Webアプリケーション開発における新しい開発スタイルの試み

公共・公益システム事業部
公共・公益システム第１開発部
ITエンジニア

小林　秀樹

Hideki Kobayashi

エンジニアリングシステム事業部
ＰＬＭソリューション部
ITエンジニア

八田　隆司

Takashi Hatta

鉄鋼システム事業部
新統合プロジェクト部
ITエンジニア

高埜　梨絵子

Rieko Takano

技術部
ITエンジニア

井関　知文

Tomofumi Iseki

開発者が異なる組織に所属し、地理的に分散した環境で協調してシステム開発を行う体制を「仮想チーム」と呼ぶ。エクサ社内システムをWebアプリケーションとして開発するにあたり、異なる組織に所属する若手技術者数名で仮想チームを構成し実施した。この開発は、オブジェクト指向をベースにした当社の方法論（エクサ方法論）に基づいて行った。仮想チームのメンバは、オブジェクト指向開発やWebアプリケーションの開発が未経験であったため、プロジェクトに参加しながら実作業を通して教育するメンタリングと呼ばれるスタイルで育成した。

本論文では、本プロジェクトの成果、仮想チームを運営するうえでの要点、エクサ方法論の評価、オブジェクト指向技術者育成の成果について述べる。

1．はじめに

開発者がそれぞれ所属する組織に分散したままで協調してシステム開発を行う体制を、「仮想チーム」と呼ぶ。その組織が互いに遠隔にある場合も少なくない。このような仮想チームを円滑に機能させることは、今日のように、小規模開発のプロジェクトが多数ある状況、あるいはオフショア開発を伴う状況では、プロジェクトの重要な成功要因となる。仮想チームを運営するに当たっては、従来のように、単に開発者のパフォーマンスを支援する仕組みを用意するだけでなく、孤立しかねない開発者の参加意識を高めるために、開発者間のコミュニケーションの質と量を十分に確保する仕組みが必要である。

われわれは、エクサの社内システム「アウトルックシステム」の開発に当たって、これまでにJavaやWebアプリケーションの開発経験のない若手５名を異なる組織から選んで仮想チームを構成し、開発を円滑に機能させる条件について調査した。

これに加えて、二つの試みを行った。一つはエクサ方法論をこの仮想チームに適用すること、二つめはこうした活動を通じてオブジェクト指向の技術者を育成することである。以下で、本プロジェクトの成果について報告する。

2．アウトルックシステムの概要

本プロジェクトの開発テーマである「アウトルックシステム」について概説する。本システムは、見いだされたビジネスチャンス（Opportunity）が、具体的なビジネスとして段階的に進展していく経過を追跡することで、営業成績の見通しを示し、営業員のアクションの立案を支援する。その主な機能は、①案件の確度ごとの売上、原価、粗利を集計し、目標値と比較する、②期間を指定して全案件の進捗を一覧するなどである。これらの情報は、営業員、部署、部門のメッシュで閲覧できる。

3．エクサ方法論

エクサ方法論は、オブジェクト指向技術による情報システムの開発プロジェクトの実践をとおして得られたさまざまな知見や経験を体系化した開発方法論である。これは、お客様の要求（requirements）を適切にとらえ、拡張や変更を容易にし、継続的に顧客満足度の高い情報システムを実現することを目的とする。これまでの方法論とは異なり、方法論自体が進化していくことを認める。これについて簡単に紹介する。

3.1．ソフトウエアプロセス

エクサ方法論のソフトウエアプロセスは、RUP（Rational Unified Process）[1]に準拠する。すなわち、活動を「方向づけ」、「推敲」、「構築」、「移行」と呼ぶ４つのフェーズに分類し、それぞれに主マイルストーンを設定する。主マイルストーンはフェーズ完了の基準であり、これを満足するまではそのフェーズを再度設定し、これを繰り返す。

「方向づけ」では開発システムのアーキテクチャを検討し、「推敲」ではアーキテクチャを完成する。「構築」ではアプリケーションを構築し、「移行」ではアプリケーションを稼働環境に展開する。

各フェーズ内では「要求記述」、「分析」、「設計」、「実装」、「検査」と呼ぶ作業（ワークフロー）を繰り返す。ウォータフォールプロセスと異なり、この順序は固定的でない。各作業の完了基準として副マイルストーンを設定する。品質、コスト、納期を確保する目的に整合する限りは、作業の逆行も認める。したがって、このプロセスには二重の繰り返しがある。こうした繰り返しによって、要求のぶれを早い段階で収束すること、隠れた要求を取り出すこと、より有効な要求セットを導くことを促し、顧客満足の向上とコストの低減を図る。

3.2．概念モデリングの重視

本方法論の特徴は、概念モデリングおよび要求記述を重視する点にある。

3.2.1．基本定義と基本課題

エクサ方法論では、お客様の要求をトップダウンに導き出すためにSSM（Soft Systems Methodology）[2]の手法を用いる。

情報システムは人間活動システムを支援するものであるが、人間活動システムはソフトな問題であり、問題の解は一意に定まらず、ステークホルダ間の合意（accommodation）が必須である。SSMは、このような合意に至るプロセスと、その成果としての情報システムの基本定義、その基本

定義を充足する基本課題の導出に関する知見を提供する。

3.2.2. 概念モデリング

ビジネスの対象世界（Universe of Discourse）の本質的な概念を整理し、型モデルとして記述する。これは、開発者およびステークホルダ間で対象世界に対する深い理解を共有する土台となる。型モデルは、問題領域の概念構造を型の構造として表現する。ここでいう「型」とは、概念をインタフェース、属性、操作をもつ抽象的な構造体として表現するものである。この型を、設計作業を経てクラスあるいはクラス群として実装する。型の構造を記述するためにUMLのクラス図の記法を用いる。▪

3.2.3. 変更容易性

ビジネスは、マーケットの動向や技術進歩に伴って変化し続ける。企業活動を支援する情報システムも、ビジネスの変化に応じて俊敏に変更できなければならない。われわれは、良い情報システムとは、要求が満足されているだけでなく、変更が容易であるべきであると考える。変更が容易なシステムとは、システムを全面的に入れ替えるのではなく、少しずつ継続的に変更していくことを許容する強さを持ったシステムである。このようなシステムは、本質的な概念モデルを追求することで達成されると考える。▪

3.2.4. 要求記述

(1)　ユースケース記述

要求記述に、ユースケース記述[3]を用いる。ユースケースは、アクタとシステムが協調して業務活動を達成する様子を形式的に記述することで、お客様の機能要求を明らかにするものである。

情報システムで支援しようとする業務活動の単位を取り上げて命名（ユースケース名）し、その目的、アクタと呼ぶユーザの役割（ロール）、システムとの対話を自然言語で記述する。業務活動の目的を明示することで、ビジネス目標に沿った機能要求であることを担保する。

ユースケース記述は、お客様が読んで理解できるので、お客様と開発側の合意を引き出す有効な手段となる。
ユースケース図は、情報システムの全体像を把握するために使用する。

(2)　シナリオ（業務フロー物語）

ユースケース記述は、アクタとシステムのやりとりを抽象的に記述するため、どのような場面を想定しているのか理解しづらいことがある。そこで、具体的なやりとりを別途記述することで、機能要求に対する理解を促し、誤解を避けるようにする。このための個別事例の具体的な記述をシナリオと呼ぶ。

シナリオは、いつ、どこで、誰が、何を、何のために、どうしたかを想定してできるだけ例示的に記述する。これを１つのユースケースについて１つ以上記述する。これは受け入れテスト時のテストケースにもなる。

(3)　活動図

ユースケース記述によって個々の機能要求が記述できるが、その実行順序の関係は記述できない。そこでUMLの活動図を使って、業務のまとまりごとに業務活動の順序（ワークフロー図）を書く。

ワークフロー図には運用に関わる業務活動も書かれるので、トップダウンアプローチだけでは導けない現実的な要求を発見できる。これをボトムアップによるユースケースとして取り上げる。▪

3.3. アーキテクチャ

開発するシステムの複雑さを軽減するためには「関心を分離する」ことが効果的である[4]。ここでいう関心とは、開発者がシステムを開発するために考慮しなければならない事柄、たとえば、DB（Data Base）へのアクセス、UI（User Interface）、業務ロジックである。本方法論では、関心の分離を達成する方法の１つとして、レイヤ構造を用いる。

レイヤ（層）構造とは、アプリケーションを抽象度の異なるサブタスクのグループ（レイヤ）に分割し、抽象度の順にレイヤを積み重ねることで、隣接するレイヤ間のインタフェースだけに依存すると同時に、インタフェースを最小化するためのアーキテクチャである[5]。

JavaのVM（Virtual Machine）をレイヤ構造の例としてとりあげる。Javaでは、VM層を設けたことで、アプリケーションの開発者はOSなどのプラットフォームを気にせずに開発でき、そのアプリケーションはプラットフォームを変更しても動作が保証される。VMはプラットフォームごとにメーカが用意するのが一般的である。

本方法論では、プレゼンテーション層、アプリケーショ

ン層、ドメイン層、永続層という４層で開発することを推奨する（図１）。ただし、永続層の実装については、さまざまな実装方法があるので、プロジェクトごとの個別判断とする。

(1)　ドメイン層

対象世界の本質的な概念と概念操作のロジックをクラスとして実装し、ドメイン層に集中させる。これを複数のユースケースに共有させることで、プログラムコードの重複を減らし、変更容易性や開発生産性を向上させる。

(2)　プレゼンテーション層

プレゼンテーション層は、UIのクラス群で構成される。

(3)　アプリケーション層

ドメイン層とプレゼンテーション層の間にアプリケーション層と呼ぶ中間層を導入し、UIとアプリケーションロジックを分離する。特に、ドメイン層のオブジェクトをアプリケーション層にキャッシュして使う方法をアプリケーションファサードと呼び、これを推奨する[6)]。

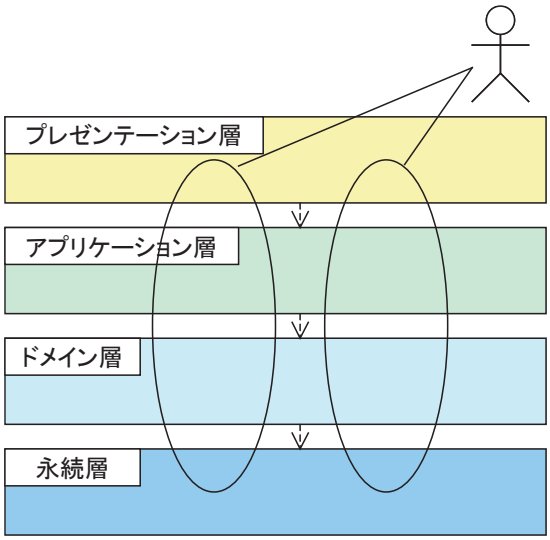

図１　レイヤ構造

4．プロジェクトの概要

4.1．プロジェクトの運営

本プロジェクトの体制は図２のとおりである。開発者は５名であり、彼らにはUI設計者、要求管理者、アーキテクト、要求記述者、DB管理者、サーバ管理者、テスト管理者、構成管理者などの役割（ロール）を割り当て、責任分担を明確化した。仮想チームにおいては、各人の作業が逐一見えるわけではないので、作業の取りこぼしが発生したり、誰かがやるだろうという依頼心が芽生えたりしがちだと考えたからである。

なお､開発者以外の役割の意味と責任は次のとおりである。

①アドバイザ：利用者の立場から、本来あるべきシステムの形を提示する。１名。

②メンタ：プロジェクトに支援的に参画し、エクサ方法論に基づくプロジェクト運営の支援、技術教育、実装指針提示、アーキテクチャ構築、難解な障害原因の追求等を行う。３名。

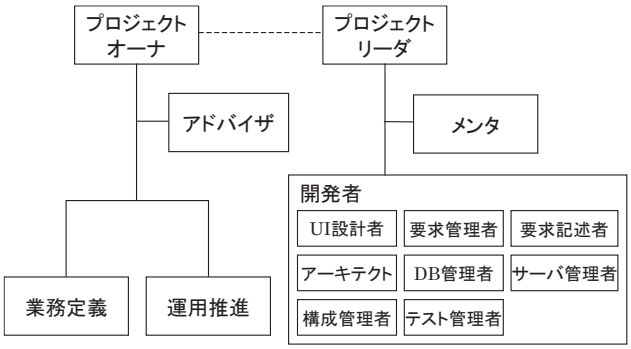

図２　組織体制図

4.1.1．プロジェクトのスケジュール

(1)　方向づけ（2002年12月上旬～2003年2月中旬）

要求の範囲を見極め、概念モデルとユースケース記述を行った。アプリケーションアーキテクチャを検討し、Webアプリケーションとして実装することにした。Webアプリケーションのフレームワークは、既存のものを基に画面遷移とデータの永続化部分を改良して用いることとした。

(2)　推敲（2003年2月下旬～6月上旬）

Webフレームワークの改良を行った。当初５月上旬までの完成を狙ったが、詳細な実装方針の決定と実装作業そのものが遅れたために、マイルストーンのクリアは６月上旬にまでずれ込んだ。構築フェーズの第１反復に対しては、永続化部分をスタブにしたフレームワークをリリースした。

(3)　構築

●　教育（2003年1月上旬～3月下旬）

Webアプリケーションとして実装することを決定した段

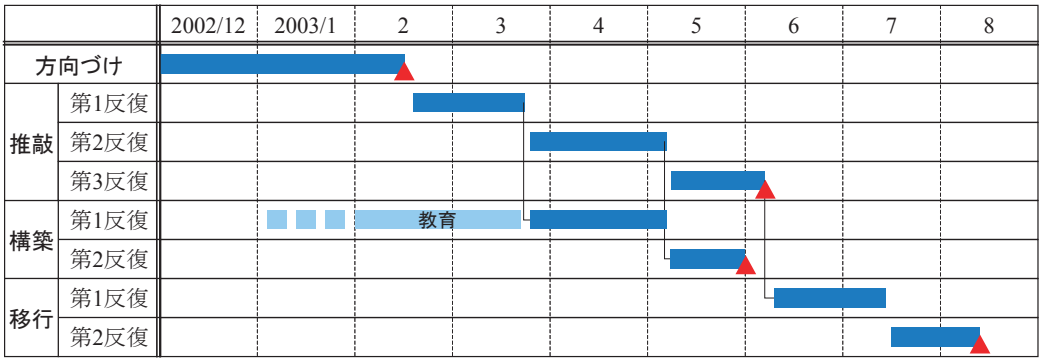

図３　プロジェクトスケジュール

階で、担当に決まった開発者の多くにはWebアプリケーションの開発経験がなかったので、スキル向上ための教育を行うこととした。

● 第１反復（2003年3月下旬～5月上旬）

スタブ版フレームワークを用いてドメイン層から上の層を開発した。アプリケーションの早期リリースによって、利用者からのフィードバックを得た。

● 第２反復（2003年5月上旬～6月上旬）

第１反復時のリリースで得られた変更要求に対応し、さらに、その時点で半完成の永続機能を持つフレームワークと結合した。

（4） 移行（2003年6月上旬～8月中旬）

完成版のフレームワークと第２反復後のドメイン層以上を結合した。これをさらに多くのユーザに使ってもらい、その中で発生した細かな要望に対応し、数度に渡ってマイナリリースを繰り返した。

4.2. 分散開発

分散開発では、一般に次のような問題が発生しやすい[7]。①ソースのバージョンが混乱する、②進捗状況の把握、問題解決に時間がかかる、③要件・仕様の変更が周知されない。

これらの問題に対応するために、成果物の共有、コミュニケーションの確保を図った。▪

4.2.1. 成果物の共有

成果物を個人ごとに保有することは、知識の共有が妨げられる点であってはならないが、共有フォルダを用いても、複数のメンバで開発を行うと、開発者間で同一ファイルの更新が衝突したり、相互更新によってバージョンの逆戻りを起こしたりするなどの問題が生じる。構成管理ツールを使用することで、これらの問題を回避するだけでなく、ソースのバージョン管理や変更管理、マージ、リリース管理が容易になった。構成管理ツールは、バグ対応やテスト作業における変更箇所の確認の支援機能も利用でき、有用であった。

・4.2.2. コミュニケーションの確保

従来型の開発ではメンバが近くにいて、容易に集まったり、話したりすることができた。分散開発ではそれができないため、開発メンバのコミュニケーションの確保が重要となる。他の開発メンバが現在何を変更しているかを知ったり、仕様や技術事項を質問／回答したりするためにメーリングリストを開設した。これによって、リアルタイムに近い情報交換が可能となり、一対一の関係にとどまらず、離れていても一対多のコミュニケーションが可能となった。

メーリングリストだけでは仮想チーム内のコミュニケーションを十分に確保することはできない。開発者が実際に顔をつきあわせて会議を行う必要がある。これをオフラインミーティングと呼ぶ。オフラインミーティングは、豊富な情報を伝達でき、即座にメンバからのフィードバックが得られるだけでなく、メンバが開発に参加しているという実感をもち、緊張感を保ち続けるという重要な役割を持つ。さまざまなテーマを掲げて、ほぼ定期的に平均週１回の頻度で開催した。▪

4.3. 教育

メンタは、方向づけフェーズで記述された概念モデルと要求内容から、アーキテクチャの実装を構想したうえで、開発者向けの教育を計画した。

その目標を「以降の開発作業においてメンタの協力がなくても、エクサ方法論に基づいた開発ができるチームとすること」と設定した。これを実現するためには、開発者は単にWebアプリケーションを開発できる知識を習得するだけでは不十分であり、図２で述べた開発者のロールごとに必要な価値観やマナーを習得しなければならない。

Webアプリケーション開発技術の教育過程においては、教材として、アウトルックシステムの「案件の登録および修正を行う」というユースケースの実装を取り上げた。こうすることで、システムの内容とそれをどのように開発するかの知識が一度に得られる。さらに、チームが協力して１つの例題を実装することを通して、構成管理ツールの使い方も学習させた。

開発者間でルールを設けずにコーディングを行うと、成果物の内容を理解するのに時間がかかる。そこで、内容の共有を促進するためにコーディングルールを定め、教育した。

教育期間終了後も、プロジェクトの作業を行うなかで問題点が発生した場合には、メンタが介入して指導する必要がある。介入もマンツーマンで行うことが望ましい。こうして、実践を通して主体的に参加する状況的な学習環境[8)]を構築した。▪

4.4. 方向づけ

方向づけフェーズでは、要求を理解するために粗いユースケースを書いた。実際に記述したユースケース記述を図４に示す。このユースケース「案件の新規登録を行う」は、案件を登録することで、その後の状態変化を追跡できるようにするためのものである。基本系列では、案件を登録するためにアクタとシステムが対話する様子を書いている。「コードを入力する」とか「ボタンを押す」といった実装を記述しない[9)]ように努力した。

ドメイン知識から、概念レベルの型図を作成した（図５）。このモデルでは、「案件」を「計画値」に展開し、それらを「契約」の進行状態と対応づける一方、「契約」に基づいた「Work」単位で原価を把握することが書かれている。それ以外の型は、それぞれの所有関係を表しているにすぎない。

ユースケース名	案件の新規登録を行う
アクタ	営業
目的	アクタは，案件の状態変化を追跡したい
事前条件	案件が登録されていない
基本系列	①アクタがこのユースケースを起動する ②システムは案件の入力を要求する ③アクタは各項目を入力する ④システムは必須項目が入力されたことを確認し，OpportunityIDを確定し，案件を登録する。
代替系列	基本系列④において，必須項目の登録漏れがあれば登録を拒否する
事後条件	案件が登録されている
備考	①基本系列④の必須項目は以下の◎とする。 ◎OpportunityID，◎プロジェクト開始日，◎確度， ◎案件名，XXXXXXX，XXXXXXX ②OpportunityIDは手入力とする。 ③OpportunityIDは従来のLegalContractNOと等しい
シナリオ	営業員の児玉は2003年4月8日に，札幌製菓の生産管理システムむけの提案書を書いて，案件を登録した。売り上げ予定は10億円であった。

図４　ユースケース記述の一例

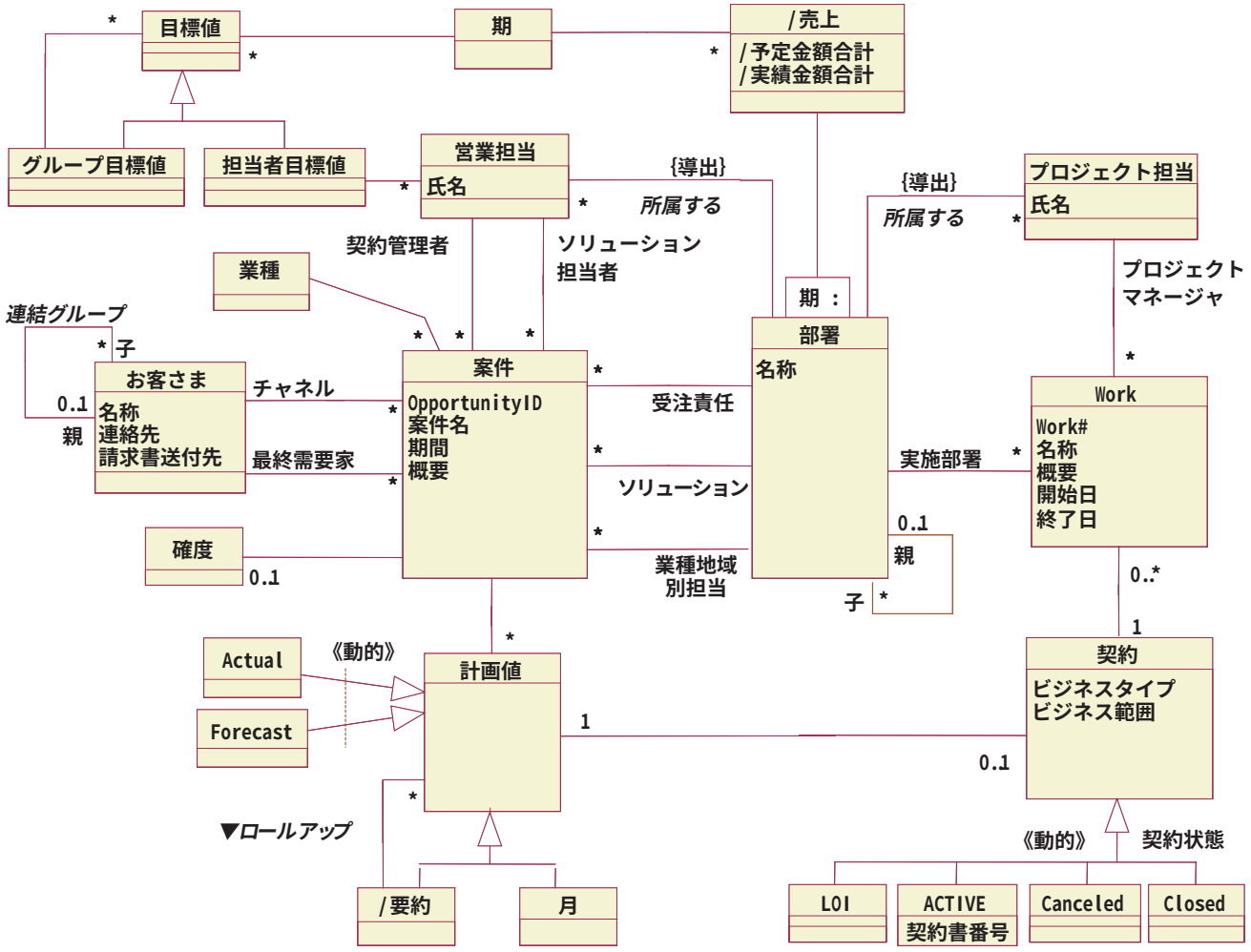

図５　概念レベルのクラス図

4.5. 推敲

推敲フェーズでは、概念レベルのモデルを実装モデルに変換し、ドメイン層とのインタフェースを決めて、Webフレームワークの拡張を行った。

概念レベルのモデルをドメイン層の実装レベルのモデルに変換するにあたって、まず、ユースケースを実現するために必要なデータとメソッドを割り当てた。次に、Java言語では実装できない動的分類をフラグで実現することにし、サブクラスをなくした。いくつかの属性型も、それが属する型の属性（データ項目）にした。さらに、型をRDB（Relational Data Base）のテーブルで表現し、関連をキーの対応関係で実装するために、主キーと外部キーを明らかにした。変換結果の実装モデルは図６のとおりである。

4.6. 構築および移行

構築フェーズでは、方向づけで記述した粗いユースケース記述を詳細化し、これを基に機能設計を行い、実装した。

移行フェーズでは、実装されたアプリケーションを稼働環境に展開した。その際、細かい要求変更に応えた。

4.7. 開発環境

本プロジェクトで採用した開発ツールは、統合開発環境（IDE）としてオープンソースのEclipse、テストツールとしてJUnitおよびHttpUnitであった。他のIDEのなかには動作にもたつきを感じるものもあるが、Eclipseは、ストレスを感じることなく使用できた。また、本プロジェクトで使用したさまざまなツールとプラグインを使用して連

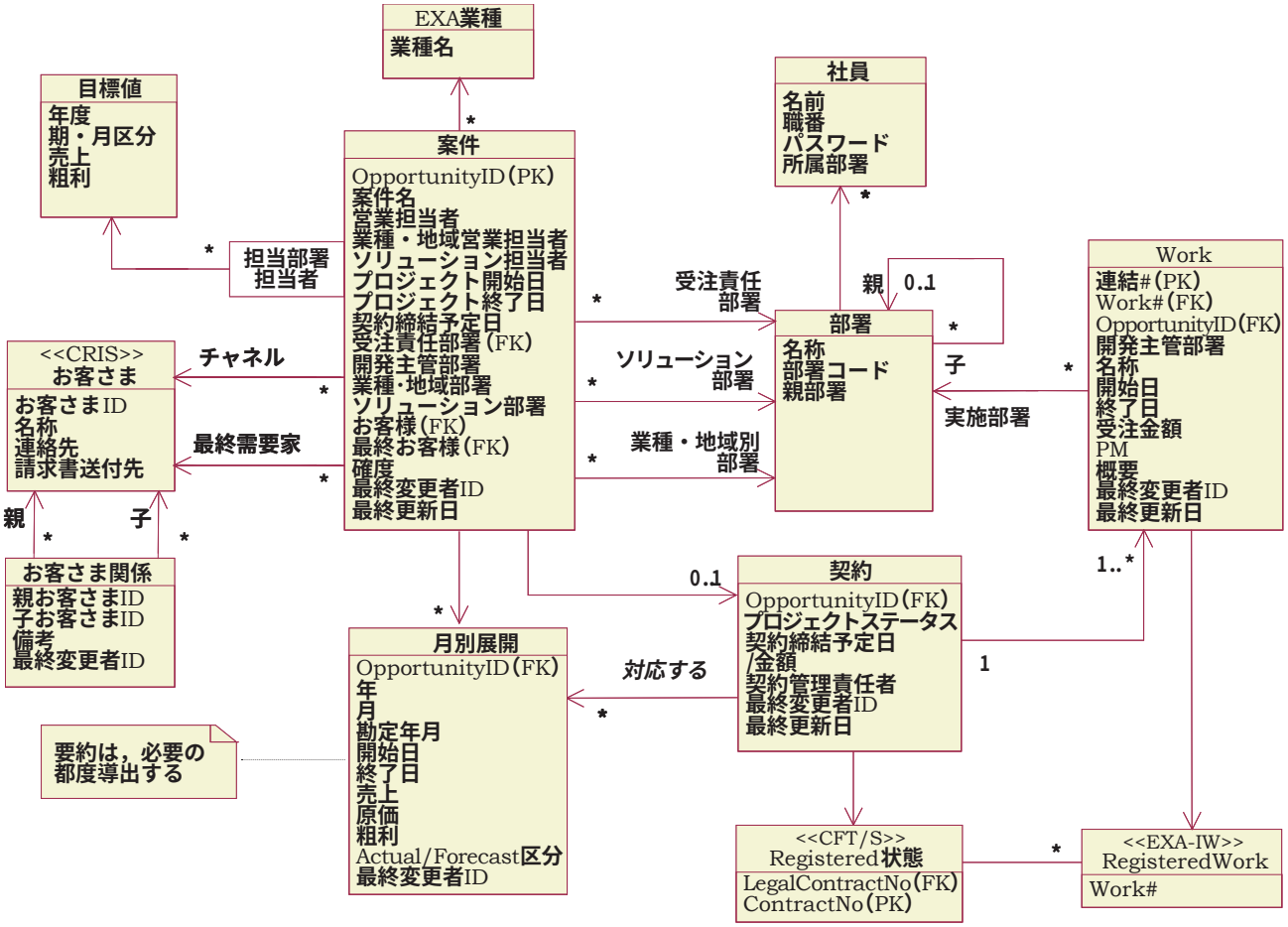

図６　実装レベルのクラス図

携することができたため、1つの環境上で統一されたUIのもとに開発作業が可能となり、開発生産性が向上した。

構成管理ツールは、Microsoft社のVSS（Visual Source Safe）を使った。これは、仮想チームによるプロジェクト運営では重要なツールなので、特に説明を加える。

4.7.1. 構成管理ツール

構成管理とはプロジェクト内における変更の要因や結果を管理するものである。バージョン管理は、ソフトウェアを構成する部品（モジュール、仕様書、開発手順、操作マニュアル、テスト手順、テストデータなど）の変更履歴を管理するもの[11]であり、履歴のみを管理する。この種のツールとしては、VSSの他にCVSが有名である。

VSSには、複数のユーザが同時に同一ファイルを編集できないようにロックする機構がある。CVSではロック機構がないため、並行性は上がるものの、複数のユーザが同一ファイルを同時に編集できてしまうので、衝突（同一個所の同時編集）が発生し、プロジェクト運営の安全性が低下する。さらに、マージ作業などの管理が煩わしく、結果的に生産性が低下するおそれがある。

VSSのもう一つの利点は、Eclipse用のプラグインを利用して、Eclipseの環境上で「チェックイン」、「チェックアウト」、「ソース差分のGUI表示」、「履歴の表示」、「過去のバージョンとの置換」が容易に行えることである。

仮想チームによる開発で最も配慮すべき点は、分散環境で資料や成果物を共有し、更新しながら、それらを保護することである[7]。分散環境で成果物を共有する方法として、Webでの資料の公開も考えられるが、変更履歴や変更者の記録を同時に表示することは面倒である。▪

4.8. 稼働環境

運用コストを最小化するために、サーバのOSにWindows

2000を、Web Application ServerおよびHttp ServerとしてTomcatを、RDBMS（Relational Data Base Management System）としてFirebirdを採用した。Tomcat、Firebirdともにオープンソースのプロダクトである。

5．評価および考察

本プロジェクトにおける方法論、仮想チームの運営、教育効果の面から評価し、考察する。▪

5.1．エクサ方法論の評価

本プロジェクトを通して、ソフトウエアプロセスに関する知見が得られたので報告する。反復型プロセスでは、要求の緊急性や重要度に応じてどの回の反復で行うかを統制するが、その反面、変更要求が次から次へと噴出し、打ち切りに苦労することがある。本プロジェクトにおいてもその例にもれない。この点について考えてみる。

（1） 誰から要求を引き出すか

変更要求が噴出した原因は、当初、要求抽出をエンドユーザからではなくアドバイザから引き出してしまったことにある。

アウトルックシステムは、新しい営業管理の方針に基づくものであり、エンドユーザが要求を持っていないので、アドバイザから要求抽出することは誤りではなかった。しかし、プロジェクトの進め方としては、エンドユーザ（営業員）も含めて、新しい管理方針に関する合意とシステムに関する議論を深めておく必要があった。結果的として、構築フェーズの第２反復後にエンドユーザからの要求が噴出することとなった。

（2） 変更要求を肯定的に受け止める

この事象を避けるためには、ウォータフォール型プロセスに沿って、要求記述時に厳密に要求を確定しておけばよかったのだろうか。そうではない。実際にはそれが不可能だからこそ、今日ではウォータフォール型プロセスはほとんど採用されない。採用したつもりでも、結果的に逆戻りを認めざるを得ない。

エンドユーザの要求が出てくるのは、事前の議論においてではない。事実、本プロジェクトにおいても、エンドユーザの代表者と何度か議論している。にもかかわらず、エンドユーザの具体的な要求が出てきたのは実際に操作してからである。それは当然のことであり、むしろ、反復型プロセスの意義は、こうしたフィードバックを肯定的に受け取ることにある。肯定的に受け取るには、情報システムの基礎にあるモデルが安定していることが必要である。このためにこそ本質な概念モデルを追求するのである。

（3） リスクへの対策

本プロジェクトの問題点は、変更要求が噴出したことではなく、その原因である新しいシステムに対する要求抽出をステークホルダの一部だけから行うことによるリスクを甘くみたことである。あらかじめ構築フェーズでの反復を多めに計画しておくなどの対策を講じておくべきであった。

5.2．仮想チームの運営

構成管理ツールの効用について考察する。構成管理ツールを利用したことで、ファイルの履歴管理、ファイルの比較を容易に行うことができた。

構成管理作業の間違い（たとえば、チェックインやベースラインの定義し忘れ）が発生すると、仮想チームの作業は混乱する。そのリスクを軽減するために、本格的開発が始まる前の教育期間中に構成管理ツールの使い方を指導した。

開発成果物に「Release1.0RC2」などのラベル付けができることで、開発途中であっても、バージョンを指定してソースを取得できることから、反復ごとの段階的リリースを行う際に有効であった。中でも、誰が何のために変更したかを記録できるので、コミュニケーションロスを防ぐことができた。▪

5.3．教育の評価

（1） マンツーマンによる指導

本プロジェクトでは一定の教育期間を与えられ、開発メンバの知識の底上げができた。マンツーマンに近い環境と、質問に対してリアルタイムに答えられる環境が効果的であった。このほかに、メンタと開発者の年齢が近かったのも、気軽に質問しやすい雰囲気を出すのに貢献した。

ロールを割り当てたことにより、開発者一人一人が習得しなければならない知識量を制限できた。しかしその結果、それぞれの知識が属人化され、担当者がいないと不具合に対処できないという問題も生じた。その対策としては、二人一組で一つの役割を割り当てる方法も考えられる。そうしておけば、知識を共有できるだけでなく、他のメンバに

理解しやすい文書やプログラムが得られ、不具合の対処を一人に集中させないですむ。

（2） 開発プラクティス

教育期間からコーディングルールを決めていたこともあり、他の開発者の書いたソースを理解しやすくなった。座学や参考書だけでは習得しにくいオブジェクト指向技術も、UMLの各ダイアグラムとそれらを実装したプログラムソースとの関係を確認することで容易に理解できた。開発メンバは、教育の後、すぐに新たな業務で活用する機会に恵まれたこともあり、オブジェクト指向技術の理解を深め、UMLのモデルを書く力をつけた。

6．おわりに

本プロジェクトにおける三つの試みは成功したと言えよう。経験の浅い技術者からなる仮想チームが、エクサ方法論によって目的のシステムを開発でき、技術者も育成できた。

この成功要因は、上で述べたように、方法論や教育体制、アプリケーションフレームワークなどであろうが、もう一つ、隠れた成功要因があると思う。それは、メンバの年齢が若かったことである。古い開発手法やソフトウエアプロセスにとらわれることなく、新しい方法論を素直に受け入れ、多くの知識を吸収できた。

開発メンバは現在、吸収したオブジェクト指向技術やWebアプリケーション開発の経験を評価され、おのおの新たな次のWebアプリケーションの開発に携わっている。本プロジェクトでの経験を基に、さらなる方法論の普及と技術者の育成を図っていきたい。

参考文献

1） Kruchten, P. 藤井拓監訳、「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

2） Checkland, P. et al 妹尾堅一郎監訳、「ソフトシステムズ方法論」、有斐閣、1994

3） Jacobson, I. et al 西岡利博ほか訳、「オブジェクト指向ソフトウェア工学 use-caseによるアプローチ」、トッパン、1995

4） Dijkstra, E.W. "A Discipline of Programming"、 Prentice-Hall、 Englewood、 N.J、 1976、 135

5） Buschmann, F. et al. 金澤典子ほか訳、「ソフトウエアアーキテクチャ―ソフトウエア開発のためのパターン体系」、トッパン、1999

6） Fowler, M. 児玉公信ほか訳、「アナリシスパターン」、アジソンウエスレイ・ジャパン、1998

7） Karolak, D. W. "Global Software Development: Managing Virtual Teams and Environments"、 IEEE CS、 1998

8） Lave, J. 佐伯胖監訳、「状況に埋め込まれた学習」、産業図書、1998

9） Constantine, L. et al "Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design"、Addison-Wesley、1999

10） Jakarta Project. http://jakarta.apache.org

11） 徳田弘昭 「ソフトウェア構成管理」、ソフト・リサーチ・センター、1999

12） Loucopoulos, P. 富野壽訳、「要求定義工学入門」、構造計画研究所、1997

＜問い合わせ先＞

技術部

開発技術チーム

Tel. 044-540-2141 井関 知文

E-mail:tomofumi-iseki@exa-corp.co.jp