決できることが，開発現場で実証されているソフトウェア開発アプローチである．ベストプラクティスは，優れた能力を持つチームがソフトウェアプロジェクトを成功できるようにすることを目的として設計されている．

ベストプラクティスは，以下の六つの原則から成り立っている．

① 開発を反復すべし
② 要求を管理すべし
③ コンポーネントアーキテクチャを用いるべし
④ ビジュアルにモデル化すべし
⑤ 品質を管理すべし
⑥ 変更を管理すべし

【開発を反復すべし】

今日のソフトウェア開発では，プロジェクト開始時に要求が明確になっていることも，その要求がプロジェクト終了時までひとつも変更されないということも，ありえない．つまり，ソフトウェア開発は必ずリスクを抱えた状態で開始しなければならないのである．だからといって，要求がはっきりするまでプロジェクトを開始しない，というのは非現実的である．そこで，リスクの高いものから着手し，検証をかさねていくというアプローチをとる．これが反復開発の原型である．

RUP では反復開発を推奨している．もっとも重要な理由は，リスクの回避ということである．ウォーターフォール・アプローチではリスクを避けられたかどうかの検証は，ライフサイクルの終わりのほうになるまで，不明である．しかし，反復開発ではライフサイクルの最初のほうでリスクの検証が行えるので，早期にリスクを回避することができる．

また，要求の変化に対しても，反復開発であれば修正の機会は反復ごとにあるので，柔軟に対応できる．つまり，反復開発は「要求は変化するもの」，「プロジェクトにリスクはあるもの」という前提にたち，それら前提に対応するにはどうしたらよいか，という現実的な解決策であるといえる．

【要求を管理すべし】

要求管理とは，システムに対する要求を獲得し，定義づけ，体系化し，追跡するといった作業への体系的なアプローチを指す．効果的に要求を管理するには，要求を明確に定義するだけでは不十分であり，ひとつの要求が他の要求や成果物にどう関連しているのかという，追跡可能性を明確にする必要がある．なぜならば，要求は変化するものなので，初期の段階で完全に要求を定義することはできず，後に要求が変化したときにその変化の影響範囲を追跡するということは必ず行われるからである．

RUP ではユースケース・ドリブンの開発を推奨し，ユースケースを使って要求を体系化し，ユースケース単位に要求の影響範囲を追跡していくモデルを提案している．すなわち，ユースケースを開発プロセスの基本とするアプローチである．

【コンポーネントアーキテクチャを用いるべし】

ソフトウェア開発が大規模，複雑になるにつれ，アーキテクチャの重要性は増している．大規模プロジェクトでは，アーキテクチャをベースラインにして開発をすすめる方法をとる．これは，システムを段階的に成長させるために効果的な方法である．RUP では，コンポーネントベースのアーキテクチャを推奨している．コンポーネントベースのアーキテクチャでは，反復のたびにシステムを段階的に成長させることができるだけではなく，システムの変更や拡張もコンポーネント単位でできるので，変更の影響範囲を最小限に抑えることができる．また，市販のコンポーネントをシステムに統合することも可能であり，開発コストを低減することができるという利点がある．

【ビジュアルにモデル化すべし】

同じシステムに対して開発者同士が別の解釈をしていることがよくある．複数の開発者に情報を伝えるには，文章よりも図を使ったほうがより確実である．また，図の表記も建築の図面のように，誰が読んでもわかるという標準化されたものを使用したほうがよい．UML はシステムの成果物を視覚化し，仕様を規定し，文書化するためのグラフィカル言語である．また，UML は業界標準のモデリング言語であり，開発者の共通言語として使用することができる．

【品質を管理すべし】

品質確保に対する活動は，RUP の全ワークフロー，全フェーズ，全反復にわたって実装する．ライフサイクル全体に対する品質管理は，プロセス品質と製品品質の両方を実装し，測定し，評価をおこなうというプロセスをとおして行う．RUP ではプロジェクトでのすべての活動についてチェックポイントを提供し，品質を評価し，測定できるガイドを提供している．

【変更を管理すべし】

ソフトウェアが大規模化するにつれて，複数のプロジェクトを並行してすすめるという方法がとられる．その際，課題となるのは，複数の成果物に対するビルド管理，ワークスペース管理といったことである．反復開発を用いた段階的なリリースを調整するためには，ベースラインの確立と並行開発をしている要素間の追跡可能性を維持することが重要である．この制御をおこなうのが，変更管理である．RUP では，変更管理に対するワークフローといくつかの解決策を提供している．

3.2.2 ユースケース駆動開発

RUP はユースケース駆動のアプローチである．要求分析段階からユースケースを使ってシステム要求の体系化をおこない，開発計画，および開発活動をユースケース単位におこなっていく．ユースケースは膨大なシステム要素を意味のあるまとまりにする．システム要素間には関連や重複も存在するが，それらは追跡可能性を維持することで，全体的な矛盾を回避することができる．

3.2.3 プロセスのカスタマイズ

RUP は一般的で包括的な範囲を対象としている．RUP をそのままプロジェクトに適用することもできるが，通常は社内標準や，プロジェクト特有の事情によってそのままあてはめることは難しい．RUP はカスタマイズ可能なプロセスであり，カスタマイズするための開発キットを提供している．RUP を使用するプロジェクトは，プロジェクトで使いやすいように，フェーズ内での活動の削除や，成果物を追加，削除することができる．また，RUP に記載されている成果物のうち，プロジェクトにとって全く新しい成果物を作成する場合には，成果物のテンプレートも用意しているので，プロジェクトで使いやすいように修正して使用することもできる．つまり，RUP は厳格な手順を押し付けるものではなく，現実の開発にあわせてカスタマイズすることを考慮してあるプロセスである．

3.2.4 ツールによるサポート

反復開発や，変更管理にかかわる活動をすべて人手に頼るのは，現実的ではない．要求変更の追跡可能性を維持するには，人手に頼るよりも，ツールに頼ったほうが正確でもある．とりわけ反復型開発においては，ラウンドトリップエンジニアリングの中で，モデルとコードのインテグリティを保つ場合や，回帰テストにおけるテストの自動化などのツールを必要とする局面が多い．RUP には，RUP プロセスを支援するツールがひととおり用意されており，開発者は，開発者しかできない部分に専念することができる．RUP のツールメンターは，プロセスの各場面でどのようなツールを使用することができるのか，またそれをどのように使えば効果的なのかを示している．

3.3 RUP の構造

3.3.1 2 次元構造

RUP の全体構造を図 2 に示す．

RUP は 2 次元の構造を持っている．図中，横軸は時間の推移を表しており，プロセスの進行に伴なう動的側面を示し，サイクル，フェーズ，反復，マイルストーンといった用語で表現される．一方，縦軸はプロセスの静的側面を表し，作業，成果物，ワーカー，ワークフロー等の用語で表現される．

3.3.2 フェーズと反復

RUP は，ソフトウェアの開発サイクルを以下に示す四つのフェーズに分割する．

① 方向付け（Inception）フェーズ
② 推敲（Elaboration）フェーズ
③ 作成（Construction）フェーズ
④ 移行（Transition）フェーズ

これら四つのフェーズは，意思決定が必要とされるマイルストーンで区切られてい

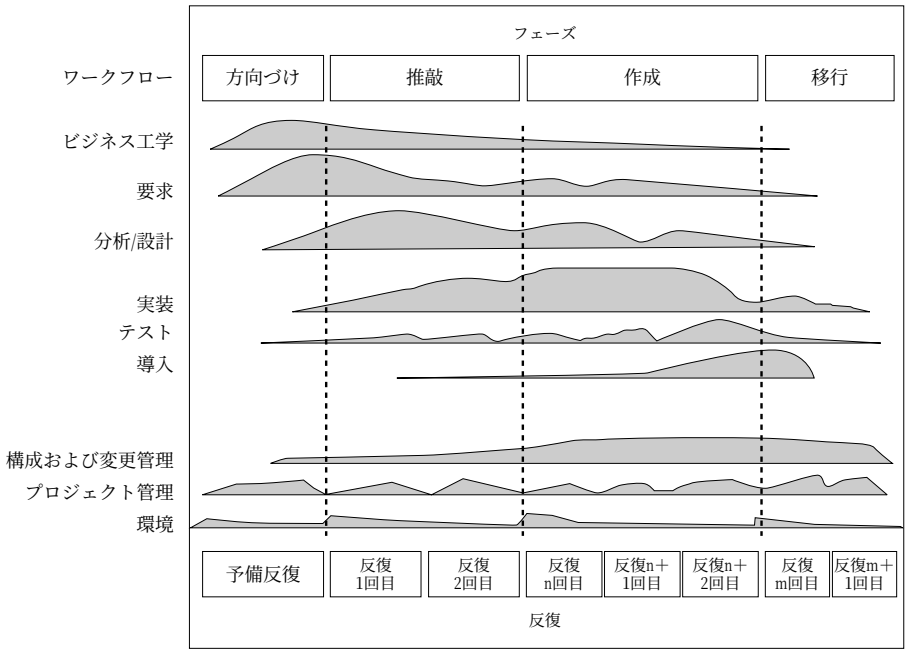

図 2　RUP の全体構造

る．なお，それぞれのフェーズの意味は，ウォーターフォール・アプローチで用いられているフェーズ（分析，設計，実装，テスト）とは異なることに注意する必要がある．以下にそれぞれのフェーズについて述べる．

なお，各マイルストーンの定義および名前については，参考文献[9]を参照している．

【方向付け（Inception）フェーズ】

方向付けフェーズは，開発対象となるシステムの範囲と規模を設定することを目的としたフェーズである．通常これらの内容は，開発企画書として記述する．まず，このフェーズで行うことは，システムとシステム外部との最も初期レベルのやりとりを規定し，システムの骨格となるユースケースを導き出すことである．次に，このユースケースを基準にして，リスクの評価，リソースの見積もりを行い，フェーズの計画をたてるということを行う．方向付けフェーズの終了は，「ライフサイクル目標（LCO）」マイルストーンである．

【推敲（Elaboration）フェーズ】

推敲フェーズは，四つのフェーズの中で最も注意すべきフェーズである．推敲フェーズの目的は，問題領域を分析し，アーキテクチャのベースラインを確立し，主要なリスクに対処し，プロジェクト全体を計画することである．従って，このフェーズでは，実行可能なアーキテクチャプロトタイプが，1 回もしくは複数回の反復によって開発される．具体的には，方向付けフェーズで識別された技術的に重要なユースケースが，このフェーズで実装され，利害関係者にレビューされることになる．

推敲フェーズの終了は，「ライフサイクルアーキテクチャ（LCA）」マイルストーンである．LCA では，対象となるプロジェクトが，本格的な構築作業に取り掛かって良いかどうかの意思決定がなされる．

【作成（Construction）フェーズ】

作成フェーズの目的は，推敲フェーズでは取上げなかったユースケース，つまり技術的にリスクの少ないコンポーネント，アプリケーション機能を実装し，テストし，システムに統合することである．このフェーズは製造工程という側面をもち，リソースの管理や運用の最適化に注意が払われる．大規模プロジェクトでは，作成フェーズで並行開発を採用し，開発効率をあげている．作成フェーズの終了は，「初期運用能力（IOC）」マイルストーンである．このマイルストーンでリリースされるソフトウェアを「ベータ版」と呼んでいる．

【移行（Transition）フェーズ】

移行フェーズの目的は，エンドユーザの環境，つまり実行環境にソフトウェアを移行し，ユーザ自身で運用できる環境を整えることである．ここでは，納品に向けたソフトウェア品質を確保し，ソフトウェアを使用するためのドキュメンテーションを作成する．移行フェーズの終了は，「製品リリース」マイルストーンである．

【反復】

上で述べた四つのフェーズは，それぞれさらにイタレーションと呼ばれる小さな反復に分割される．各イタレーションは，ビジネスモデリングから実行可能形式モジュールを作成するまでを含む．システムは，反復を繰り返すごとにインクリメンタルに機能が拡張されて行き，最終的に完成したシステムに成長する．

3.3.3 プロセスの静的構造

プロセスとは，誰が，いつ，何を，どのように行うかを表すものである．RUP では以下の要素でプロセスを表現する．

① 誰が：ワーカー
② どのように：作業
③ 何を：成果物
④ いつ：ワークフロー

なお，ワークフローはソフトウェア開発のビジネスモデルに基づき，以下の九つの基本ワークフローがある．

① ビジネスモデリング
② 要求
③ 分析/設計
④ 実装
⑤ テスト
⑥ 導入
⑦ プロジェクト管理
⑧ 構成/変更管理
⑨ 環境

上記ワークフローは，ソフトウェア開発ライフサイクルにおける各「フェーズ」内の各「反復」で，必要に応じて何度も実行される．各ワークフロー内では，「ワーカー」が「作業」を行い，「成果物」を生み出していくことで，反復型開発を遂行する．

3.4 RUPの強み

RUPには次のような三つの強みがある．

第1に，「RUPは，反復開発，要求ドリブン，アーキテクチャベースのアプローチといった，ソフトウェア開発のベストプラクティスをもとにしている」ということである．ベストプラクティスは実際のプロジェクト開発で得られた数々の経験を反映したものであり，理論的に体系化したので，単なる経験則ではなく実践できるものである．

第2に，「RUPは，各反復におけるプロトタイプ開発，フェーズの終了基準といったように，開発プロセスを可視化し，管理しやすくする様々なメカニズムを提供している」ということである．これらのマイルストーンを明確に定義することで，これまでの反復型開発にありがちであった主観的な評価基準から曖昧さを排除し，厳密で管理可能なプロセスとして再構築している．主マイルストーンをクリアしなければ次のフェーズに進むことはできず，サブマイルストーンをクリアしない場合は，反復さえも終了できない．ただし，その評価基準自体はプロジェクトごとに調整可能であるため，あらゆるプロジェクトで適用し得る柔軟性も同時に備えている．

第3に，「RUPの定義はHTML形式で製品化されているため，個別の組織にあわせてRUPのカスタマイズが可能になっている」ということである．

4. 適 用 事 例

これまで述べてきたRUPの具体的実装例として， カナダ航空管制システム（以下，「CAATS」）への適用事例を以下に述べる．

4.1 RUP導入の背景

CAATSは，カナダ航空の管制システムであり，航空機の航路を監視し制御をおこなうシステムである．CAATSの開発を請け負ったソフトウェア開発会社は，システム操作概要とその他の要求に同意が得られた後，その要求の複雑さ/規模からラショナルソフトウェアが推奨していたRUPを適用することを決定した．CAATSプロジェクトにおける反復開発は，推敲フェーズからの適用することとした．

4.2 反 復 開 発

CAATS開発プロジェクトでは，約6〜8ヶ月の期間をもつ反復を7回以上実施し（図3参照），各反復の終了後にはその反復内の作業/成果物がレビューされ，次期反復の目標/評価基準が調整された．

初期の反復（1〜3）は推敲フェーズに実施されており，ソフトウェアアーキテクチャの確立が目標となったが，安定したアーキテクチャが達成されるまでは，特にソフトウェア要求の中で最もリスクが高いものをクリアすることに注意が注がれた．各反復のレビューでは顧客を交え，成果物の品質を確認した．また，新たに発覚したリスクや組織の変更など，開発プロセスに影響を与える事象に関しては，次の反復の計画（目標/評価基準）に反映させた．

他のソフトウェア要求は，作成および移行のフェーズ内の反復4〜7で実装され，各反復の成果物（含むソースコード，実行ファイル）を次の反復への入力として使用し，機能追加および洗練作業が行われた．

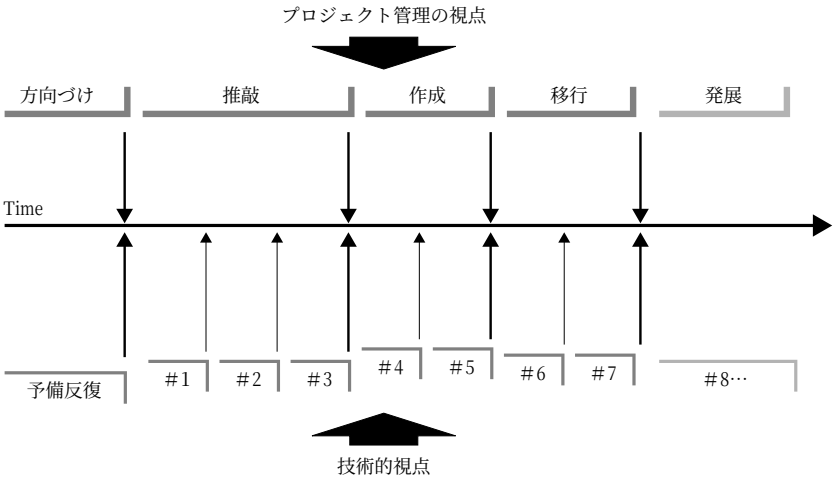

図 3　反復の視点

4.3　各反復の作業内容

各反復で行われた開発作業で，重要なものを以下に挙げる．

1)　反復＃1：準備段階

この反復では，技術的に困難で，顧客の優先順位としても高い四つの重要なユースケースに注目し，候補となるソフトウェアアーキテクチャを選定する事を目標にした．分析・設計および実装のワークフローでは，基本的な航空管制（以下 ATC）機能のみに集中し，そのパフォーマンスと実装のリスクを軽減するアーキテクチャを選定した．また，アーキテクチャの選定に伴い，今後の反復で必要となる開発ツール，作業ガイドライン，作業工程に関する教育をプロジェクトメンバーに対して行った．

2)　反復＃2：基本となるメカニズムの実装

2 回目の反復は強靭なアーキテクチャの確立を目標とし，他の基本的な ATC 機能とそれをサポートするメカニズムの実現を行った．対象となったメカニズムは，分散性（プロセス間通信），永続性（データベースアクセス）等である．また，並行して反復＃1 の成果物であった作業ガイドラインと作業工程を以後の反復のために洗練し，完成させた．

3)　反復＃3：アーキテクチャの確立

この反復では，アーキテクチャがサポートすべき全てのメカニズムを実現し，アーキテクチャを完成させレビューを行った．アーキテクチャパターンとしては共通システムサービス，再利用性の高い共通オブジェクト等のグループ化を行い，基本的なレイヤ構成をとった．このアーキテクチャの概要は参考文献[7]，基本レイヤに関しては参考文献[8]を参照されたい．

4)　反復＃4：基本飛行プランの計算

ATC のうち，基本飛行プランの計算に関するユースケースの実現を行い，関連する要求が実装された事を顧客とレビューをし，確認した．

5)　反復＃5：無線航法

ATC システムの核となる部分のビルドとテスト．ATC による航路制御，飛行プランの計算，無線および手動による飛行制御のモニタリングに関するユースケースの実現を行った．

6） 反復＃6：ベータ 1

移行フェーズに入り，初期ベータをビルドし実動環境でのテストを行った．人的作業に関わる機能を自動化した部分を主にテストした．

7） 反復＃7：ベータ 2

残りの自動化部分を含めたビルドと実動環境でのテストを行った．

4.4 経験より得たもの

大規模なシステム開発でよく見られるように，本プロジェクトにおいても全てを当初の予定どおりに進める事は不可能であった．ここでは，今後同じようなプロジェクト戦略を取り入れようとしている方が興味をもたれるであろう部分に関して，本プロジェクトを通して得た経験をまとめる．

1） 利害関係者全員の意思統一が必要

ウォータフォール型開発から反復型開発へ開発工程を変更するためには，全ての利害関係者（開発組織，全レベルのプロジェクト管理者，顧客，エンドユーザ）の意見の一致が必要である．そして新しい“ゲームのルール”を確実に彼らに伝える必要がある．これが実施できない場合，議論，正当化，誤解等，“前にそう話したではないか？”的な事柄に多くの無駄な時間を過ごすことになる．

2） 実行可能なアーキテクチャとは美しいものではない

推敲フェーズで確立されるアーキテクチャは，スケルトンであり，またシステムとして完全ではないが，ビジネスロジックとの統合の際のリスクを軽減するために“美しくなくとも”実行可能である必要がある．特に，ユーザインタフェース部分のプロトタイプが他の部分と分割されている場合は，それをデモ後に捨て去る覚悟が必要である．似たような複数のアーキテクチャプロトタイプを保持し拡張すると思わぬ後戻りが発生する．また，インクリメンタルな開発とは，インクリメンタルな納品（T. Gilb[5]）とは意味が違う．インクリメンタルな開発とは，反復ごとにシステムを拡張し成長させつつ開発するものであり，各反復の終了時ではシステムは不完全の状態である．一方，インクリメンタルな納品とは反復の単位でそのたびに完結したシステムの納品を行っていくというものであり，反復の終了時にはシステムはその都度，完成した状態にある．インクリメンタルな納品は，非常に成熟したプロダクトにおいて有効であり，前例のないシステムでは不可能と考えてよい．

3） 棚上げの要求を残さない

反復開発では，要求を棚上げされることはなく，最終的にシステムにすべての要求が反映され，実現される．なぜならば，反復開発では，推敲フェーズにおいて各要求に注目し，重要なユースケースおよびリスクの高いユースケースをもとにシステムの骨格となる設計とプロトタイピングをおこなうことによってシステムのベースラインを作成するからである．もちろん要求の変更は必ず発生するので，各ワークフローは互いに連携している必要がある．

4) 後戻りを最小限に

反復プロセスは後戻り作業と混同されがちである．しかし，“今なにが必要なのか”という作業とは違い，反復プロセスは“物事をベストの状態にする”ことを可能にする作業である．

5) 反復の完結

反復開発アプローチの利点は，それぞれの反復が完結したかどうかでのみ実現される．たとえば，そのソフトウェアが実際に統合されテストされたかどうかを確実にレビューするという事である．これを省略するとリスクの軽減やプロセスの成熟は成されない．この点では，本プロジェクト初期の三つの反復がそれぞれ6ヶ月であったのは，期間的には短かったのかもしれない．反復の期間はその反復にどれぐらいの人数が関わるかによるが，長い反復を組むことは，プロジェクトの初期段階からメンバに対してスケジュールのプレッシャー（または士気）を与える事ができるというメリットを半減させてしまう．

① 各反復で学んだ事を次の反復に活かす

特に，開発プロセスと工程を洗練するために必要である[4]．

② 同時期に多くの変革を行わない

プロジェクトメンバーの学習曲線（学習にかかる作業量）は初期の三つの反復で非常に高くなった．これは多くの新しい事柄が開発チームにふりかかったためであり，段階的な教育を考慮すべきであった．

5. RUPの課題

この章では，RUPの課題について述べる．

RUPには次のような課題が指摘されている．

① RUPは開発のプロセスであり，保守，サポートフェーズを対象としていない

② 複数のプロジェクトによる大規模開発に対応できない

③ RUPはツールベンダーに依存している

④ RUPはメトリクス管理，再利用管理，人的管理，テストに関して弱い

これらの指摘の中には，RUPの課題であるものと，RUPへの誤解が混在している．

たしかに，メトリクス，再利用管理，人的管理について，RUPは定義していない．これらの領域は実際のプロジェクトに依存する問題であるため，実際にRUPをプロジェクトに適用する場合には，プロジェクトに適したようにRUPのカスタマイズもしくは，すでに存在する企業標準への適用をおこなう必要がある．また，テストに関しては，抽象的なレベルでの定義はされているものの，実際的な適用に際しては，カスタマイズが必要である．したがって，4番目の指摘はRUPの課題といえる．

次に1番目の指摘である，保守およびサポートフェーズに関して述べる．指摘は，RUPは開発部分を対象としたプロセスであり，保守やサポートを対象としていないというものである．たしかに，RUPの全体像からは保守，サポートフェーズがないという印象をうける．しかし，この指摘は開発プロセスとソフトウェアのライフサイクルを混同していることから派生したものである．RUPは保守，サポートを含むソフトウ

ェアのライフサイクルを構成するものとして位置づけている．したがって，保守やサポートフェーズは，RUP ではそれぞれ別のプロセスとしてとらえており，保守，サポートで RUP を適用して行うことは可能である．

次に 2 番目の指摘である，複数のプロジェクトによる大規模開発に対応できない，という指摘に関して述べる．RUP はプロジェクト規模を限定してはいない．この指摘は，RUP 適用に対する誤解から派生している．

RUP 適用に際する誤解のひとつに，プロジェクトを構成するサブプロジェクトすべてに RUP を適用しなくてはならない，というものがある．たしかにすべてのプロジェクトを RUP で統一的に実施することは理想的ではあるが，現実的ではない．大規模プロジェクトの場合，RUP 適用の方法として二つの方法がある．

第 1 に,「全体プロセスに RUP を適用し, 個々のサブプロジェクトは RUP でも RUP でなくともよい」とする方法である．この場合は，プロジェクトのマスター計画において，反復回数，アクティビティ，成果物を規定し，どのイテレーションで段階的なリリースを行うのかといったマイルストーンを設定し，サブプロジェクトは，マスター計画で設定したマイルストーンにあわせて，開発プロセスを設計するということになる．

第 2 に「全体プロセスは従来使用していたプロセス，または社内標準プロセスを適用し，個々のサブプロジェクトのいくつかに RUP を適用する」という方法である．この場合も，第 1 の方法と同様に，マスター計画のマイルストーンを基準にして個々のサブプロジェクトのプロセスを設計すればよい．

最後に，3 番目の指摘である，RUP はツールベンダーに依存しているという指摘に関して述べる．たしかに製品としての RUP はラショナル社が提供しており，その製品にはラショナル社で提供している開発ツールの操作方法をはじめ，どのフェーズでどのツールを使用するとよい，といった内容が含まれている．製品としての RUP を初めてみた場合，これら開発ツールがないと RUP で開発することはできないのではないか，といった印象を受ける可能性はたぶんにある．しかし，これは RUP と開発ツールの位置付けに対する誤解である．RUP はプロセスを支えるものであり，開発ツールはプロセスにおいて行うアクティビティをサポートするものである．したがって，開発ツールは代替可能であるので，ラショナル社の製品でもよいが，他の製品でもよいのである．RUP が提供しているドキュメントのテンプレートもラショナル製品には依存していないので，従来使用している開発ツールを使用しても問題は発生しない．またドキュメントのテンプレートはあくまでもテンプレートであるので，プロジェクトの標準ドキュメントや社内標準ドキュメントを使用しても RUP から外れていることにはならない．しかしながら，またラショナル社の開発ツールは RUP を適用したプロジェクトを考慮しているため，反復開発など RUP のプロセスに対応しやすくなっているため，RUP 導入負荷は低く抑えられる．

したがって，RUP の課題としては 4 番目の指摘である「メトリクス管理，再利用管理，人的管理，テストに関して弱い」ということだけがあげられる．

6. 今後の動向

最後に，RUP の今後の動向について述べる．

2001 年 1 月，はじめての日本語版の RUP がリリースされた．この日本語版のリリースは，日本の開発者にとっては RUP の歴史上最もインパクトのある出来事であるとともに，RUP にとっても英語圏以外への RUP の普及を促進するための大きな第 1 歩となる．この初の日本語版のリリースは，バージョン 2000 である．前回のリリースであるバージョン 5.5 からの大きな変更点として，e-ビジネスアプリケーション開発のための拡張があげられる．これは，経済社会の変化に伴い，e－ビジネスアプリケーション開発への期待が急速に高まったことへのラショナルの回答と言える．また，次にリリースされる予定の RUP バージョン 2001（2001 年夏頃を予定）では，e－ビジネスアプリケーション開発をさらに具体的に支援するため，マイクロソフトや IBM の Web ソリューションプラットフォームに対応したフレームワークを提供する．また，Web アプリケーション開発で多く見られるような小さなプロジェクトでも RUP を採用し効率的にシステム開発が行えるよう「小さなプロジェクトのためのロードマップ」の提供も興味深い．

ところで，RUP はカスタマイズして導入することが考慮されている．しかし，RUP を手作業でカスタマイズするには，ある程度の限界があると言わざるを得ない．そこでラショナルは，大掛かりなカスタマイズでも効率的なカスタマイズを可能とするための Rational Process Workbench と呼ばれるツールの提供も行っている．残念ながらこの RPW の日本語版の提供はまだスケジュールされていないが，近い将来に RPW も日本語化されることが期待される．この RPW がリリースされれば，日本国内においても RUP のカスタマイズを比較的に容易に行うことが可能となり，本格的な RUP の普及が予想される．

＊ 1　特定のソフトウェア・ツールを使う，特定のプロセス作業やプロセス・ステップの実行方法に関する実用的な手引き

参考文献
［1］ フィリップ・クルーシュテン「ラショナル統一プロセス入門」ピアソン・エデュケーション, 1999 年 12 月.
［2］ Philippe Krucheten “ The Rational Unified Process An Introduction Second Edition ” Addison-Wesley, 2000.
［3］ Scott W. Ambler “ Completing The Unified Process With Process Patterns ”.
［4］ D.Emery and J. Mdhur “ HCSD Unit development Process : stepwise process improvement, ” Proc. of Ada-Europe Conf., 1996.
［5］ T.Gilb, “ Principles of Software Engineering Management ”, Addison-Wesley. Workingham, UK, 1988.
［6］ Philippe Kruchten and Christopher J. Thompson “ Iterative Software Development for Large Ada Programs ”, Proc. of Ada-Europe Conf., 1996.
［7］ Philippe Kruchten and Christopher J. Thompson “ An object-oriented architecture for large scale Ada programs, ” Proc. Tri-Ada ’94, Baltimore, 1994.
［8］ Christopher J. Thompsonand V. Celier, “ DVM : an object-oriented framework for building large distributed Ada system, ” Proc. of Tri-Ada ’95, Anaheim, Ca., Nov. 1995.
［9］ Barry Boehm “ Anchoring the Software Process. ” IEEE Software, July pp. 73-82, 1996.

執筆者紹介　野　田　勝　彦（Katsuhiko Noda）
日本ラショナルソフトウェア(株)プロフェッショナルサービス部部長．多種多様なソフトウエア開発プロジェクトの経験を持ち，プロジェクト管理が専門．プロジェクトにおける技術的側面と人間的側面のかかわり方にも関心をもつ．

呉　　暁　　星（Robert Wu）
日本ラショナルソフトウェア(株)ソフトウェアエンジニアリングスペシャリスト．ソフトウェアテスト技術が専門．テストという視点からみたソフトウェア品質の向上に関心をもつ一方，ダブルバイト市場へのプロダクト展開にも興味をもつ．

安 竹　由 起 夫（Yukio Yasutake）
日本ラショナルソフトウェア(株)ソフトウェアエンジニアリングスペシャリスト．要求分析，システム分析・設計が専門．フィッシュ理論愛好者．

岡　村　敦　彦（Atsuhiko Okamura）
日本ラショナルソフトウェア(株)ソフトウェアエンジニアリングスペシャリスト．構成管理を中心にソフトウェア開発のあらゆる側面を担当する，ナレッジリポジトリ．根底にプログレ不滅の精神を持つ．

執筆者紹介　荒　井　玲　子（Reiko Arai）

日本ラショナルソフトウェア(株)ソフトウェアエンジニアリングスペシャリスト．システム分析・設計，要求分析が専門．音楽とソフトウェア開発，美しい構造に興味をもつ．仕様化言語 Z 愛好者．