AMSの拡張開発におけるRUPのカスタマイズ

畑 秀明

Customization of RUP for Applying to Enhancement Projects Contracted as AMS

Hideaki Hata

本論文はラショナル統一プロセスRUP（Rational Unified Process®）をAMS（Application Management Service）の拡張開発に適用した際の，成果物と作業計画立案のカスタマイズ方法のヒントを提示する．成果物の観点では作成と修正という2つの特性に見合った成果物作成のヒントを説明し，作業計画の観点では機能追加と機能変更の2つの特性と，他の要素との組み合わせでよりよい計画はどうあるべきかを説明する．

This paper provides some hints for customizing both the work products and activity plan of RUP (Rational Unified Process®) in case it is applied to projects to enhance existing systems contracted as AMS (Application Management Service). The work products in an existing system enhancement project should be different in their needs, characteristics and creation methods from a project to develop a new system from zero. Some changes are suggested to be made to the ways in which the RUP method is used. To enhance an existing system, there are two cases, development mainly for function additions, and development mainly for function changes. The nature of the enhancement and other factors causes changes to activity plans. Especially development strategies centering around the 'use cases' vary from one case to another.

Key Words & Phrases ：RUP, AMS, 拡張開発, 成果物, 作業計画
RUP, AMS, Enhancement, Work Product, Activity Plan

1．はじめに

e-ビジネス時代における開発プロセスは年々重要さを増してきているが［1］，近年ではラショナル統一プロセスRUP（Rational Unified Process®）［2］およびこれを応用したプロセスが業界で標準となってきている．ただし，このプロセスはカスタマイズを前提とし（［3］の2.4章を参照），新規開発を前提としたプロセスである．昨今ビジネスとして伸びているAMS（Application Management Service）における拡張開発においてRUPを使用する際には，ある程度カスタマイズが必要となる．このカスタマイズにはRUPを熟知した上で，その拡張開発プロジェクトのプロセスを定義する必要がある．この際，RUPの本質をはずさないために，RUPそのものはもちろんのこと，RUPで重要な基本10要素［4］や，プロセス自体の目的［5］などを十分に理解しておくことが重要となる．

今回はこうしたRUPの基本概念の上に立ち，拡張開発にフォーカスして，さらに一歩進んだ実践レベルでのカスタマイズについて説明していくが，この検討にあたっては小規模におけるRUPの適用［6］や，1人でのRUP適用［7］といった極端な形態を参考にして各成果物や作業の意図を理解し，そこへ著者が長年関わったいくつかのAMSプロジェクトでの実践経験を加味して，現場のプロジェクトチームにより分かりやすいことをねらいとする．

最初にRUPにおける拡張開発の考慮点を検討し，それに対応するための戦略を提示し，成果物のカスタマイズのヒントと作業計画，特に反復計画立案のヒントを述べて，最後に健全なプロジェクト運営のための考慮点をまとめる．

提出日：2003年8月28日

2．拡張開発でのRUPの課題

まずRUPにおいて，拡張開発のためのカスタマイズが必要となる要因を考察する．これには主に新規開発と拡張開発の違いを考察し，そこにRUPの特徴を照らし合わせてどのような点にカスタマイズが必要

かを検討する.

2.1 要件定義

新規開発におけるRUPの要件定義は, システムの要求元(システム開発を要求する人・組織)の漠然とした要求を正しく把握することに重心を置いている. システムがどんなものになるか分からないステークホルダーに対し, システムの振る舞いを洗い出していくことで, 要求をシステム要件に落としていくユースケースのテクニック[8]などはその典型といえるであろう.

一方, 拡張開発の要件定義は, 通常は現存するシステムを元に, それに対する「追加」「変更」「削除」という形で表現される. つまり, 要求元は, 今回の開発の結果のシステムがイメージできており, 詳細な機能要件を提出してくる.

現存するシステムの振る舞いを示すユースケースがそろっているケースでは,“ そのユースケースに対する「追加」「変更」「削除」”という形で要件を定義することはできるであろう. 存在しない場合は,「ユースケースの部分的な変化(追加・変更・削除)を表現するため」だけにユースケースを作成しなければならなくなる.

要求元にとってみると「A画面にTテーブルのF項目を追加してほしい」という要求を表現することだけを考えると, A画面に関わる一連のシステムの振る舞いをユースケースというもので書き起こさなければならない労力は多大なものになる一方で, それによる恩恵は薄いものとならざるを得ない.

2.2 テスト

新規開発の場合, プロジェクトで構築した機能の全てをテストすることが原則である.

一方, 拡張開発の場合のテストは, 大きく以下の2つに分類される.

(1)今回のプロジェクトで機能に変化のあった部分のテスト

(2)今回のプロジェクトでは機能に変化はなかったが, 変化のなかったことを確かめるテスト

依然としてシステムの有する全ての機能をテストするのが品質確保の点では望ましいが, コストがかかるため, (2)に関してはコストとワークロードのバランスを取る必要がある. このためには, テストに対する要件が何であり, 戦略をどう立案するかが, 新規開発時に比べてより重要となってくる.

2.3 アーキテクチャ

新規開発ではアーキテクチャの作成がシステムの性能を大きく左右するため, 大きなワークロードを投入することになるが, 拡張開発ではアーキテクチャに変更がない場合, このエリアにそれほど大きな投資は不必要である.

その拡張開発でアーキテクチャの変更の有無があるかどうかは, 非常に大きな意味をもつ.

2.4 ビジネス・オブジェクト・モデリング

アーキテクチャ構築同様, RUPにおいて新規開発では設計の柱となるビジネス・オブジェクト・モデルを作成するが, 拡張開発では,“ それがある ”状態を前提とする. 前回のプロジェクトでオブジェクト指向型開発プロセスを採用していなくても, 既存のクラスにはそれに相当するクラスが存在するはずである.

RUPではビジネス・オブジェクト・モデリングは要件定義でもかなり上流に位置しており, 問題は, 拡張開発においても, それを行う必要性があるかどうかということになる. 前述のように拡張開発においては, 要件は設計などに密接に結びついているため, そこまでさかのぼる必要があるか判断が難しいことがある.

3. 拡張開発のための戦略

ここまでに見てきた課題に対して, これを解決するための方向性を検討する.

3.1 反復の一部として位置付ける

アーキテクチャもあり, ビジネス・オブジェクト・モデルもあるという状態は, RUPでいうなら作成フェーズ以降と同じことになる.

今回の要件を実現することにリスクが少なく, リスク軽減のために特別な配慮を必要としないのであれば, その拡張開発はまさに複数の反復の途中の1つとみなされるので, 大きな問題にはならない.

しかしながら, RUPによる反復であっても, 前の反復があるからこそ, 今の反復があるわけであり, 途中から反復を実施するということはそれほど単純ではない. 既存のリソースを, プロジェクトの成果物にスムーズに移しかえることが必須となる.

3.2 ユースケースの目的の認識

2.1で述べたように, ユースケースを作成する動機は拡張開発においては単純ではない.

それでも筆者は拡張開発でもRUPの柱である「ユースケース駆動」にこだわる. その理由は, この考え方の中で言及されているユースケースの目的のうち, 下記のものは拡張開発であっても有効であるからである.

- テストケースのインプットとして
- システムを理解するため
- 作業管理の基礎として

ここでテストケースのインプットとしての意義を高めるのであれば, 2.2で述べたようにテストのスコープを定義した上で, それを網羅するユースケースは作成するべきである.

いずれにしても, ユースケースの目的を再認識した上で, プロジェクトの環境やコストなどの制約と, そのプロジェクトでユースケースのもたらす価値とのバランスを取ることが拡張開発では非常に大事になる.

3.3 モデルのリバース

現行システムを表現する設計モデル(クラス図, シーケンス図)が保守資料として存在したとしても, 今回の拡張開発に照らし合わせた際には, 視点が異なる可能性がある. その際には, 拡張開発の要求にあう設計モデルを起こさなければならない.

現行システムが稼動しているということは, モデルのソースとクラスファイルが存在するということである. つまり設計モデルを起こすといっても新規に作成するのではなく, これらの稼動モジュールそのものを使用することになる.

このためには, ソースやクラスファイルから設計モデルを簡単に生成(リバース)できる方法があると非常に効果的である. RUPではこのようなソースからモデルへのリバースとモデルからソースの生成を繰り返すラウンドトリップ開発を採用している[2]が, この戦略をそのまま使うことになる. 現在出回っているモデリング・ツールはこのラウンドトリップ機能を保有しているものが多く, 拡張開発には大変有効である.

4. 拡張開発における成果物

本論文の中心である拡張開発のための成果物作成のヒントを詳述する.

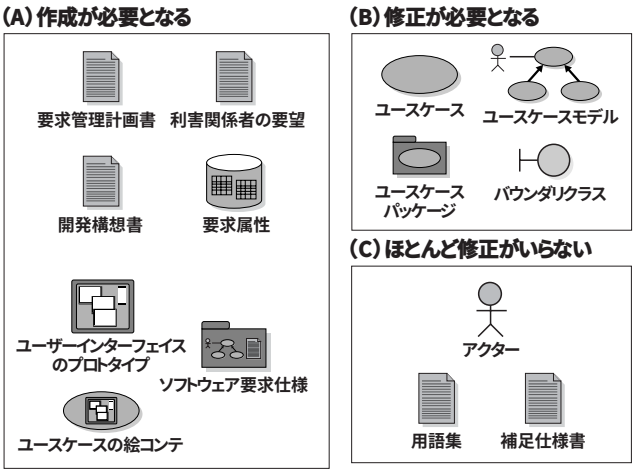

図1. 要求の成果物セット－拡張開発編
(RUP[2]より抜粋の後分類)

ただし拡張開発といっても, ベースとなるアクティビティと成果物はRUPのもので, 新たな成果物を提案するものではない.

次から述べるのは成果物について, 新規に作成するのか, 修正をするのかを色分けするものであり, この情報を元にプロジェクトの計画を考えるヒントを与えるものである.

なお, 本章ではポイントを絞るため重要なワークフローである, 要求, 分析・設計, 実装・テストのワークフローだけを取り上げる.

4.1 要求ワークフロー

4.1.1 目標

要求を把握する作業は, 拡張開発だから特別なことをするのでなく, あくまでRUPに従って行われるべきであろう. この際に必要となる成果物セットを**図1**に一覧した.

利害関係者の要望をまとめて, 優先順位を付けて管理する作業は通常の開発と同じである.

4.1.2 概要

プロジェクトの最初は開発構想書から始まるのはRUPのままである. 新規開発では, 求めているシステムがぼんやりとしているため, 開発構想書にもシステムを意識する記述はそれほど多くはならないと思われるが, 拡張開発では利害関係者が実際のシステムをイメージしているため, 内容としてはシステムへの機能要求に終始してしまいがちである. そのため, なぜそのような要求があるのかを的確に客観的に把握するために, ユーザーオペレーションやビジネスの変化などは正しく記述するべきである.

拡張開発においても新しいシステムへの要求を正しく把握するためにはプロトタイプは必要である. 既存システムをうまく利用してプロトタイプも迅速に生成できるように工夫したい.

一方で, 既存システムがあるため, ユースケース仕様などは拡張部分の記述を追加・変更することで表現するのが自然だと考えられる. これらは図1中では(B)のグループであり, (A)とは異なり, 何もないところから作成するものではないことを示している. ユースケース仕様以外はダイアグラムの類いなので, 変更部分もビジュアルに表現できることころがRUPの利点でもある.

また, 図1中の(C)グループは, システムのライフサイクルにおいて変化が少ないと考えられるため, 拡張開発ではほとんど手を付けない成果物とする. この(C)グループに変更が入るとシステムに大きな影響を与えるので, 変更の有無は慎重に判断すべきである.

4.1.3 カスタマイズ

RUPではプロジェクト中に発生する成果物への変

更は変更要求（CR:Change Requests）で管理される[2]. これを応用して（B）グループのシステムへの変更部分をCRで表現することで, システム変更管理を容易にすることが可能と考えられる.

この場合, 追跡性の観点で, その元となる開発構想書や利害関係者の要望書などは省略すべきではない.

4.2 分析・設計ワークフロー

4.2.1 目標

拡張開発における分析・設計の作業の特徴は, 現行設計の分析が必要な点であるが, これも成果物としては特別な形をとらず, 通常のRUPの成果物を使用する. この一覧を**図2**に示したが, 組み込み系の開発の特色の強い成果物およびDB設計の部分を除いて, 従来の業務アプリケーション開発での分析と設計との相違に焦点を絞っている.

4.2.2 概要

拡張開発の場合の分析・設計は, 一般的に以下のステップとなる.

- ソースコードから設計モデルを把握する
- 設計モデルを抽象化して分析モデルを把握する
- 開発スコープの要求を取り込んだ, 分析モデルの作成および設計モデルを作成する

新規開発の時は, ビジネス・オブジェクト・モデルとユースケース仕様から分析・設計を駆動していくので抽象度は高から低へ一方向であるのに対して, 拡張開発では現行のクラスの把握といった抽象度の低いところから抽象度を高くし, 要求を組み込んだあと, また抽象度を下げていく, という作業になる. いずれにしても, 作業対象ユースケースの分析・設計作業セットである図2の（A）グループは作成が必要である.

図2中の（B）グループは要求の内容によっては修正が入らない場合もあるが, 基本的には既存のものを利

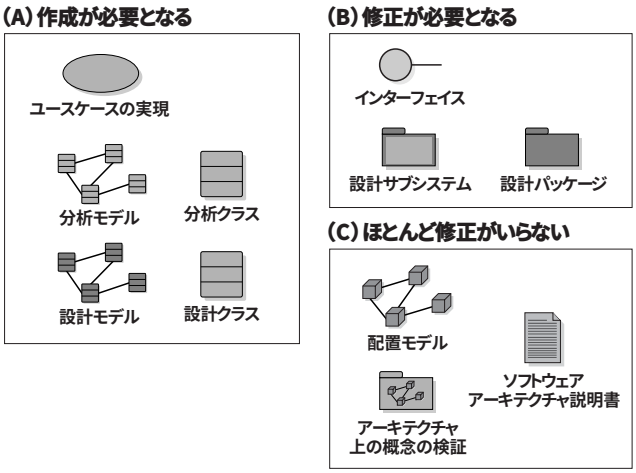

図2. 分析・設計の成果物セット－拡張開発編（RUP [2] より抜粋の後分類）

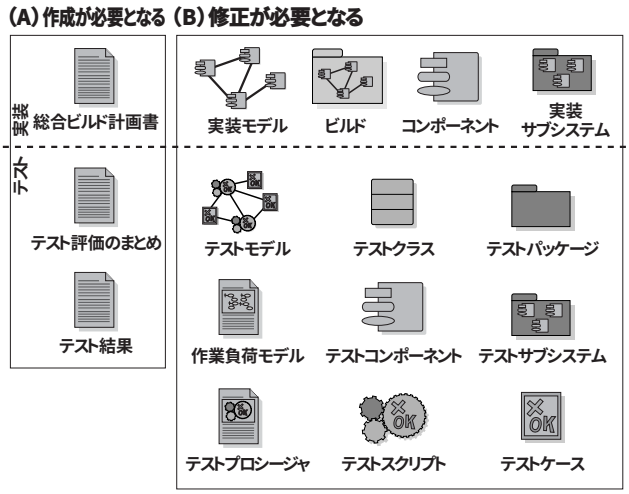

図3. 実装・テストの成果物セット－拡張開発編（RUP [2] より抜粋の後分類）

用できる. この点は（C）グループも同様であるが, （C）グループは要求の成果物セットの（C）グループに修正要求がある場合に連動して修正が入る.

4.2.3 カスタマイズ

先述の通り, 分析・設計作業はソースコードの把握から入るため, 変更内容をその場で「ソースコードのここを修正すればよい」と判別でき, システム全体のバランスや保守性・拡張性などを考慮しても最良の策であると判断できるのであれば, 分析モデルまで立ち返って修正する必要がない場合もある.

4.3 実装・テストワークフロー

4.3.1 目標

実装ワークフローにおいて, 今回は成果物を固まりでとらえることとし, たとえ新規に作成されるコンポーネントがあったとしてもコンポーネント（という成果物の集合）は既に存在するはずなので, 修正と定義した. テストについても同様である. すると**図3**のようにほとんどの実装・テストワークフローの成果物は修正となり, 当該プロジェクトのために新たに作成する成果物は, （A）グループのように作業を管理する類いのものに限られるようになる. これは言うなれば, 拡張開発においてこのように再利用可能な成果物の在り方を追求することの提言でもある.

4.3.2 概要

戦略的に開発効率を上げるには, テスト成果物の再利用を推進することが必要である. 図3中にある（B）のテストに関する成果物は再利用を前提としてある.

拡張開発においては1つのユースケースの中で修正部分と非修正部分がでてくる. ユースケースとして完全に機能することを確認することが望ましいが, 2.2

で述べたように, 修正しなかった部分のテストについては, 一般的にコストとのバランスによりどこまで実施するかが決定されることになる.

4.3.3　カスタマイズ

RUPでのテストクラス,テストパッケージ,テストコンポーネント,テストサブシステムは「テスト固有の機能を設計し, 実装する場合のみ使用します」とされている. 拡張開発でもこれは継承される.

5. 拡張開発の難易度を決定する要因

拡張開発の特徴として, 以下の点を事前に把握することで, プロジェクトの容易性を推し量ることができる.

(1) アーキテクチャの変更の有無

今回の拡張開発でアーキテクチャに変更があるか, ある場合はどの程度の規模の変更になるか. 非機能要件に変更がある場合は, アーキテクチャに影響を与えることがあるので注意したい.

(2) 既存システムのユースケース仕様書の有無

既存システムの振る舞いを表すユースケース仕様書が存在するか.

(3) テストケースの有無

(2)を検証するためのテストケース.

(4) テストクラス・スクリプトの有無

(3)を自動化するクラスやスクリプト. テスト自動化ツールに依存するものが多い.

(5) 開発環境の有無

開発を実施するための環境がすぐに入手できるか. 特にデータベースとその中身であるデータが大きく影響する. このデータはテストケースと密接な関係を持っていることが期待される.

(6) テスト環境の有無

即座にテストが実行できる環境が入手できるか. 開発の結果をすぐに載せることができ, 本番環境に影響を与えない環境であることが必要である.

(7) 新規ツールの採用

今まで使用していなかった開発支援ツールを導入するかどうか. 使用スキルの習得にかかるコストの費用対効果などが考慮点となる.

(8) オリジナルの要求の管理

拡張開発を実施する直前, 安定稼動している本番システムが「どんな要求を満たしているシステムなのか」を正しく管理している“ 要求リスト ”があるかどうか.

拡張システムとはまさに, この要求に対する“ 拡張 ”といえるからである.

6. 拡張開発におけるプロジェクト計画

この章では, アクティビティをWBS（Work Breakdown Structure）のような実施計画に落とす時に必要な, 拡張開発の特徴的な留意点を提言する.

拡張開発はシステム要求を詳細化していくと

- 機能の追加
- 機能の変更

に大きく分類できる. ここでの機能の追加というのはユースケースの追加と同じものと捉える. この場合, ユースケースが増加しないことを考えると, 機能の削除も変更の一種とみなす.

これから述べるのは, ユースケースを上記2つから

- 追加ユースケース（機能の追加を実現）
- 変更ユースケース（機能の変更を実現）

に分類した際に, それらを元に反復計画をどのように立てるかの戦略を考える際にヒントとなるものであるが, インクリメンタルな開発を行う時にも同じ観点で計画を立てるのに役立つであろう.

6.1　基礎となる考え方

拡張開発といえども, 反復を導入する動機は「リスクの早期解消」が主であろう.「変化への適応」や「学習効果」といった効果も享受できると考えられる. これを考慮すると, 通常のRUPの以下の戦略が有効である.

- リスクの高いユースケースを早期の反復で採用
- 同じリスクであれば, 単純に時間的な長さを考慮して追加ユースケースを先に着手

さらに別の観点を導入しよう.

RUPに定義されている反復の戦略は「広く浅く」と「狭く深く」の2つとされている.

「広く浅く」は, 広範囲なユースケースを元に広範なアーキテクチャを作成していき, 推敲フェーズまでは反復をあまり行わないタイプの戦略で, アーキテクチャが前例のないエリアを扱う時や, チーム要員のスキル不足の時に用いられる.

「狭く深く」は問題領域をスライスし, 1つのスライスの問題を徹底的に理解していくタイプの戦略で, 主要なリスクを堅実に克服したい時や, 締切りを厳守するために実装に早期に着手することがカギとなる時に用いる.

これを拡張開発の特徴と照らし合わせると以下の戦略を加えることができる.

- 更新はアーキテクチャに影響を与えることが稀なので全般で「狭く深く」を採用する
- 追加は通常のように方向付けと推敲フェーズで「広く浅く」を, 作成フェーズで「狭く深く」を採用する

いずれにしても, ユースケースの優先順位付けとしっかりした反復計画は不可欠である.

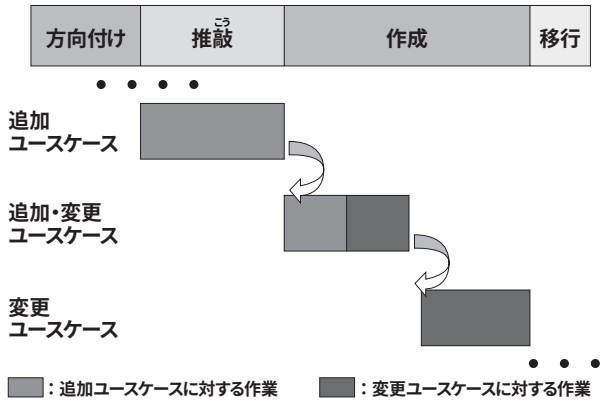

図4. 追加ユースケース中心の反復計画

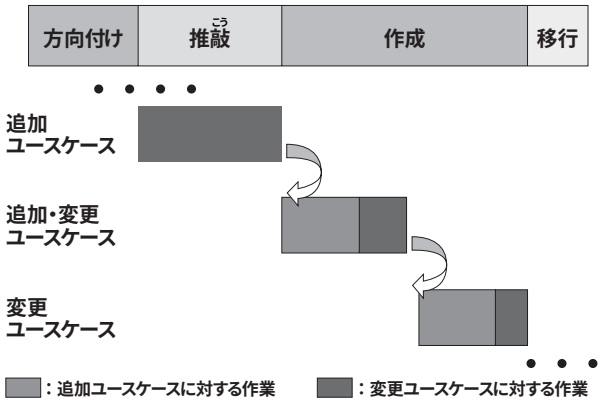

図5. 変更ユースケース中心の反復計画

6.2 ユースケースの優先順位付けの考察

ここでは以下の特性に合わせて, どの反復で実装をするかを特徴的な2つのケースにわけて考えてみる.

- 追加と変更のユースケースの優先度合い
- アーキテクチャへの変更の有無
- プロジェクトの抱える課題

これらの特性とこれから述べる戦略を理解し, 同じようなアプローチを取ることで, 様々なプロジェクトに応用することができると考える.

(1) ケース1

「追加ユースケースが多く, もしかしたらアーキテクチャに影響を及ぼすものがあるかもしれない. その拡張開発ではこの機能追加こそがプロジェクトの大きな目的であり, 重要なユースケースとなっている.」

この場合, 方向付けフェーズである程度変更ユースケースの要件を固められれば,「狭く深く」の戦略通りに作成フェーズで反復を繰り返す計画とする. 肝心な追加ユースケースについては**図4**に示す様に, 推敲フェーズから着手して早期に実現のめどを確保しリスクを下げる.

(2) ケース2

「変更ユースケースが多く, アーキテクチャに影響を及ぼすものはない可能性が高い. その拡張開発では変更要求を寄せ集めたプロジェクトである. 一方でスキルに不安があり, 短納期を実現することが重要である.」

この場合, 要員のスキルに不安を抱えつつ開発スピードを上げるという矛盾した課題を克服するために, 作業しやすいユースケースから着手して「学習効果」を得ることに重点を置く反復計画となる. オブジェクト指向分析設計作業や, RUP自体に慣れていない要員が多い場合などに有効である.

また, 機能追加は一般的に機能変更より要件が固まりづらいことから, 追加ユースケースを反復の中盤から後半に持っていく, という戦略も含んでいる. これを**図5**に示したが, この場合機能追加部分にリスクが含まれているとすると, リスク回避がプロジェクトの遅い時期になるので注意が必要だが, 拡張開発プロジェクトでは機能追加でも低リスクの場合が少なくないことを活かした戦略である.

6.3 プロジェクト運営上の考慮点

ここまでRUPにおける拡張開発のヒントをいくつか提示してきた. しかしながらこれらをうのみにしていればプロジェクトが成功するほど現在のe-ビジネス・プロジェクトは簡単ではない. ここまで述べてきたヒントを頭にいれて, 主に以下のようなことを, プロジェクトを通して継続的に観察し, 必要に応じて対処していくことが, 経験上健全なプロジェクト運営には不可欠であると考える.

- 要求を管理する
- 要員のスキルを管理する
- 進捗を管理する
- 成果物の完了基準を明確化し, 要員に浸透させる
- 反復の完了基準を明確にし, 評価を確実に実施する
- 品質を管理する
- 変更を管理する
- リスクを管理する
- プロジェクトチームの開発スピードを把握し, 向上させる工夫をする

7. おわりに

プロジェクトを成功させるためには, プロジェクトの特性や, プロジェクトリーダーやメンバーのスキル, お客様の理解や協力といった様々な要素から, プロジェクト毎に最適な計画を立てることが重要である. 今回はAMSでの拡張開発における計画のための判断基準を述べてきたが, 拡張開発ではこうでなくてはならないといった制約を与えるものではなく, あくまで1つの実践的なアプローチとして利用することで計画

をよりよいものにし，プロジェクトを成功に導くことが本論文のねらいである．

今後AMSというサービスが伸びていく傾向の中で，RUPというベストプラクティスを積極的に採用し，効果を出していくことは何よりもお客様の満足度に貢献するものと考える．

謝辞

本論文の作成にあたって，内容のレビューや議論に参加してくださったAMS技術の方に改めて感謝いたします．

参考文献

［1］鈴木 和博，次世代e-business時代に対応する開発メソドロジーの考察，ProVISION，Winter 2002，No.32，pp.32-41，2002

［2］IBM Corporation，Rational Unified Process 日本語版 Version 2001A.04.01，Rational Suite Enterprise製品に同梱，2002

［3］イヴァー・ヤコブソン，グラディ・ブーチ，ジェームズ・ランボー，UMLによる統一ソフトウエア開発プロセス，翔泳社，ISBN4-88135-863-7，2000

［4］Leslee Probasco，RUPにおける10の重要な要素：効果的な開発プロセスの重要な要素，Rational Softwareホワイトペーパー TP177，IBM Corporation，2000

［5］Philippe Kruchten，*What Is the Rational Unified Process?*，Rational Edge http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/feb03/WhatisRUP_TheRationalEdge_Feb2003.pdf，IBM Corporation，2003

［6］Liz Augustine，*Using the Rational Unified Process（RUP）Successfully for Small Development Projects*，Rational Edge http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/sep01/UsingtheRUPForSmallDevelopmentSep01.pdf，IBM Corporation，2001

［7］Philippe Kruchten，プロジェクトに1人で果敢に取り組んだあるプロジェクトリーダーの日記，Rational Edge http://www.atmarkit.co.jp/fjava/devs/redge03/redge03_1.html，2002

［8］Roger Oberg，Leslee Probasco，Maria Ericsson，ユースケースを使用した要求管理の適用，Rational Softwareホワイトペーパー TP505，IBM Corporation，2000

日本アイ・ビー・エム株式会社
ITアーキテクト

畑 秀明 Hideaki Hata

［プロフィール］

1993年，日本アイ・ビー・エム入社．AMSサービスのシステム・エンジニアとして，社内システムの開発・保守作業に従事．主にIBMのインテグレーテッド・テクノロジー・サービス関連システムを担当し，IBMビジネスの向上に貢献．2001年よりAMSサービス内の技術サポート・メンバーとして，ソフトウェア・エンジニアリングの向上に取り組み，ラショナル統一プロセス（RUP）の導入をリード．
現在ITアーキテクトとして開発プロセスの改善に取り組んでいる．

e26814@jp.ibm.com