

15-A-6

「フィーチャー」の概念を取り入れたモデルベース開発¹

1. 適用した技術や手法の概要

ソフトウェア設計とは、要求を、各開発フェーズにおいて適した粒度で詳細化していく作業である。本事例では、アジャイル設計手法の考え方の一つである「フィーチャー」（小さな機能のかたまり）の概念を取り入れ、ソフトウェア要求分析からソフトウェア詳細設計までシームレスに繋ぐモデルベース開発の仕組みを紹介する。

2. 適用した技術や手法の導入に踏み切った理由や経緯

ソフトウェア開発現場において、若年層や詳細設計書段階から開発に参加したエンジニアは、作ろうとしているプログラムの上位設計書として何を見れば良いのか、どの部分を仕様として捉えれば良いのかが分からず苦勞していることが多い。これは、システムへの理解不足や設計手法の習得不足が原因ではあるものの、そもそも、当該プログラムに関する機能方式設計や詳細設計と対応づけるものがないことや、あったとしても、必要な機能方式設計の項目抜けなどを容易に検出できるものではないことが原因であることも多い。

本事例では、上記の設計段階の課題を解決できるよう、各設計フェーズの設計粒度を具体化し、それぞれを関係付ける方法を明確にした。具体的には、機能方式設計に、アジャイル設計手法のひとつである FDD（Feature Driven Development：ユーザー機能駆動開発）で定義されている「フィーチャー」という単位を導入した。これに基づいて、要求仕様間の依存度を可能な限り小さくなるように表現すれば、仕様追加・変更や設計変更に対応できるようになる。

さらに、特定分野でしか成功していない MDA（Model Driven Architecture：モデル駆動アーキテクチャ）の考え方を、一般的なソフトウェア開発に対して可能な限り取り入れた。取り入れた概念は以下の 3 点である。

- ① 各フェーズ内の設計のモデリング作業
- ② フェーズ間の繋ぎをモデル間の連携と考えることで、上流から下流までの設計をシームレスにつなげる
- ③ 設計（機能）とアーキテクチャを独立させる

昨今、多くなっている派生開発においては、ソフトウェアアーキテクチャ設計（後述の基盤方式設計）やクラス設計を経験することが無いため、その方法を知るエンジニアが育たず、

¹ 事例提供: 三菱スペース・ソフトウェア株式会社 関西事業部 藤原 啓一 氏

新規開発に問題が発生する場合がある。このような問題に対しても、本事例は具体的手順を示しているので、設計方法の擬似教育として有効利用することができると思う。

<フィーチャー概念の定義>

フィーチャーは『価値のある小さな機能のかたまり』であるサービスである。フィーチャーが、独立性の高いアプリケーション機能をプログラムとして実現する単位となる。

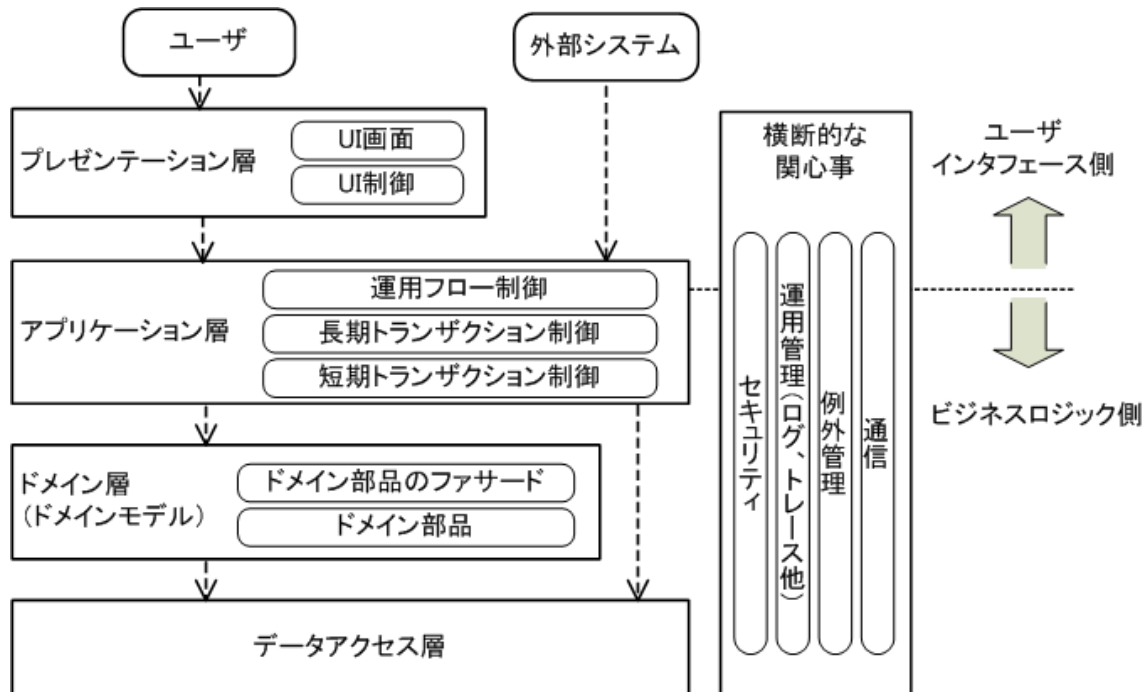


図 15-A-6-1 階層アーキテクチャ

図 15-A-6-1 のような階層アーキテクチャを想定すると、フィーチャーは、アプリケーション層の責務（≒そのクラスの主メソッド）を表し、ドメイン層のサービスを使って、アプリケーション固有の振る舞い（アプリケーション毎に異なる処理）を実装したものである。

概念形式的には、フィーチャーは、

〔目的語（オブジェクト）〕〔結果〕〔動詞（アクション）〕

という形式で表現される（ただし、〔結果〕は省略可能）。

<フィーチャーの例>

- ・ 販売員の成果を査定する。
- ・ スケジュールされた自動車整備を実施する。
- ・ カード所有者のクレジットカード取引を認証する。
- ・ 銀行預金口座の残高を取得する。

3. 活動の状況と定着のための事前準備や工夫

開発プロセスについては、「4. 適用した技術や手法の効果測定の方法とその結果」にその具体的な内容を記載し、ここでは、本活動を定着させるための工夫等について以下に記載する。

(1) 本活動の状況

- ・ 初期の開発プロセスの策定に2年をかけた。業務系に適用しつつ、具体化手順を作成。社内教育資料（パワーポイント）、設計ガイドラインを作り、ホームページにて公開した。
- ・ 1年間の教育を社員に限らず、協力会社の人たちにも実施した。
- ・ パイロットプロジェクト（通信組込み系）への適用に1年を費やした。
- ・ 新規のプロジェクトに適用中（2年を経過）である。

(2) 活動を開始するための工夫

教育には1年をかけた。教育は、社員に限らず、協力会社の人たちにも実施した。この開発手法は、全開発プロセスにわたり、今までとは違うやり方・設計フォーマットを採用するので、「現場がやってみてもいい」という気持ちになるまでの期間を考慮した。

また、部門を預かる責任者が強力に推進し、最後は、トップダウン指示としてスタートできた。

(3) 現場に合わせたカスタマイズやテーラリング等

「教育→実プロジェクトへの適用→課題の修正→実プロジェクトへの適用…」を繰り返している。初期の設計ガイドラインは開発者が1人で作成したが、設計フォーマットの改善や新たなガイドライン（USDM²作成ガイド等）は、現場の実施者が作成した。また、この作業はトップダウンで進めた。

(4) 使いこなせる人材の育成

Off-JT³、OJT⁴を織り交ぜながら育成している。今では、新規プロジェクトはすべてこの開発プロセスで実施しているので、新人でも自然とOJTの中で教育されることになる。

4. 適用した技術や手法の効果測定の方法とその結果

4.1. ソフトウェア設計手順の概要

本設計手法の全体像として、ソフトウェア要求分析からソフトウェア詳細設計までの流れ

² Universal Specification Describing Manner

³ Off-the Job Training

⁴ On the Job Training

を図 15-A-6-2 に示す。図中、黄色の塗りつぶし部分は、フィーチャー関連の仕様書や設計書であり、UML で定められている以外の記法も設計手法の中に導入している。ユーザーの要求は、業務フローとユースケース／ユースケースシナリオで表現される。これらから、コンピュータを使った場合の仕様表現として、機能仕様と画面設計書を作成する。そして、それらを受けたコンピュータ内部表現として、機能方式設計書、基盤方式設計書を作成する。図 15-A-6-2 ではソフトウェア方式設計の箇所であり機能方式設計を行うことである。この時、機能は、前述のフィーチャーを単位として表現する。そして、詳細設計はフィーチャー毎に、さらに設計を進め、フィーチャー設計書（クラス図、シーケンス図、フィーチャー内の操作概要）として表現する。本設計手法は、機能方式設計で、フィーチャーの粒度を決めるようにしたことが特徴（工夫ポイント）である。これにより、機能仕様から詳細設計までの成果物が、フィーチャー単位に、相互に参照できるようになっている。

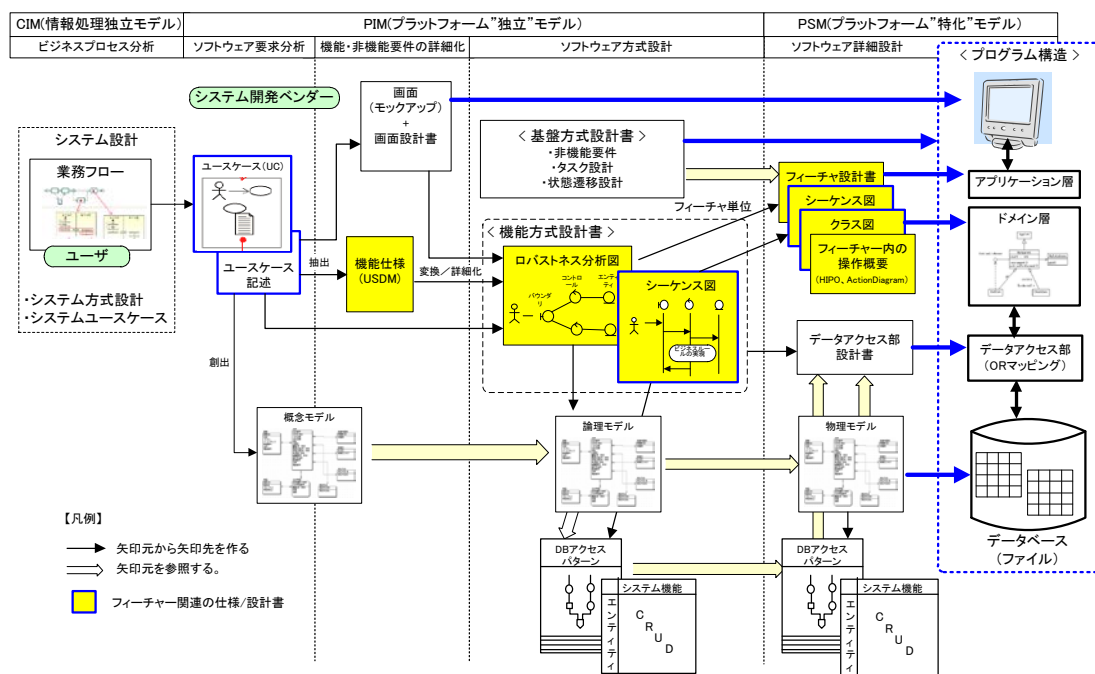


図 15-A-6-2 ソフトウェア設計手順

図 15-A-6-2 は、汎用系（業務系）の表現として示しているが、プロセスの一部を修正することで、汎用系、組込み系のどちらにも使える開発手法となっている。例えば、汎用系のシステム設計での「業務フロー」は、組込み系のシステム設計では「外部イベント」などに置き換えることができる。また、同様に汎用系におけるソフトウェア詳細設計の「データアクセス部のデータベースアクセス」については、組込み系のソフトウェア詳細設計での「ハードウェアへのアクセス」などに置き換えて考えることができる。

また、本事例では、モデリングを中心にシステムを開発していく MDA の考えを取り入れているので、MDA の分類を図 15-A-6-2 の上部に図示（CIM、PIM、PSM）した。

CIM（Computation Independent Model）は、ドメインに着目して開発するモデルで、コンピュータの処理方式は取り扱わない。PIM（Platform Independent Model）は、実装技術は特定せずにドメインの中で使用される処理方式に着目するモデルである。PSM（Platform Specific Model）は、実装技術を特定した上で詳細化するモデルである。

本設計手法の各開発フェーズにおいて採用している設計アプローチは、表 15-A-6-1 に示すとおりである。

表 15-A-6-1 設計アプローチ

開発フェーズ	図 15-A-6-2 の工程	キーとなる設計アプローチ
要件定義	ソフトウェア要求分析、機能・非機能要件の詳細化	ユースケース、USDM
方式設計	ソフトウェア方式設計	ロバストネス分析、フィーチャー概念、ドメイン駆動開発、OR マッピングアーキテクチャモデリング、品質特性
詳細設計	ソフトウェア詳細設計	クラス内処理部の構造化（HIPO ⁵ 、アクションダイアグラム）、ピアレビュー

4.2. ソフトウェア要求分析

ソフトウェア要求分析は、システム設計で作成された業務フロー、システム方式設計、システムユースケース（とユースケース記述）、などから機能を抽出し機能仕様を作成することである。

（1）機能仕様

機能仕様書は、USDM を使って記述する。USDM の形式を図 15-A-6-3 に示す。USDM の特徴は、①要求と仕様を分離する様式となっていること、②要求と複数の仕様の対応が分かり易いこと、である。

⁵ Hierarchy plus Input Process Output



図 15-A-6-3 USDM 形式

(2) ユースケースと USDM の対応関係

ユースケースは、アクター（ユーザー）が、システムの中をブラックボックスとして、システムの外側から見えるシステムの振る舞いを表現したものである。USDM は、システム内部の振る舞い（例：データ変換仕様等）を表現したものである。ユースケースと USDM の関係を図 15-A-6-4 に示す。

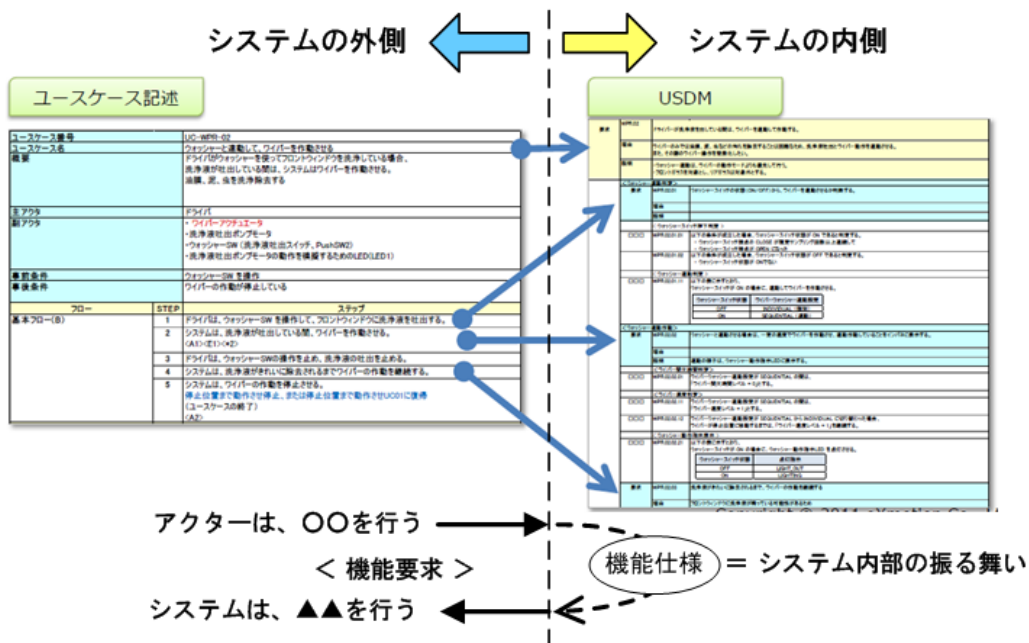


図 15-A-6-4 ユースケースと USDM の関係

ユースケースと USDM を対応づけるルールは以下のとおりとする。

- ・ ユースケース名（ユースケース全体）を USDM の上位要求に対応づける。
- ・ 各ステップは USDM の下位要求あるいはグループに展開する。
- ・ 各ステップの実現方法を仕様展開する。

アクターの振る舞いステップの仕様としては、システムがそのイベントに反応して動作する条件、イベントのインターフェース仕様、イベントを受けた直後に行うシステムの振る舞い等を考える。アクターとシステムの振る舞いを別々の要求と考えて仕様を考えるのが難しい場合は、それらをひとまとめの要求として仕様を考えるのが良い。

4.3. ソフトウェア方式設計手順

ソフトウェア方式設計は、大きく「基盤方式設計」と「機能方式設計」の2つに分けて行う。設計書も2つに分けるのが良い。

(1) 基盤方式設計

要件は、機能要件と非機能要件から構成される。非機能要件は、制約と品質を含む。そして、非機能要件の客観的な尺度として品質特性（品質属性とも言う）を導入する。品質特性は、いくつかの種類に分類できるが、ここでは、6 種類（可用性、変更容易性、性能、使いやすさ、テスト容易性、セキュリティ）を使う。

基盤方式設計では、静的構造（モジュール・ビュー）、動的構造（コンポーネント・コネクタービュー）、配置的構造を表現する。静的構造とは、時間に関わらず変化しない構造であり、レイヤ関係、クラス、関数等の定義、実行時のタスクやリソースの構造である。動的構造とは、静的構造に現れる要素（クラスやタスク）の相互作用、つまり望ましいシステムの振る舞いの表現である。そして、基盤方式設計項目が1対1に非機能要件に対応することはなく、複数項目の組合せで、非機能要件を満足することになる。逆に、ソフトウェア構造（静的／動的）は、性能、信頼性、保守性など、ソフトウェアのさまざまな品質特性を決定づける要因となっている。

この様子を図 15-A-6-5 の基盤方式設計書の記述内容に示す。

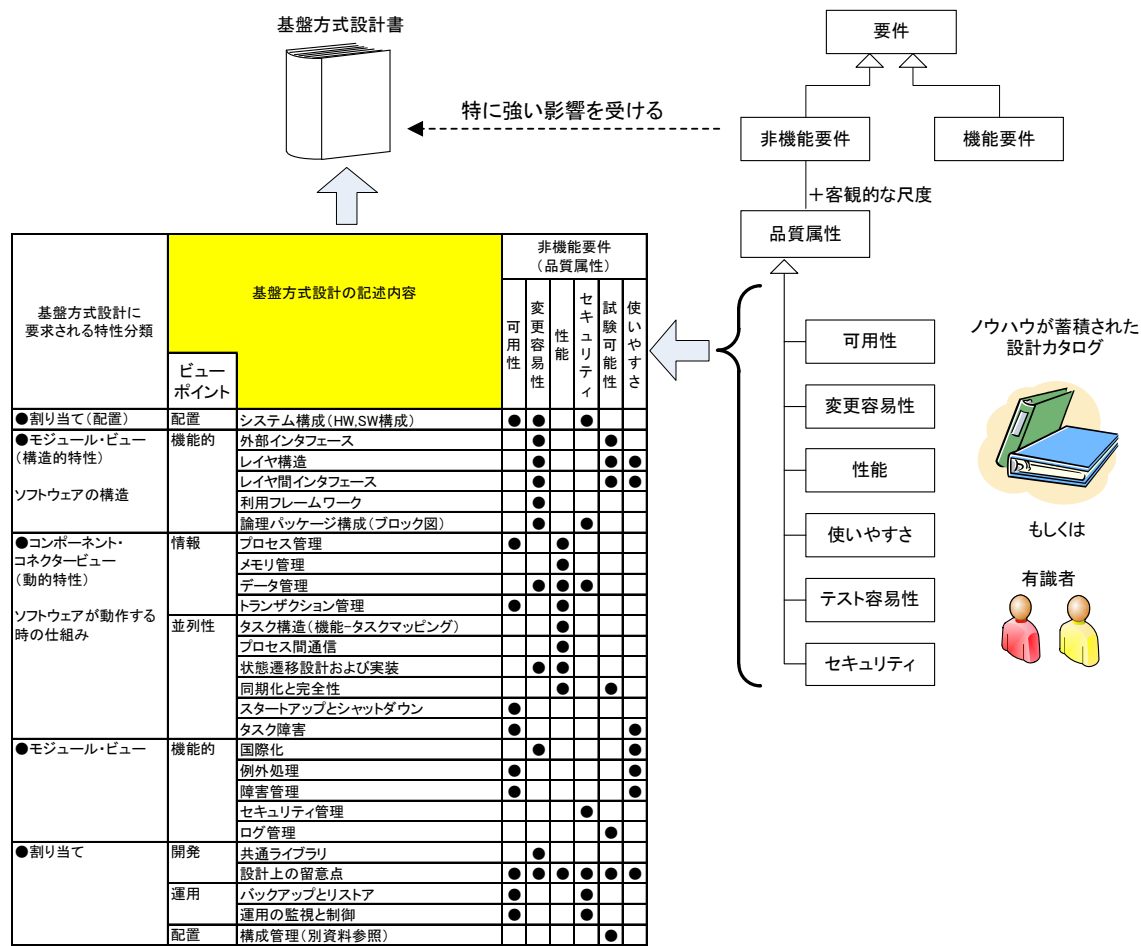


図 15-A-6-5 基盤方式設計書の記述内容

基盤方式設計は、主として非機能要件(品質属性)を実現する具体的な仕組み(How)を表現するソフトウェア構造(タスク構造、ファイルの分割構造、重要な処理手順)を設計するものであり、次の2つの作業を行う。

- 1) 非機能要件を満足するために必要な設計上の方針、制約事項や設計上まもるべき原則、メカニズムの設計パターンを示す。
- 2) 次に、それらを踏まえた上で、非機能要件を満足するソフトウェア構造はどうあるべきかを“具体的な仕組み”として表現する。

ここで重要なのは、次の3点である。

- ① 基盤方式は、ざっくりとした雰囲気(方針)を伝えるものではなく、開発、保守において具体的な判断の拠り所となるべき内容でなければならない。例えば、例外処理に考慮すべきコーディング規約まで含めた汎用的な準正常処理を記述しておき、詳細設計者すべてに考慮させる等。
- ② 非機能要件と基盤方式設計の記述内容のマッピング表は、必ず作成して設計書に入れる。これが、非機能要件をどのような設計で実現したかのトレーサビリティ

ティ・マトリクスとなる。マトリクスの中に書かれている黒丸（●）は、重要性によって、高／中／低と区別するのが望ましい。

- ③ 具体的なメカニズムを考える際の検討資料（トレードオフ表等）は別資料として作成し、該当箇所から参照できるようにする（これが真のノウハウである）。

(2) 機能方式設計

機能に対する要求は、典型的には入力からどのような出力が得られるかという、『入力から出力への変換』として定義される。機能方式設計は、この機能要求を満足する概念的な（論理的な）構造や振る舞いを明確にする作業である。ただし、入力に対してとにかく出力すれば良い、ということではなくシステム内部の状態（例：リソース残量）に応じたリアクティブな振る舞いに関する記述は必要となる。

機能方式設計の目的は次のとおりである。

- ・ ユースケースを実現するために必要なオブジェクトやクラスを識別する。
- ・ 必要なオブジェクトやクラス間の関係を識別する。
- ・ 属性や操作を識別して、それらをクラスに割り当てる。
- ・ 分析シーケンス図を使用して操作の振る舞いを特定する。

(3) 基盤方式設計と機能方式設計の進め方

基盤方式設計と機能方式設計は相互に関係するので並行に進める。その設計作業の進め方と内容を図 15-A-6-6 に示す。

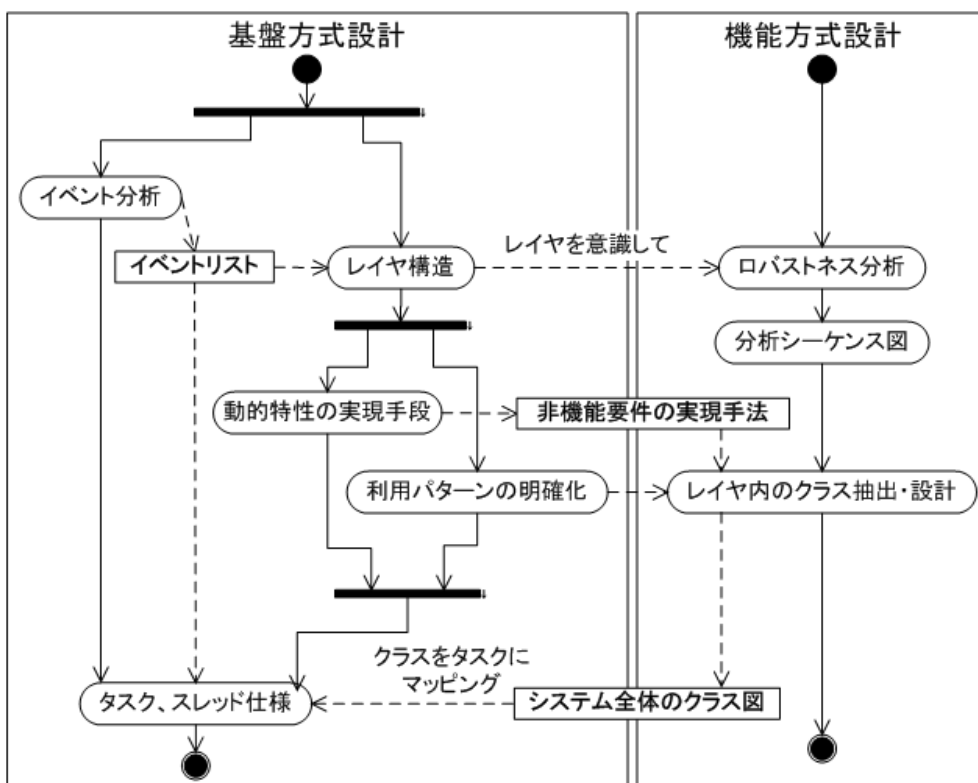


図 15-A-6-6 機能方式設計と基盤方式設計の進め方

4.4. 機能方式設計手順の詳細

機能方式設計では、機能を構成する論理構造（コンポーネント）を明確にすることから始める。その基本は、意味的にひとまとまりの処理、ひとまとまりの情報をできるだけ同じコンポーネントにまとめるということである。これは、ロバストネス分析を利用して行う。

(1) ロバストネス分析

図 15-A-6-7 に示すロバストネス分析図は、アプリケーションに要求される機能を、バウンダリ（画面、印刷）、コントロール（機能）、エンティティ（データ、ファイル）に分類し、それぞれの関係、振る舞いを表現するものである。

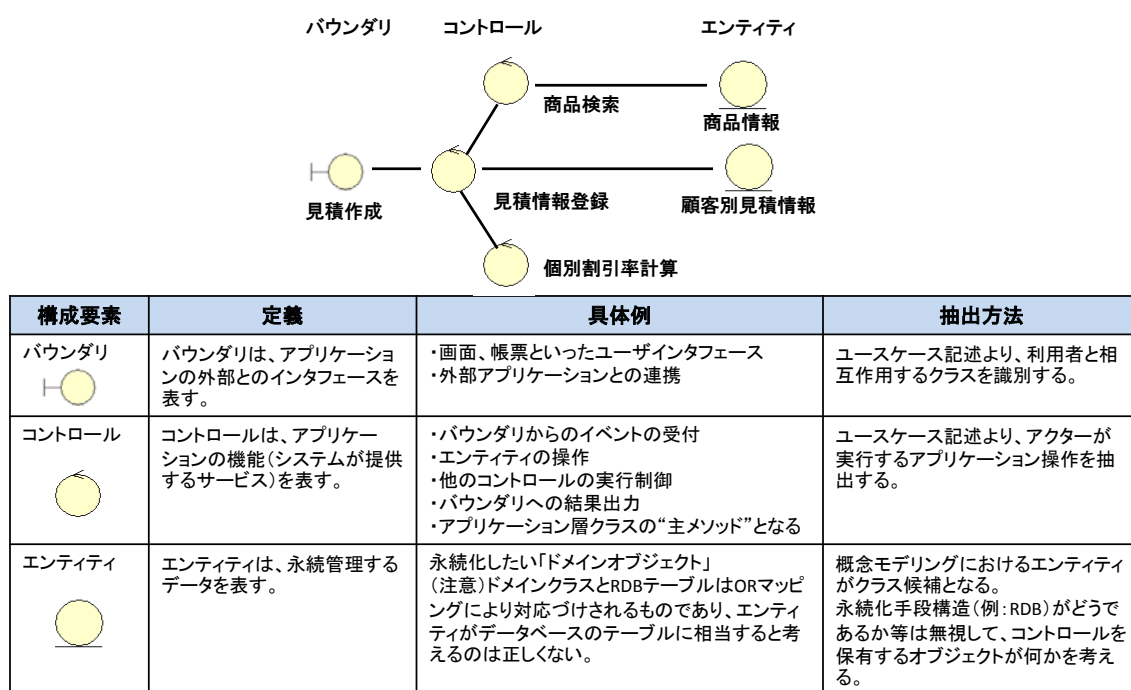


図 15-A-6-7 ロバストネス分析図

ロバストネス分析図作成の粒度は、ユースケースに対応させて作成する。つまり、ユースケースひとつに対して、ロバストネス分析図を1つ作成する。

(2) ユースケース／USDM 仕様／ロバストネス分析図の関係

ロバストネス分析図はユースケースをオブジェクトの絵として表現したものである。つまり、ユースケース記述と機能仕様である USDM から、アプリケーションの機能コンポーネントの構成（関連）を決めるのがロバストネス分析図であると考えることができる。ただし、ロバストネス分析図では、アプリケーションを「どのように」実行するかではなく、アプリケーションに「なに」をさせるかに焦点をあてている。

ユースケース（ユーザー要求）と USDM（機能要求）とは、4.2 (2) で示した関係

で繋がっている。したがって、ロバストネス分析図のコントロールと USDМ の仕様のトレーサビリティを取ることで、コントロール仕様、すなわち、コントロールが「なに (what)」を行わなければならないかが明確になる。そして、この仕様に基づいて、ソフトウェア詳細設計では「どのように (How)」実行するかの実現方法を考える。この関係を図 15-A-6-8 に示す。

文章のみで記述されている機能方式設計書では、要求仕様に記述している内容の多くを再記述している場合も多く見られるが、この設計手法では、USDМ に要求と仕様を集約し、参照関係 (トレーサビリティ) を用いることで、二重に記述することを防ぐことができる。

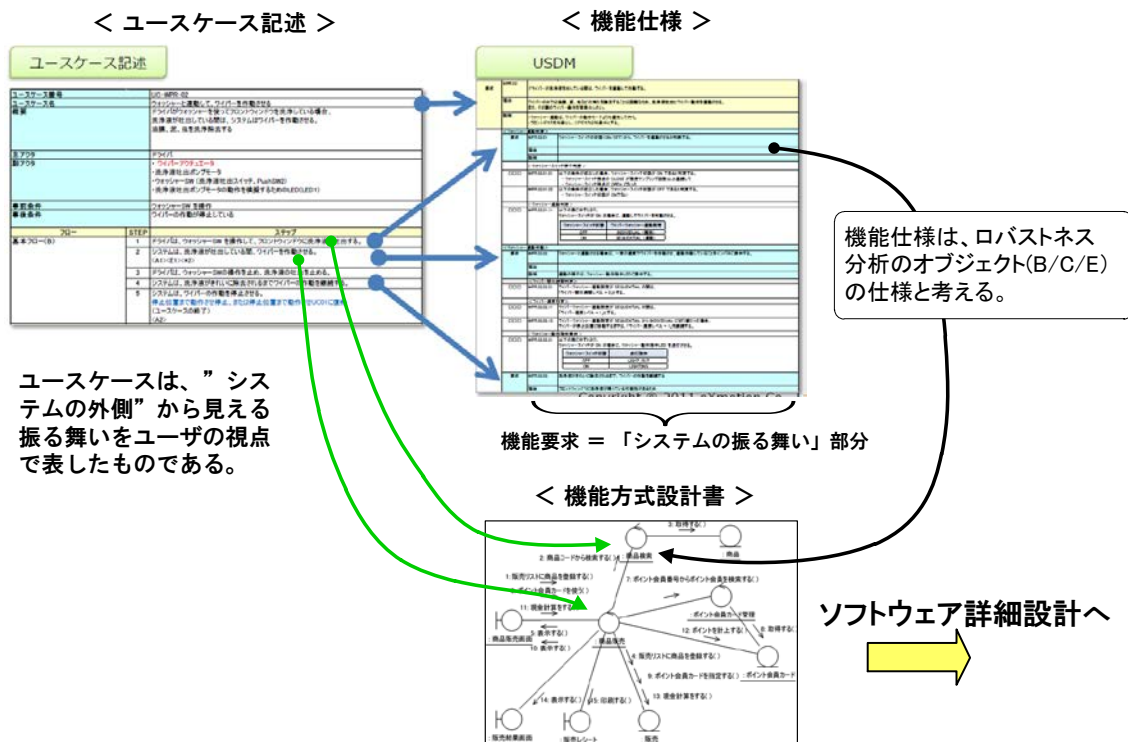


図 15-A-6-8 ユースケース/USDМ/ロバストネス分析図の関係

(3) 分析シーケンス図作成

1) 作成の目的

分析シーケンス図は、ユースケースを実現するために必要なオブジェクト間のやり取りを時系列に表現したものであり、ロバストネス分析図に出てきたクラス及び、それを補完する主要なクラス間の関連、クラスの属性、操作を見つけ出すために作成する。

この分析シーケンス図は、実装を忠実に表すものとして、ロバストネス分析図で登場したすべての論理的な静的クラスに加えて、ソフトウェアの実現に必要な新たなク

ラスも登場させて、それらの振る舞いを表現するものである。

2) 具体的な作成方法

ロバストネス分析図と分析シーケンス図の関係を図 15-A-6-9 に示す。

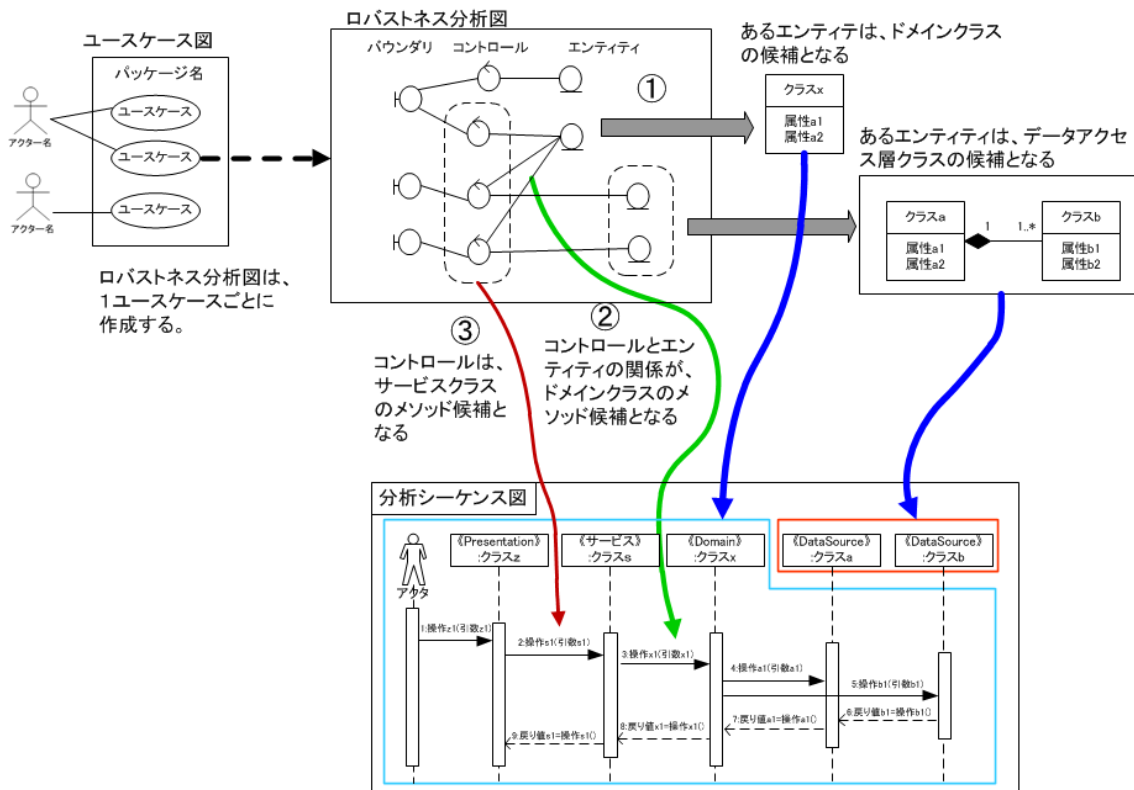


図 15-A-6-9 ロバストネス分析図と分析シーケンス図の関係

アプリケーション機能の動的な振る舞いを分析するために、ロバストネス分析図の構成要素である、バウンダリ (B)、コントロール (C)、エンティティ (E) をオブジェクトにした分析シーケンス図を作成する。

コントロールとエンティティを結ぶ線は、それぞれのクラスのメソッドに相当するので、これを分析シーケンス図のメッセージで表現する。コントロールは、アプリケーション層のクラスの主メソッド (サービス) になると考える。

4.5. 機能方式設計から詳細設計への展開

詳細設計は、方式設計を入力として、フィーチャーのプログラム実装方法を検討する作業である。

詳細設計では、主に下記の成果物を作成する。

- ① クラス図：プログラムとして実現するためのクラス
- ② シーケンス図：プログラムとして実現するためのオブジェクトの振る舞い
- ③ 状態遷移図：プログラムとして実現する上で重要な状態遷移

- ④ アルゴリズム設計書：②のシーケンス図では表現できない処理手順

この時、重要なのは次の 2 つである。

- ① フィーチャーの仕様範囲は、対象となる機能仕様（USDM）とトレーサビリティを確立することで明らかになる（図 15-A-6-10 参照）。
- ② 詳細設計と基盤方式設計で規定している内容に矛盾がないことを必ずレビューで確認する。

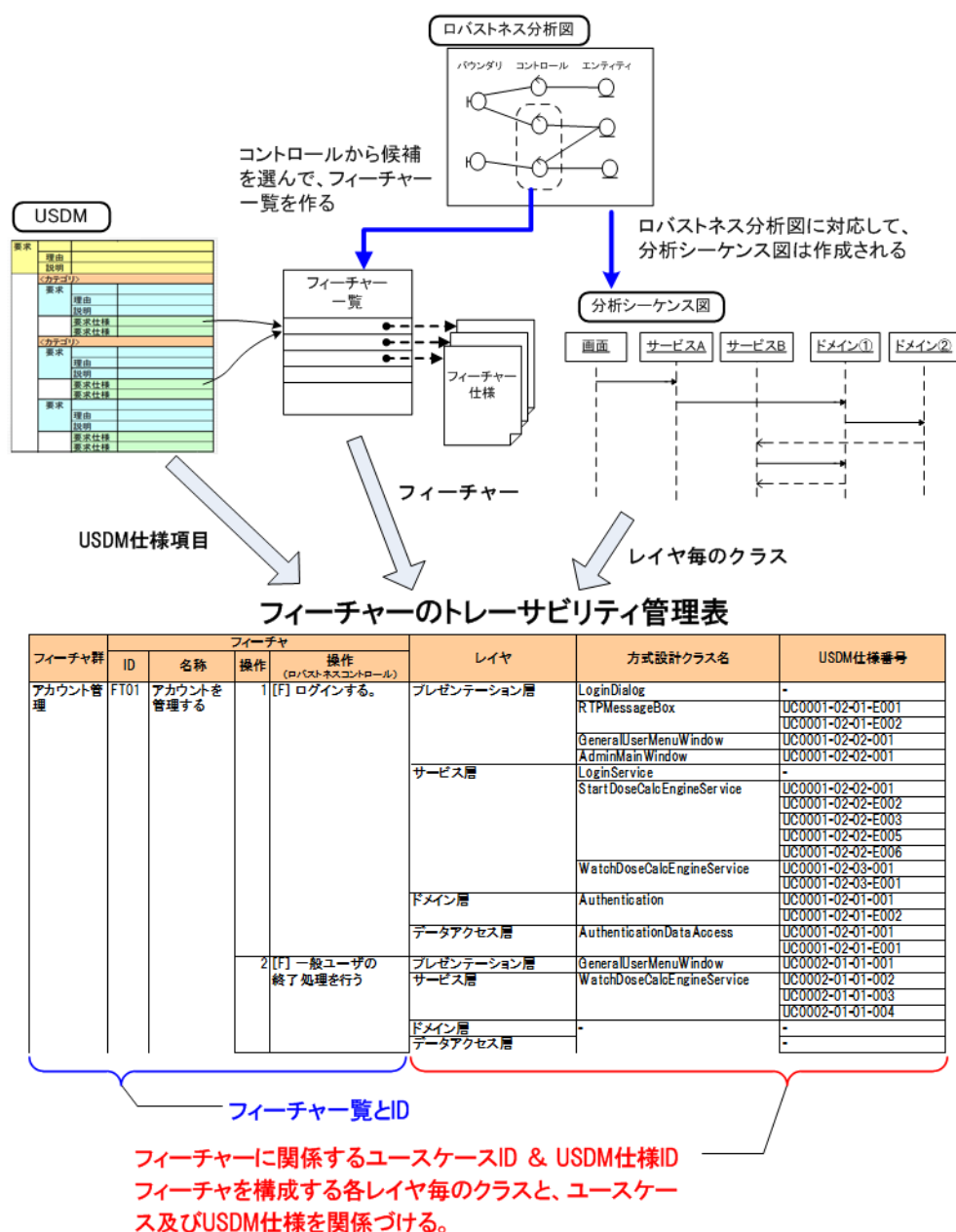


図 15-A-6-10 フィーチャーのトレーサビリティ管理表

4.6. アジャイル開発としての捉え方

アジャイル開発とは、「顧客価値を持続的に提供するために、繰り返しながら、コミュニケーションを重視し、ソフトウェア開発に取り組むやり方」である（設計せずにコーディングを始める開発というのは誤解）。

本事例でのフィーチャー単位での開発プロセスは、一番重要な部分から、動くソフトウェアを細かく作って完成させていく「短期の繰り返し型」であることが一つの特徴であり、アジャイル開発として適用できる。

本事例の中では、独立性の高い、ユーザーから見た小さな機能であるフィーチャーが繰り返しの単位となる。フィーチャーは、設計とプログラム開発を2週間程度（もしくは更に短い期間）での開発が可能な単位であり、独立性が高いので複数人で並行開発、かつ、反復（イテレーション）開発ができる。

4.7. 短納期開発が可能

フィーチャーは、それぞれが独立性の高いサービスとして区分したもののなので、並行開発ができる。

また、仕様変更、設計変更時でも、ドメイン層がかなりの割合で流用可能なまでに成熟していれば、プレゼンテーション層とアプリケーション層内のフィーチャーの追加・修正でプログラムを完成させられる。

5. 適用時の課題と対策

(1) 現場への導入時の具体的な工夫点や苦労

結果的に、現場の理解が難しかったのは、①「フィーチャーとは何か」、②「ドメイン駆動によるドメイン層とフィーチャーの構造の違いと具体的な作り方」、であった。

それらを理解させる方法として、プログラム構造でのフィーチャーとドメインの違いを表 15-A-6-2 のフィーチャー・ドメイン関係で説明した。

表 15-A-6-2 フィーチャー・ドメイン関係

フィーチャー	ドメイン層
手続き型で作成する	オブジェクト指向型で作成する
ドメインを駆動するもの	フィーチャーから駆動されるもの
自己の中にドメインに関する共通処理、アルゴリズム構造等を持ち込まず、可能な限り、ドメインを駆動するだけのシンプルな構造にする。	そのドメインのノウハウが含まれるので複雑になる（なって良い）。
開発システム毎に異なる。	開発システムが異なってもドメインが同じであれば共通となる部品群である。

作成順序は、まず、有識者が先行してドメインを作成し、その後、ドメインを利用

する制御構造としてフィーチャーを作成する。

また、ロバストネス分析図を受けて、方式設計段階の分析シーケンス図は、より詳細化しなければならないにもかかわらず、ロバストネス分析図と分析シーケンス図の粒度が同じレベルになることがあった。分析シーケンス図には、ロバストネス分析図のコントロール以外に、ここに出てきていないオブジェクトも登場させて、どう処理するかを検討する必要がある。これは、方式設計のコア部分であるが、このことが理解されていなかった。解決方法として、「設計ガイドライン」を作成し、より具体的な手順を周知させた。

(2) 現場への導入時の課題とその対応

(1) で述べた、現場を理解させることが難しかったことについては、全員が理解するよりも、技術キーマンが正しく理解することの方が重要であることが分かった。なぜなら、ソフトウェア開発の場合、多くのエンジニアは技術キーマンの指導に従ってしまうからである。したがって、現場での成果物により、その理解度を確認し、理解が不十分な部分に関して何度も繰り返し、技術キーマンの指導を行った。

ドメイン層の共通化を進めるために、ドメイン層をフレームワークにする開発を先行して行った。フレームワークの開発は、ライブラリ等で共通部品を作るよりも難しく、ドメイン層の共通部分の規模が増えていかないという現象が生じた。これに対しては、いくつかのプロジェクトに対し横展開し、共通化の候補を増やしていくことにした。

6. 導入後の改善活動と今後の課題

(1) 導入後の改善活動

開発プロセスが設計手順、設計フォーマットまできめ細かく決められているので、品質メトリクス基準の有効性改善が継続的に進められる。CMM⁶レベル4を取得した10年前（2004年）から、品質メトリクスは事業部統一（横通し）で測定しているが、単体テストの粒度、方式設計と詳細設計の境界の曖昧さなどがあり、必ずしも、現場が全面的信頼をおくような有効な品質基準が作れているわけではない。

品質データ計測文化は定着しているので、開発フェーズ間の移行判定に十分に活かせる品質メトリクス基準に修正していく活動を進行させている。

(2) 今後の課題と対応

社員とは別に、協力会社は従事者が変わるので、その都度教育を行う必要がある。

⁶ Capability Maturity Model

7. 導入体制と適用プロジェクトの概要

(1) 導入体制

本開発プロセスの作成は、専任者1名で行った。また初期の設計ガイドラインの整備も同様に1名で行った。パイロットプロジェクトに適用したときは、利用するエンジニアの言葉で、より詳細な「設計ガイドライン」としてまとめた。この設計ガイドライン作成は、3名で行った。

(2) 適用プロジェクトの概要

本開発プロセスを適用したプロジェクトの例を表15-A-6-3に示す。

表 15-A-6-3 適用プロジェクトの例

	初期開発			仕様変更（派生開発）		
	規模	工数	期間	規模	工数	期間
A	149KL	165 人月	11 か月	24KL	24 人月	4 か月
B	25KL	25 人月	4.5 か月	29KL	29 人月	6 か月

8. 品質について

(1) 定性的品質状況

ソフトウェア要求分析～ソフトウェア詳細設計までシームレスに繋がりが明確になり、開発フェーズ毎に何を成果物とするかが、これまでよりも明確に決めることができた。また、試験で発見された不具合は、①本来どの設計フェーズのレビューで発見されるべきものであったのか、②試験でしか見つけられない仕方のない不具合なのか、の区分を明確にすることができるようになった。その結果、「ソフトウェア要求仕様」の品質が、プロジェクト全体の品質に最も大きく影響することが分かった。

(2) 定量的品質状況

あるプロジェクトの途中段階の品質データを表15-A-6-4に示す。この品質データから品質の妥当性を評価する。社内開発標準から考えて、ユースケース数は妥当であり、フィーチャーのクラス当たりの仕様数は経験的に少ないと思われる。したがって、もう少し仕様（USDM）の記述を増やす必要がある。

表 15-A-6-4 品質データ

項目	実績値	
システム規模	117 KL	
業務フロー	8 数	
ユースケース	152 U.C	0.8 KL/U.C
		19 U.C/業務フロー
仕様（USDM）	1349 項目	11.5 仕様/KL
クラス	1334 クラス数	11.4 クラス/KL
フィーチャー数	387 クラス数	3.5 仕様/クラス
メソッド	8411 メソッド数	13.9 Line/メソッド

9. 生産性について

新規作成部分に関しては、7（2）にあるように、ソフトウェア要求分析から適格性確認テストまでで、0.9KL/人月～1.0KL/人月の生産性となり、適正である。

さらに以下のようなことから、さらなる生産性の向上が期待できる。

- ① ドメイン層が複数プロジェクト間の共通部となるため、ソフトウェアプロダクトラインの実現が可能となる。
- ② ソフトウェア要求分析～ソフトウェア詳細設計までシームレスに繋がりが明確になり、トレーサビリティも厳密に取れるので、改修時には必要な設計書が取り出しやすく、影響範囲の把握と理解が速くなっている。また、どこに何が書かれるかが明確なため、トレーサビリティをたぐらなくても、ある程度、記述場所を見つけやすい。
- ③ 基盤方式設計は、プロジェクトに依存しない設計を集めた物なので、設計の流用部分が明確になっている。

10. 検証への影響

設計品質の良し悪しを計る指標として、「成果物項目間の多重度」を用いることを考えた。成果物項目間の多重度とは、表 15-A-6-5 のように、ある成果物の項目が、別の成果物の項目といくつの関係を持っているかを示す指標値のことである。

良い設計が行われた際、成果物項目間の多重度を取るべき大凡の範囲が予め分かっているならば、その範囲内にあれば良い設計、範囲外であれば悪い（どこかおかしい）設計、と判断できるようになる。

表 15-A-6-5 多重度管理表

	USDM仕様項目			フィーチャー			クラス			プログラム規模		
	最小	最大	単位	最小	最大	単位	最小	最大	単位	最小	最大	単位
ユースケース(UC)	5.0	15.0	仕様/UC	3.0	6.0	FT/UC	8.0	20.0	クラス/UC	1.0	1.8	KL/UC
USDM仕様項目				3.0	5.6	仕様/FT	3.8	5.9	仕様/クラス	14.0	22.0	仕様/KL
フィーチャー(FT)							1.0	3.0	クラス/FT	88.0	178.0	Line/FT
クラス										10.0	18.0	クラス/KL
メソッド										15.0	25.0	Line/メソッド

11. まとめ

開発プロセスに構築するにあたり、小さな機能のかたまりであるフィーチャーに着目して、アジャイル設計手法の1つであるFDD（Feature Driven Development：ユーザー機能駆動開発）の考え方をベースにするとともに、MDA（Model Driven Architecture：モデル駆動

アーキテクチャ) の考え方を取り入れた。

構築した開発プロセスのプロジェクトへの定着を図るため、教育やパイロットプロジェクトへの適用などトップダウンで進めた。また、開発プロセスを使う側である開発者による設計ガイドラインの作成など、利用者である開発プロジェクトに合わせたカスタマイズ等にも対処した。さらに、開発プロジェクトでの開発者への影響度が高い技術キーマンの指導を重点的に実施した。

このような工夫や施策により、ソフトウェア要求仕様から詳細設計までの開発プロセスが明確化され、トレーサビリティの確立もできた。新規プロジェクトでは、本プロセスを適用し易くなり、開発プロセスの定着が進んでいる。

参考文献

- [1] 要求から詳細設計までをシームレスに行うアジャイル開発手法 藤原啓一 MSS 技報・Vol.24
- [2] ソフトウェアアーキテクチャ 岸知二、野田夏子、深澤良彰、共立出版、2005.6
- [3] 実践ソフトウェアアーキテクチャ Len Bass/Paul Clements/ Rick Kazman 前田卓雄 他訳、日刊工業新聞社、2005
- [4] システムアーキテクチャ構築の原理 IT アーキテクトが持つべき 3 つの思考 (IT Architects' Archive ソフトウェア開発の実践) ニック・ロザンスキ、イオイン・ウッズ、榊原 彰、牧野祐子、翔泳社、2008.12.3
- [5] リアルタイム UML ワークショップ ブルース・ダグラス 鈴木尚志訳、翔泳社、2009.12
- [6] 入門 オブジェクト指向設計—変更に強く生産性が高いシステムを 滝澤克泰、ソフトバンクパブリッシング、2005.1
- [7] [改訂第 2 版] [入門+実践] 要求を仕様化する技術・表現する技術・仕様が書けていますか? 清水吉男、技術評論社、2010.6
- [8] 「派生開発」を成功させるプロセス改善の技術と極意 清水吉男、技術評論社、2007.11
- [9] Java/Web でできる大規模オープンシステム開発入門 林浩一、他 丸善出版、2012.10
- [10] ユースケース駆動開発実践ガイド ダグ・ローゼンバーグ、マットステファン、佐藤竜一他訳、翔泳社、2007.10
- [11] エンタープライズアプリケーションアーキテクチャ・パターン マーチン・ファウラー 長瀬嘉秀監訳、翔泳社、2005.4
- [12] エリック・エヴァンスのドメイン駆動設計 エリック・エヴァンス 今関剛監訳、翔泳社、2011.4
- [13] リアルタイム組込 OS Qing Li/Caroline Yao 翔泳社、2005.11
- [14] 組み込みソフトウェアの設計&検証 藤倉俊幸、CQ 出版、2006.9

掲載されている会社名・製品名などは、各社の登録商標または商標です。

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター (IPA/SEC)