81

統合モデル技術によるユーザ主体のソフトウェア開発手法「モデル指向開発（MOD）」の紹介 MOD：Model-Oriented Development[1]

1. 適用手法の概要

1.1. モデル指向開発の必要性

ソフトウェア開発と建築は、よく似ていると言われるが、本質的に大きく異なると判断している。建築物は、ソフトウェアと同様に構造の概念を持つが、基本的に振る舞いの概念を持たない。それに対して、ソフトウェアの本質は、構造よりも振る舞いの概念にあると考えているからである。そのため、ソフトウェアは構造よりも振る舞いに、より大きな価値を有し、また、ソフトウェア開発では、建築と比較して、より多様な品質特性を考慮しなければならない。

以上は、ソフトウェア開発と建築についての私見であるが、ソフトウェア開発にモデルを活用しようと考えたきっかけであり、重要な発想であると考えている。

そこで、開発すべきソフトウェアの構造と振る舞いを把握するには、ソフトウェアの多様性や複雑性に対処する有効な手段が必要となる。その有力な手段の一つとしてモデルを活用したソフトウェア開発のアプローチがある。ここでモデルとは、ソフトウェアを開発する際、対象を抽象化し単純化して表現したものである。別な言い方をすれば、複雑な対象について余計な部分を削ぎ落とし本質的な部分のみに着目して形式的に単純化したものである。通常モデルは図式やマトリクス表などで可視化して表現される。

また、モデリングとは、視点を決めて対象を抽象化することにより、複雑さを解消して単純化する行為である。言い換えれば、対象をモデル化することである。モデリング技法をソフトウェア開発に導入したメソドロジとして代表的なものは、モデルベース開発（MBD：Model-Based Development）とモデル駆動開発（MDD：Model-Driven Development）である。

モデルベース開発は、制御工学の分野で発展してきた開発手法であり、主に連続系システムを対象として、MATLAB/Simulink ツールなどを用いて数式モデルを扱う。一方、モデル駆動開発は、情報工学やソフトウェア工学の分野で発展してきた開発手法であり、主に離散系システムを対象として、UML（Unified Modeling Language）ツールなどを用いて論理モデルを扱う。2 つの開発手法は、これまで分野を分けて個々に発展してきたものである。

[1] 事例提供：株式会社日立産業制御ソリューションズ 渡辺 滋 氏

現状、スマートグリッド、スマート医療、自動車制御や Industrie 4.0、IIC（Industrial Internet Consortium）などの基盤となる IoT（Internet of Things）システムへの対応が求められる中で、両者を統合した開発手法のニーズが高まりつつある。このような状況下でモデル指向開発（MOD：Model-Oriented Development）は、モデルベース開発とモデル駆動開発の両手法を統合し、開発現場における経験に基づいた新たな着想やオリジナルな技法を織り込んだメソドロジであり、モデル技術を軸にしたソフトウェア開発手法の集大成となっている。ここで、モデル技術とは、ソフトウェアモデルに関する技術の略称であり、様々なソフトウェアモデルを活用した技術やモデリング技法の総称であると定義する。

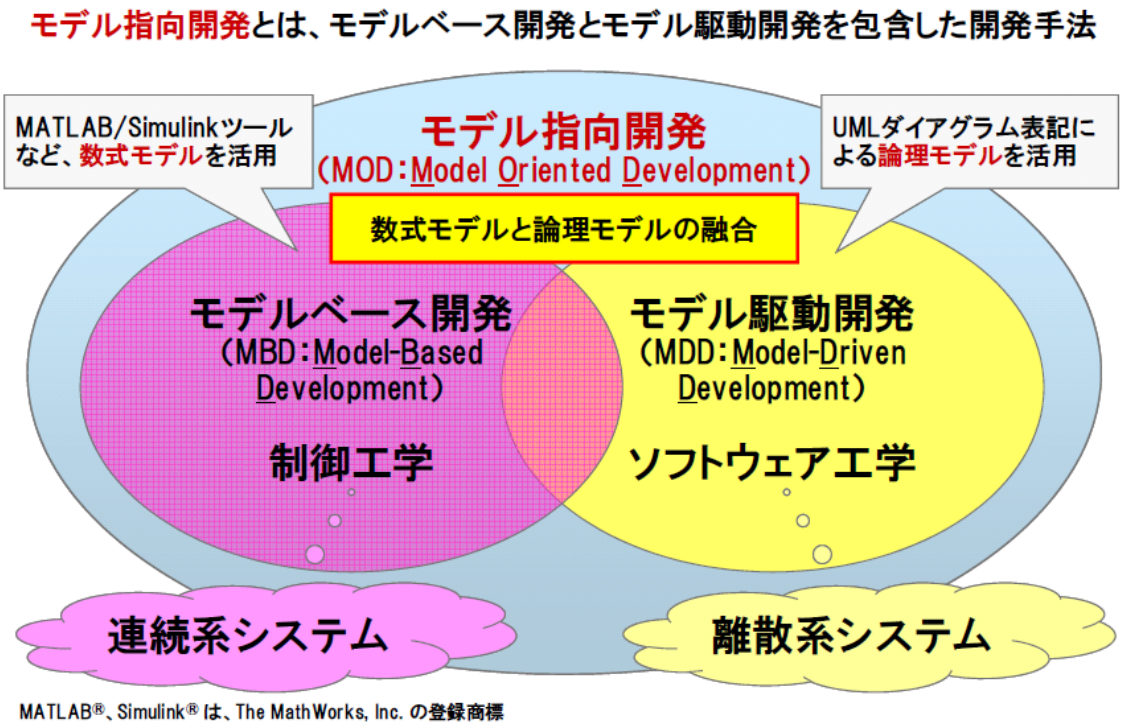


図 81-1 課題と対象となる開発分類

1.2. モデル指向開発の特徴

モデル指向開発のオリジナルな特徴は、開発の超上流から実装前までのアクティビティの細部に存在するが、ここでは代表的な要素技法のみに着目して以下に示す。

(1) 開発プロセスを固定化しない柔軟なフェーズ構成による適用プロセス技法

モデル指向開発手法は、固定化した開発プロセスを有していないので、適用対象の開発プロセスに応じた開発フェーズを構成することができる。適用対象となる開発プロセスで最も多いのはウォータフォール型開発プロセスであるが、他にインクリメンタル開発やアジャイル開発に代表される反復型開発プロセス、派生開発プロセスなど、様々な開発プロセスが存在する。モデル指向開発は、構成要素としての各開発フェーズにおけるアクティビティを定義しているだけなので、適用対象の開発プロセスに応じて適用プロセスを構成することができる。以降、

最も多いウォータフォール型開発プロセスを標準開発プロセスとして説明する。

図 81-2 は、モデル指向開発をウォータフォール型開発プロセスに適用する際の適用プロセスフロー例である。本図は、株式会社システムクリエイツの清水吉男氏が考案した PFD（Process Flow Diagram）の技法によって表記されている。要件定義フェーズから始まり、3 種類の分析技法を有する要件分析フェーズ、方式設計フェーズ、詳細設計フェーズ、テスト設計&プログラム製作フェーズ、単体テストフェーズ、および組合せテストフェーズから構成される開発プロセスとなっている。

(2) 要求仕様から設計につなぐ要件分析技法（図 81-2 の P2：要件分析フェーズ）

定義された要求仕様とソフトウェアの設計仕様との間には、かなりの距離があると感じている。その距離を意識して紐付けるためには、設計の準備段階として要求仕様の分析が必要であると考えている。分析は複数の視点で行うべきであるという考え方から、3 つの視点を設定している。すなわち、ソフトウェアの使い方（ユースケース）の視点、外部との相互作用（イベントケース）の視点、および相互作用のための動作条件（ステートケース）の視点である。開発対象に応じて、これらの視点を組合せて要求仕様の振る舞いを分析する。

(3) 設計方針決定と準備のための要件分析図法（図 81-2 の P2：要件分析フェーズ）

要件分析フェーズは 3 種類の視点による分析フェーズから構成されており、必要に応じて組合せた分析を行う。1 つ目は、ユーザの使い方の視点で分析するユースケース分析である。2 つ目は、外部との相互作用の視点で分析するイベントケース分析である。3 つ目は、相互作用のための動作条件の視点で分析するステートケース分析である。ここで、要件分析の中心となるのは外部との相互作用の視点であるイベントケース分析である。要求仕様としての振る舞いには、必ず外部からのイベントが関係しているという考えからである。イベントケース分析では、ソフトウェアの動作環境であるタスク構造やソフトウェアの構成単位であるモジュール構造、ソフトウェアの連携単位である関数構成、および必要となるデータ構造など、設計要素の規模や構成について仮決めを行う。イベント単位の外部との相互作用のモデル表記は、UML アクティビティ図を基に拡張したインタラクションフロー図を使用する。文法的には UML アクティビティ図を継承しているが、図の要素の意味付けについては関数分割やデータ構造分析、および責務抽出を行うために拡張されている。また、ユースケース分析は、一般に提示されているものとは異なり、2 種類のオリジナルな分析手法となっている。1 つは、要件分析の分析に使用するものであり、もう 1 つは、コンポーネント設計の準備としてユーザ機能の分析に使用するものである。ユースケース分析とステートケース分析については、「3.3　モデル指向開発の方法論」と「3.4　課題解決の展開」で説明する。

(4) 関数分割とデータ構造抽出の技法（図 81-2 の P2：要件分析フェーズ）

イベントケース分析では、インタラクションフロー図にて外部との相互作用としての振る舞いを明確にし、具体化していく。その際、振る舞いを関数処理の連鎖として分析するため、アクティビティノードを関数処理として扱い、その前後のオブジェクトノードを関数処理の入力、

出力としての一時データとして扱い表記する。ここで、関数の凝集度を最大に高めるための検討を行い、適切な関数処理の粒度に分割しながら、同時に関数処理から見た入力データと出力データを具体的に決定していく。また、関数の処理として保存すべき永続データがある場合には、一時データとは別に表記し、そのデータ構造の概略を定義していく。これらの要素を漏れなくインタラクションフロー図に明記していくのである。

(5) ソフトウェア構造を抽出する責務指向技法（図 81-2 の P3：方式設計フェーズ）

各インタラクションフロー図で表現された振る舞いについて、責務(責任、役割と同様)の視点で振る舞いの分担を考える。そのために責務を大きく 3 つに分類（boundary、control、entity）した上で、どのような責務が存在するかを考え、責務を抽出し各責務に名称を付与していく。この責務をソフトウェア・モジュールの単位であるクラスに割り当てることで、分析レベルにおけるクラス構造とクラス間の関係を定義してクラス構成を決定する。インタラクションフロー図において、責務の構成要素である一時データや永続データ、および関数は、各々クラスの属性や操作としてカプセル化され、クラス構造として定義される。

(6) 要件分析から設計につなぐ集約設計技法（図 81-2 の P4：詳細設計フェーズ）

各インタラクションフロー図の振る舞いから責務を抽出し、ソフトウェアのクラス構造とクラス間の関係を定義してクラス構成を決定するが、これは分析レベルのクラス構成である。分析レベルのクラス構成は、インタラクションフロー図単位に決定したものであり、開発するソフトウェア全体としての唯一のクラス構成になっていない。そこで、複数存在している分析レベルのクラス構造とクラス構成を集約して、結合度、凝集度を考慮しながら、設計レベルとしての唯一のクラス構造とクラス構成を設計する必要がある。ここで集約には、クラス内の属性に関する集約と操作に関する集約があり、それ以外にクラス間の集約もある。また、クラスの操作によっては、インタラクションフロー図から抽出できていない制御処理構造が必要になる場合があるため、新たな操作として追加することを考えなければならない。これらの一連のソフトウェア設計が集約設計である。

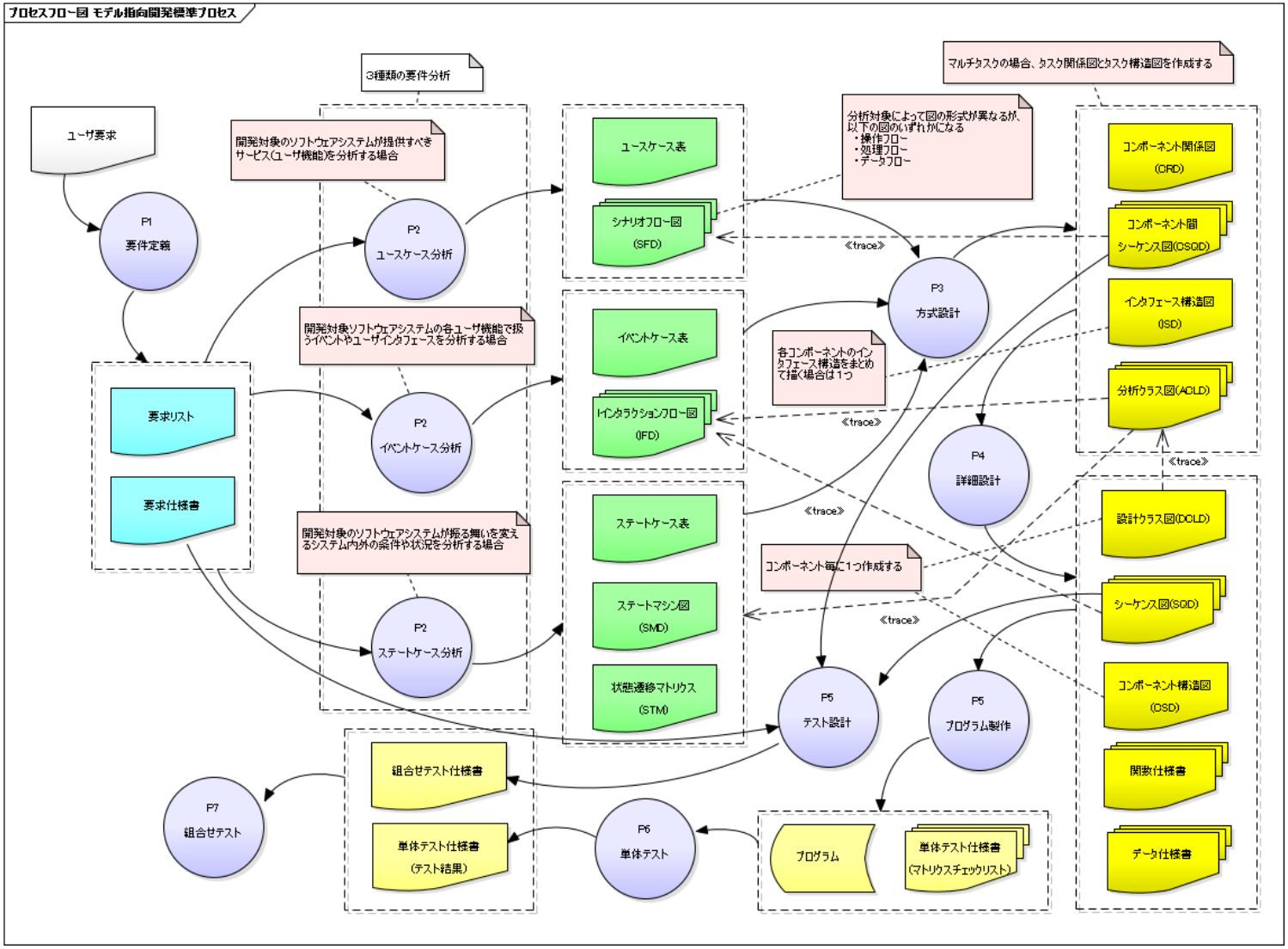


図 81-2 モデル指向開発のウォータフォール型適用プロセスフロー例

2. 適用手法の目的

モデル指向開発手法を考案した主な目的としては、以下に述べるような開発現場の問題点を解決することであった。

2.1. 開発現場の問題点と対応策としての課題

ソフトウェア開発の現場では、ソフトウェアに対するユーザ要求をまとめることから始まり、要求を実現させるソフトウェアの要求仕様を定義し、要求仕様に従ってソフトウェアの設計、実装、テストを行っているが、それらの開発工程の中では、様々な問題が発生している。

当社では、派生開発が全体の 80%以上を占めていることから、派生開発で発生している問題点の比重が大きくなっている。派生開発の現場における大きな問題点としては、以下のものが挙げられる。これらの問題点は、新規開発に起因しているところが大きく、同時に新規開発の問題点でもあると考えることができる。

問題点 1：機能の追加や変更に対して既存のソフトウェアが大きな影響を受けてしまう
問題点 2：ソフトウェアの変更がどこに影響するのか把握できていない
問題点 3：変更するたびにソフトウェアの構造が劣化して複雑になっていく

以上の問題点が発生している原因としては大小、様々なものがあると思うが、以下の原因が根本的であると考えている。

原因 1：ユーザが考えている機能の単位でソフトウェアが設計されていない

ユーザが考えている機能とは、ソフトウェアを使用する側から見た外部的な機能である。それに対して開発者が考える機能とは、ソフトウェアを構築する側から見た内部的な機能である。この両者は必ずしも一致するとは限らず、むしろ一致しない場合が多いのである。そのギャップがユーザからの機能の追加や変更に対して大きな影響を与えることになる。

原因 2：ソフトウェア開発におけるトレーサビリティが確保されていない

ここでのトレーサビリティとは、上流のユーザ要求から要求仕様、設計仕様を介して実装するプログラムまでのトレース、およびその逆となる実装プログラムからユーザ要求までのトレースが常に可能なことである。一般的なトレーサビリティでは、テストまで含めるが、ここではテストを除いて考えている。トレーサビリティが確保できていなければ、既存ソフトウェアの変更を伴う派生開発において、ソフトウェアの品質や開発効率を低下させることになり、短期間での開発が困難となる。

原因 3：適切なソフトウェア・アーキテクチャの設計と維持管理がされていない

適切なソフトウェア・アーキテクチャとは、ユーザ要求を実現させるソフトウェアの要求仕様を根拠として論理的に設計されたソフトウェアの構造（静的視点）と振る舞い（動的視点）のメカニズムであると言える。適切に設計されたソフトウェア・アーキテクチャは、明確で分かりやすいと共に、無駄がなくコンパクトで、かつ効率的に動作するソフトウェアを生み出す。新規開発では適切なソフトウェア・アーキテクチャを設計し、派生開発では適切に設計されたソフトウェア・アーキテクチャを維持していく必要がある。新規開発時のソフトウェア・アーキテクチャ設計が適切に行われないことで、ソフトウェア・アーキテクチャが不明確で分かりづらいものとなり、派生開発が困難となる原因になる。そして、派生開発時の変更が適切に行われないことで、ソフトウェア・アーキテクチャが崩れ、ソフトウェアの構造が劣化して複雑になっていく原因となる。

これらの対応策を考える上で、新規開発のみで対応すべきものと新規、派生の両開発で対応すべきものに分類する。問題点の対応策としての課題は、現状、開発の現場でできていないことをできるようにするための対応策としての方法論を確立させ、開発の現場に導入して浸透させることである。以下の表に問題点の対応策としての課題と、その対象となる開発分類についてまとめた。

表 81-1 課題と対象となる開発分類

No.	問題点の対応策としての課題	新規開発	派生開発
1	ユーザが考えている機能の単位でソフトウェアを設計していく	○	－
2	トレーサビリティを確保できるようなソフトウェア開発を行う	○	○
3	適切なソフトウェア・アーキテクチャを中心に考えた開発を行う	○	○

［記号説明］○：該当、－：非該当

上述の課題を解決できれば、ソフトウェア開発の現場における問題点に対応でき、解消することができると考え、そのための手段としてモデル指向開発手法を考案するに至った。本開発手法は、これらの課題解決に向け、オブジェクト指向技術を基盤に過去に考案された数々の優れたメソドロジを要素技術として取り入れながら、開発現場の中で洗練させてきたものである。

2.2. 課題解決のための目標と着眼点

なぜモデル指向開発が課題解決の手段として有効なのか、その理由をここで明らかにしたい。モデル指向開発手法の適用によって提示した課題を解決していくための目標、および着眼点について以下に述べる。

課題 1：ユーザが考えている機能の単位でソフトウェアを設計していく

［目標］ユーザ機能の導出は、要件定義、または要件定義の前後で行われるが、ユーザ機能の単位でソフトウェアの構造や振る舞いを設計していくことは簡単ではなく、具体的な方法論が必要になる。従来では構造化分析やオブジェクト指向分析などの技法があるが、ユーザ機能単位の設計という面からは、いずれも一長一短があり、決定的ではないと認識している。そこで、ユーザ機能の導出から、ユーザ機能の単位を意識してソフトウェア設計へつないでいく方法論を確立させる。

［着眼点］ユーザ機能や製品機能は一般にフィーチャと呼ばれることが多い。ユーザ機能の導出は、要件定義の前段階である要求分析で行うか、要件定義の後に要件分析にて行うかのいずれかを考える。要求分析では、フィーチャ分析モデリングによってユーザ機能の導出を行う。要件分析では、要求仕様をもとに機能分析モデリングによってユーザ機能の導出を行う。ユーザ機能の導出後は、外部インタフェース仕様から、ユーザ機能単位にインタフェース仕様を決めて扱うイベントを洗い出し、ユーザ機能の振る舞いと必要なデータ構造を明確にしながら、責務に着目してソフトウェアの構造と振る舞いを設計できるようにする。

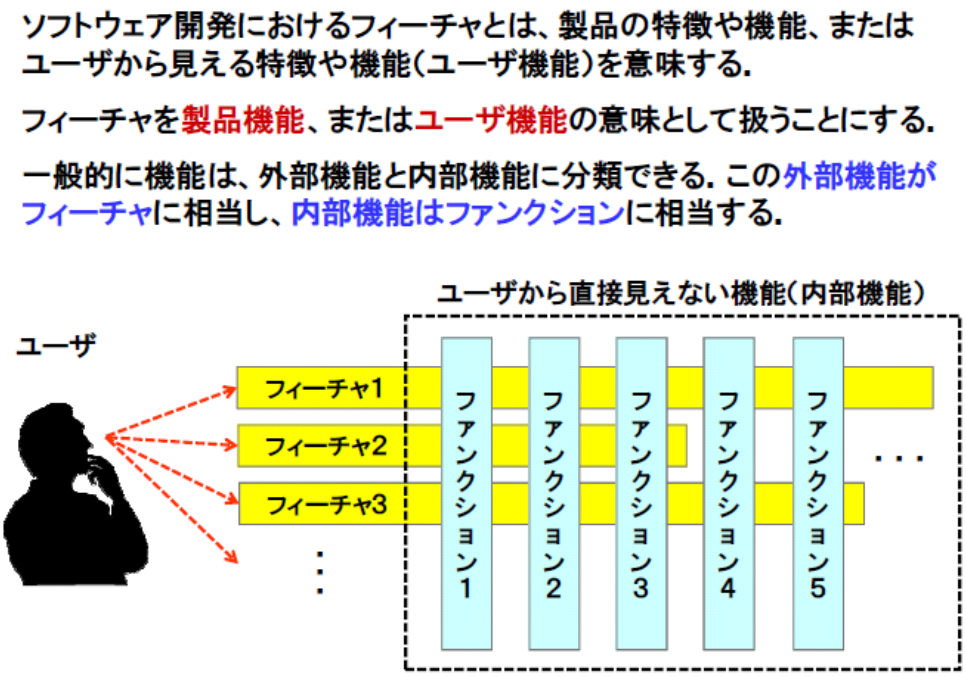


図 81-3 ユーザ機能としてのフィーチャの概念

図 81-3 にユーザ機能としてのフィーチャの概念を示す。ユーザが考える外部機能に相当するフィーチャは、開発者が考える内部機能に相当するファンクションに対して横串のように跨った関係となる。したがって、フィーチャの変更は、多くのファンクションの変更を誘導することになり、既存のソフトウェアが大きな影響を受けてしまう結果となる。モデル指向開発では、ファンクション単位のトップダウン的な「機能分割型アプローチ」と異なり、フィーチャ単位で機能を捉え、ボ

トムアップ的な「責務割当型アプローチ」と呼ぶ独自の視点を基盤とする。図 81-4 は、2 種類の開発アプローチについて、それらの概念と相違点を示している。

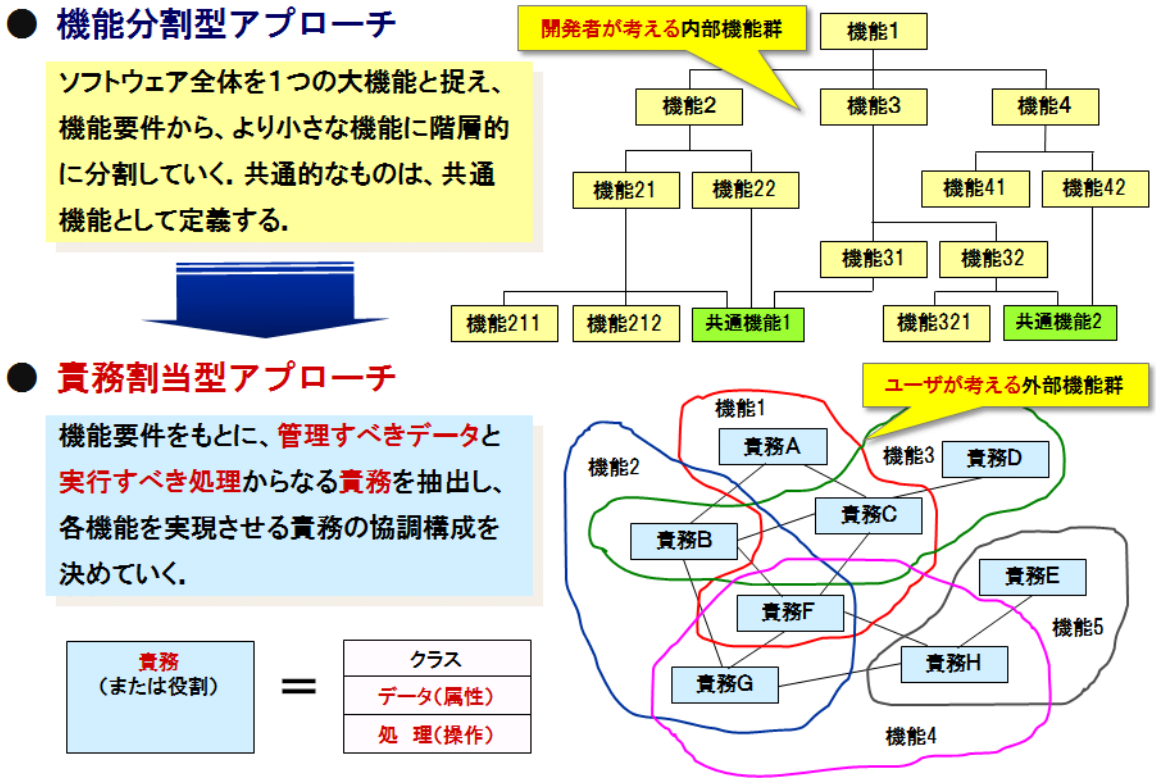


図 81-4 ユーザが考える外部機能からの開発アプローチの概念

課題 2：トレーサビリティを確保できるようなソフトウェア開発を行う

［**目標**］新規開発や派生開発のトレーサビリティを確保するためには、要件定義から実装までの各開発フェーズの成果物について、開発フェーズ間をつなげるキー情報が必要になる。開発成果物のキー情報の候補としては、要求仕様、機能仕様、設計仕様、実装仕様があるが、これらを相互につなげ、管理していくことは容易ではない。そこで、要件定義から実装までの開発成果物のキー情報を定義し、それらを論理的につなげ、管理していく方法論を確立させる。派生開発時には、派生開発プロセス手法 XDDP（eXtreme Derivative Development Process）で提唱している変更の差分に限定したコンパクトなトレーサビリティ管理を実現させる。

［**着眼点**］各開発フェーズにモデル技術を導入し、開発フェーズの成果物であるモデルをキー情報として要件定義から実装までのトレーサビリティを確保することを考える。モデルをキー情報とすることの利点は、文章では困難な、関連する全体像の把握が容易にできることである。要件定義の要求仕様とインタフェース仕様から、要求モデリングによりユースケース分析モデル、イベントケース分析モデル、ステートケース分析モデルなどの要件分析モデルにつなげる。さらに要件分析モデルから分析モデリングによりタスク、クラス、コンポーネント、シーケンス、ステ

ートマシンなどのアーキテクチャモデルにつなげる。アーキテクチャモデルとは、ソフトウェア全体の構造と振る舞いを表現するモデルであり、成果物としては、タスク関係図、タスク構造図、コンポーネント図、コンポジット構造図、クラス図、シーケンス図、およびステートマシン図などである。次にアーキテクチャモデルからクラス設計、コンポーネント設計、処理設計、状態遷移設計などの詳細設計モデルにつなげる。最後に詳細設計モデルからプロセス／スレッド、タイミング、関数・メソッド、プログラムなどの実装モデルにつなげる。各モデルには、トレーサビリティを確保するためのキー情報が定義される。

図 81-5 モデル指向開発の開発プロセスと成果物

開発プロセスの各フェーズで作成されるモデルは、トレースモデルとして作成するモデル・トレーサビリティマトリクスによってトレース可能となる。図 81-6 にモデル・トレーサビリティマトリクスの作成例を示す。

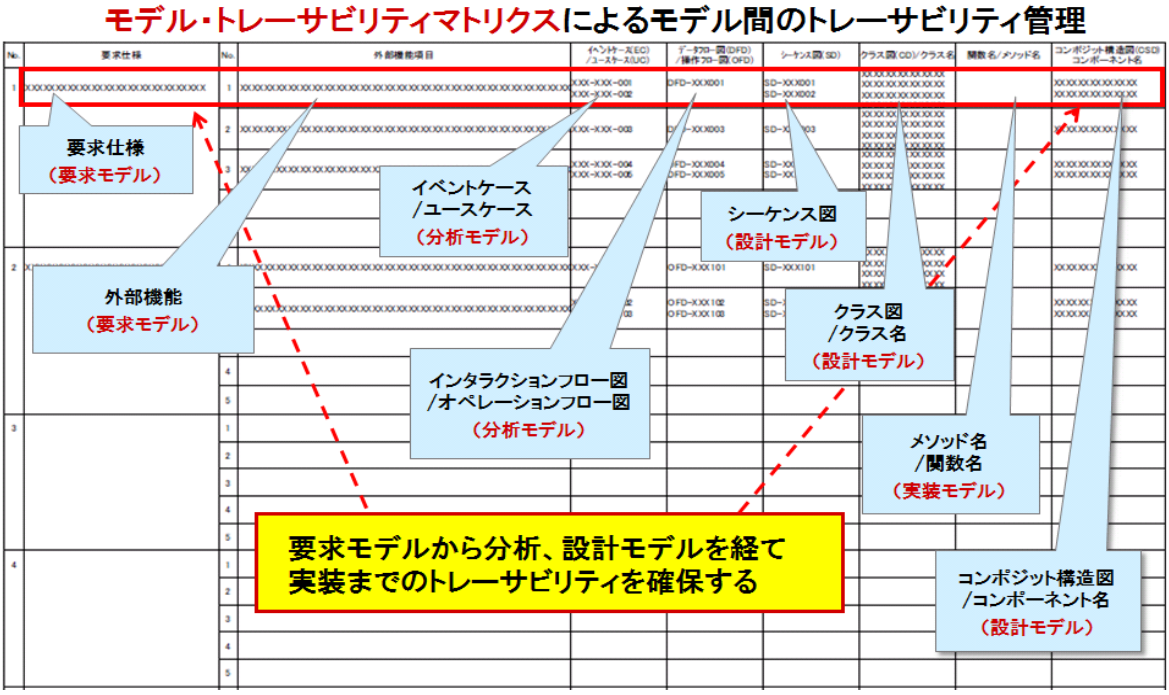


図 81-6 モデル・トレーサビリティマトリクスの作成例

各モデルで定義されるキー情報によって、要件定義から実装までの開発トレーサビリティが確保されるメカニズムについて、図 81-7 のパッケージ図に示す。パッケージは、開発フェーズを表現し、パッケージ中のクラスは、開発フェーズで作成する成果物（主にモデル）を表現している。また、クラス中に定義されている属性は、トレース対象となるキー情報を表現している。パッケージ間の依存関係《trace》は、各フェーズ間のトレース関係を表現し、クラス間の依存関係《trace》は、各成果物間のトレース関係を表現している。

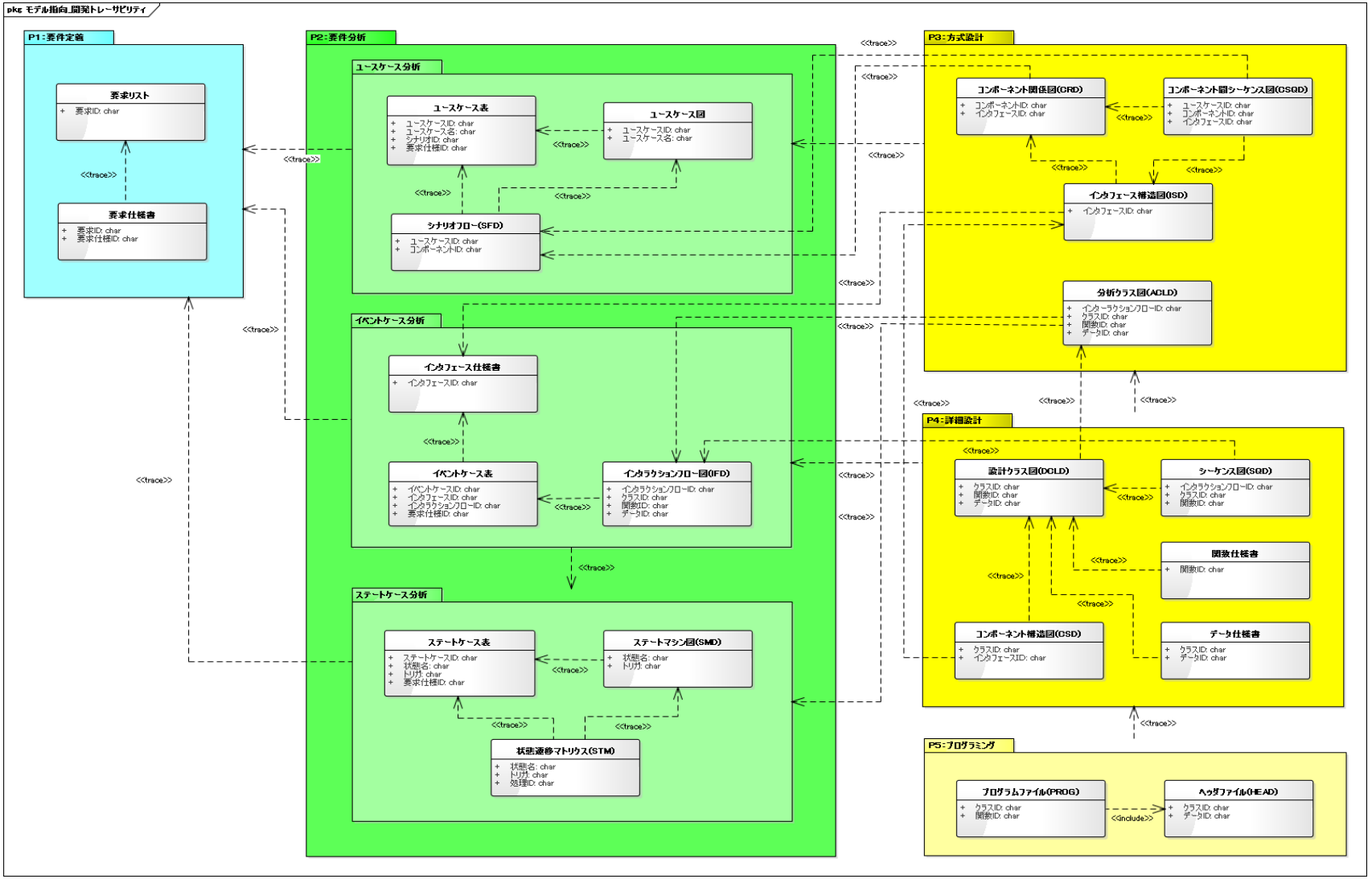


図 81-7 モデル指向開発の開発トレーサビリティ

課題 3：適切なソフトウェア・アーキテクチャを中心に考えた開発を行う

［**目標**］ソフトウェア・アーキテクチャは、ソフトウェアに対する要求仕様と外部とのインタフェース仕様から決定される。これらの仕様は主にソフトウェアの振る舞いに関するものであるが、ソフトウェア・アーキテクチャは構造と振る舞いの両面で決定されなければならない。また、その決定は要求仕様と外部とのインタフェース仕様を満たしていなければならないが、その根拠を設計レベルで示すことは難しい。そこで、新規開発ではアーキテクチャ決定の根拠を示すことができ、派生開発では既存のアーキテクチャを劣化させない方法論を確立させる。

［**着眼点**］新規開発では、要件定義の要求仕様と外部とのインタフェース仕様から要求モデリングによりアーキテクチャ決定の根拠となるユースケース分析モデル、イベントケース分析モデル、ステートケース分析モデルなどの要件分析モデルを作成する。この要件分析モデルから並列処理構造、責務構造、およびデータ構造を抽出して、ソフトウェアの実行環境となるタスク構造、ソフトウェアの構造としてのクラス、コンポーネント、振る舞いとしての処理シーケンス、ステートマシンなどのアーキテクチャモデルを作成し、アーキテクチャを決定していく。派生開発では、モデル指向開発導入済みの場合、変更要件定義の変更要求仕様と既存の要件分析モデルから差分分析モデリングによりユースケース差分分析モデル、イベントケース差分分析モデル、ステートケース差分分析モデルなどの差分分析モデルを作成する。この差分分析モデルと既存のアーキテクチャモデルから変更によるアーキテクチャへの影響範囲を特定し、アーキテクチャの劣化防止を行う。また、アーキテクチャが既に劣化している場合には、アーキテクチャ改善のために依存関係の局所化を行っていく。派生開発にモデル指向開発を新規導入する場合は、変更の差分に関するモデル（差分モデル）を作成して、既存の仕様書に追加していくことになる。図 81-9 と図 81-10 にモデル指向開発による派生開発プロセスの概要を示す。

モデル指向開発における派生開発時の依存関係の局所化に関するアプローチについて、図 81-8 に概要を示す。

変更範囲内の処理群と結合している変更範囲外の処理群について処理間の依存関係を局所化する.

依存関係としては以下の3つが重要であり、各々に分類して局所化する:

① データ共有関係　② 呼出し関係　③ クローン関係

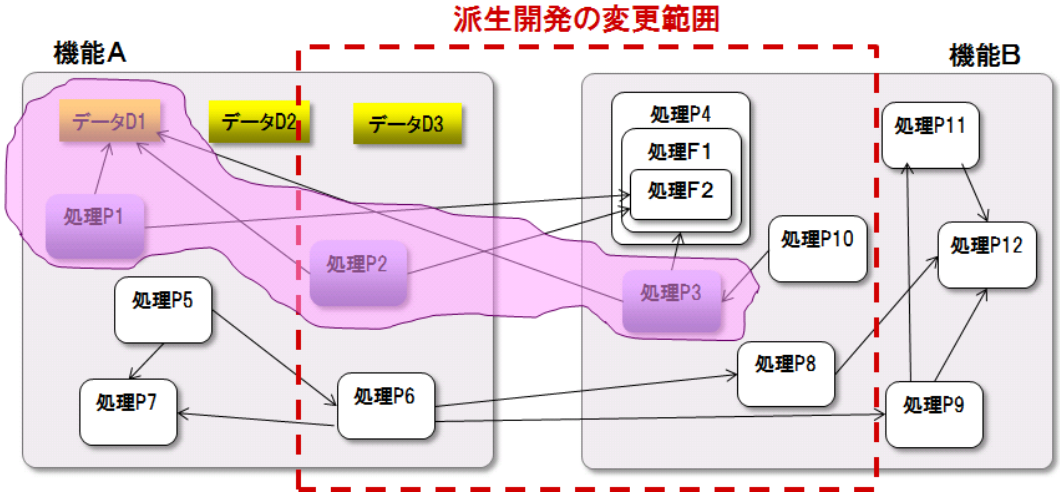


図 81-8 依存関係の局所化

モデル指向開発を派生開発へ導入する際の開発プロセスは、2 種類に分類して用意されている。1 つ目は、派生開発へのモデル指向開発適用のための開発プロセス（図 81-9 を参照）であり、2 つ目は、モデル指向開発への派生開発プロセス適用のための開発プロセス（図 81-10 を参照）である。

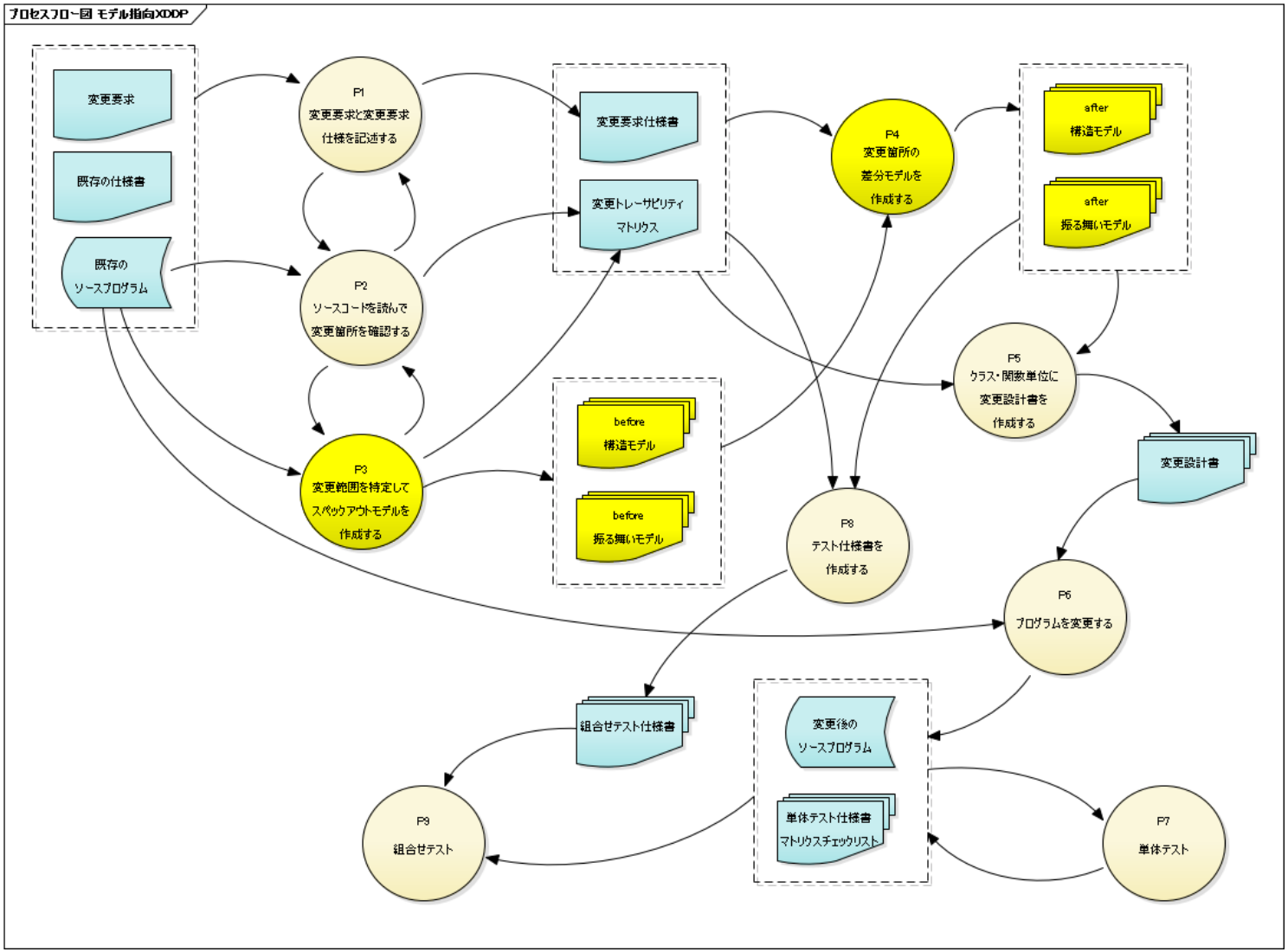


図 81-9 派生開発へのモデル指向開発適用プロセスフロー

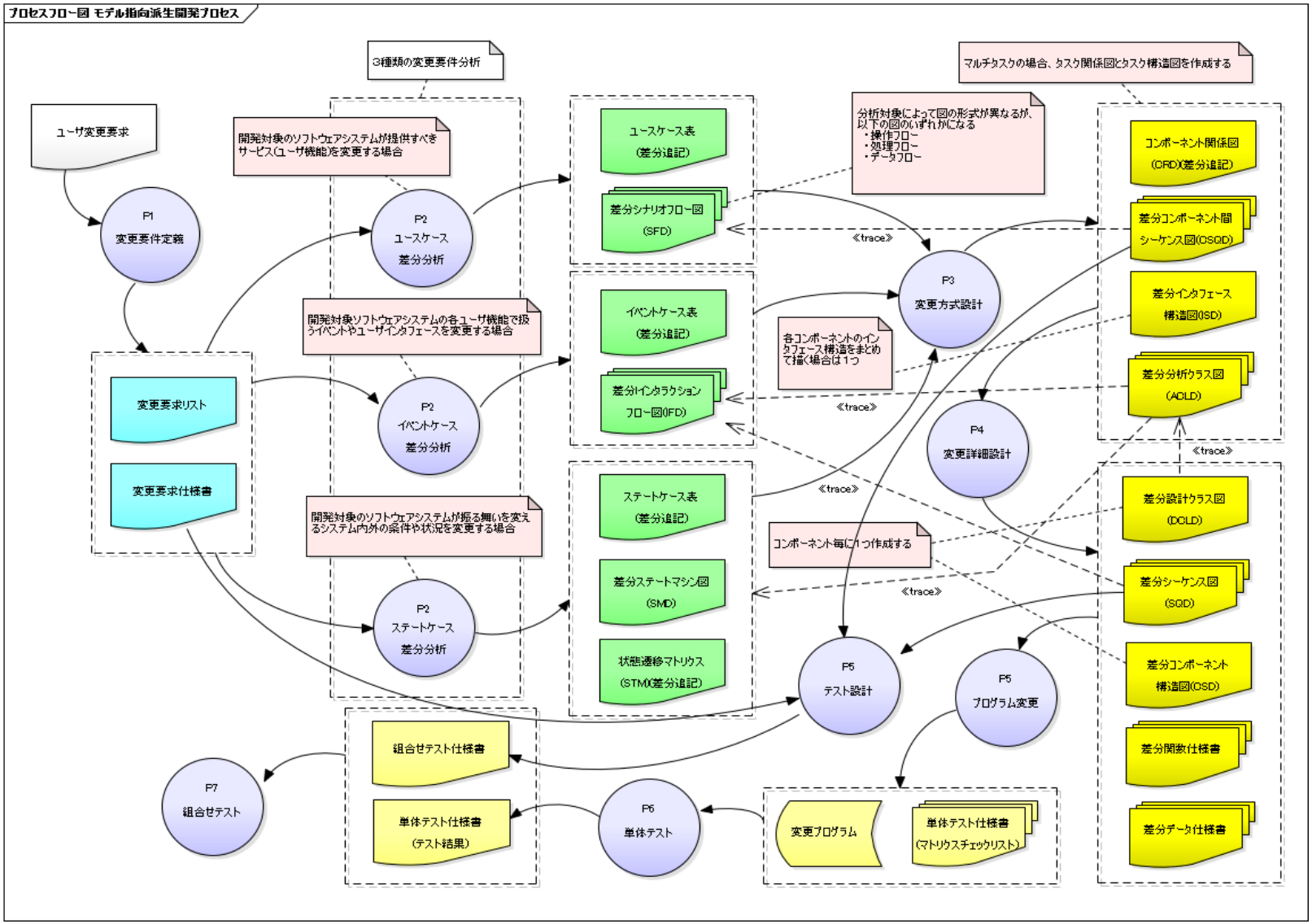


図 81-10 モデル指向開発の派生開発型適用プロセスフロー

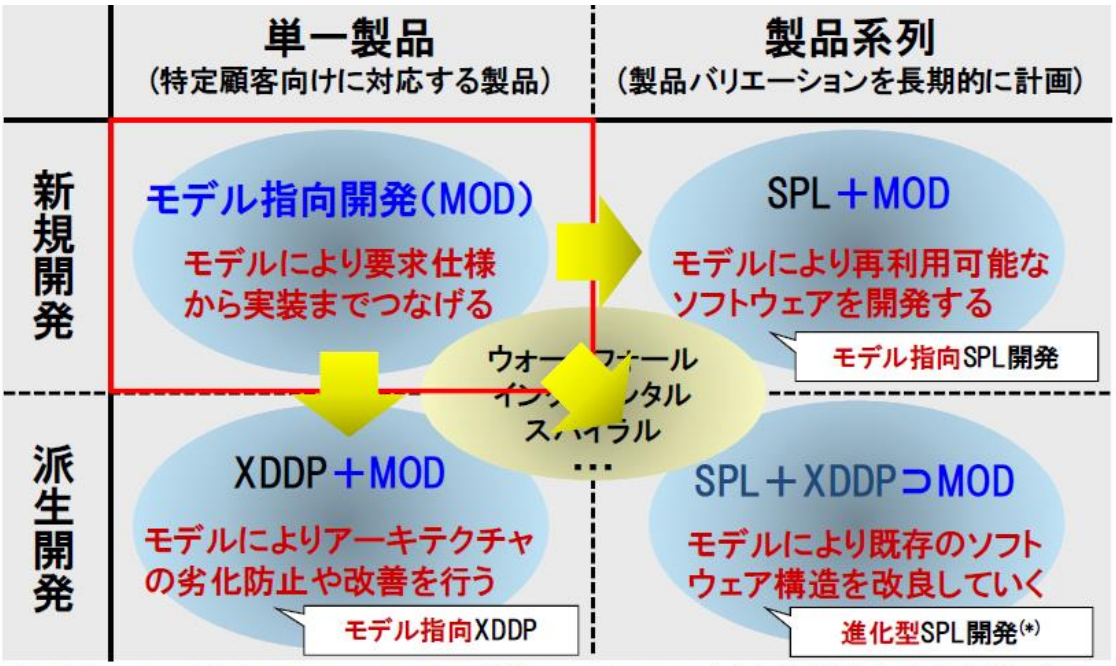
3. 適用手法の対象と要素技術・方法論

3.1. 適用対象の製品・プロジェクト

適用対象としては、機能追加や変更がある程度予測できる製品系列と、そうではない単一製品、ソフトウェア・アーキテクチャを作り上げる新規開発と稼働中の既存製品を改造したり、継承したりする派生開発がある。これらは各々開発における状況や観点が異なる。そこで、適用対象の製品として単一製品（特定顧客向けに対応する製品）と製品系列（製品バリエーションを長期的に計画）を持つ製品に分類し、適用対象のプロジェクトとして新規開発と派生開発に分類すると、以下の図81-11 のように 4 象限の適用対象となる。モデル指向開発の適用対象として最初のターゲットにしたのは、単一製品の新規開発であった。ここでモデル指向開発の基盤技術が確立された。次に適用対象のターゲットになったのは、製品系列の新規開発であった。

これは、ソフトウェアプロダクトライン開発をモデル指向開発と融合させた開発手法（モデル指向 SPL 開発）（SPL : Software Product Line）である。製品フィーチャ分析におけるフィーチャモデリングにより再利用可能な製品機能をコア資産として定義し、コア資産コンポーネントモデルを分析、設計していく方法論である。その後、最も難易度の高い適用対象である製品系列の派生開発にチャレンジして、その方法論を完成させた。これは、モデル指向 SPL 開発に派生開発プロセス手法 XDDP（eXtreme Derivative Development Process）を融合させ、派生開発の中で再利用可能なコア資産コンポーネントを段階的に開発していくソフトウェアプロダクトライン開発手法（進化型 SPL 開発）である。適用対象として最後のターゲットにしたのが単一製品の派生開発であった。これは、派生開発プロセス手法 XDDP にモデル指向開発を融合させた開発手法（モデル指向 XDDP）である。

以上の適用実績から、モデル指向開発について、分類した 4 象限の適用対象をすべてカバーできる開発手法として確立させることができた。



XDDP:eXtreme Derivative Development Process、(株)システムクリエイツ清水吉男氏が考案した派生開発プロセス
SPL:Software Product Line、MOD:Model Oriented Development (モデル指向開発手法)
(*)派生開発時にSPLの構造に徐々に近づけていく、当社が独自に研究・考案した手法

図 81-11 モデル指向開発の適用対象開発

本紙では、単一製品の新規開発を主な対象にして、モデル指向開発手法の方法論を展開している。モデル指向 XDDP 手法、モデル指向 SPL 開発手法、進化型 SPL 手法の方法論については、別な機会の紙面に展開を譲ることにしたい。

これまでのモデル指向開発の適用実績から、適用対象としては分野を限定せずに制御系システム、組込み系システム、および情報系システムの各分野にわたり適用可能であることが確認できている。適用実績のある分野について、制御系システム、組込み系システム、および情報系システムに分類して以下に示す（図 81-12 を参照)。

3.2. モデル指向開発の要素技術

モデル指向開発手法の中には、これまでに多くの効果や実績が示されている有効な要素技術が導入され、他の要素技術と融合され、応用されている。代表的な要素技術について、概要とモデル指向開発における活用のポイントを以下に示す。

(1) 要求の仕様化・表現技術 USDM（Universal Specification Describing Manner）技法

株式会社システムクリエイツの清水吉男氏が考案した、要求を仕様化して表現するための仕様表記技法である。本技法には、仕様の抜け漏れや誤り、仕様間の衝突（conflict)、勘違

いなどを防止するための具体的な考え方や工夫、および仕掛けが盛り込まれている。USDMの原則として、「仕様は要求の中の目的語、および動詞に存在する」という考え方がある。

(2)　システムズモデリング言語 SysML（Systems Modeling Language）技法

SysML は、非営利の標準化団体 OMG（Object Management Group）が 2007 年に標準仕様として公開したシステム開発向けのモデリング言語である。特徴としては、システム要求やハードウェアとソフトウェアの関係を表記可能であり、方法論やツールではない。SysML は、UML のサブセットに UML のプロファイル機構を使って拡張したものであり、システムズエンジニアリングの考え方に基づき、複数視点から要求やシステムの構造と振る舞いを表現可能な 9 種類のダイアグラムから構成されている。

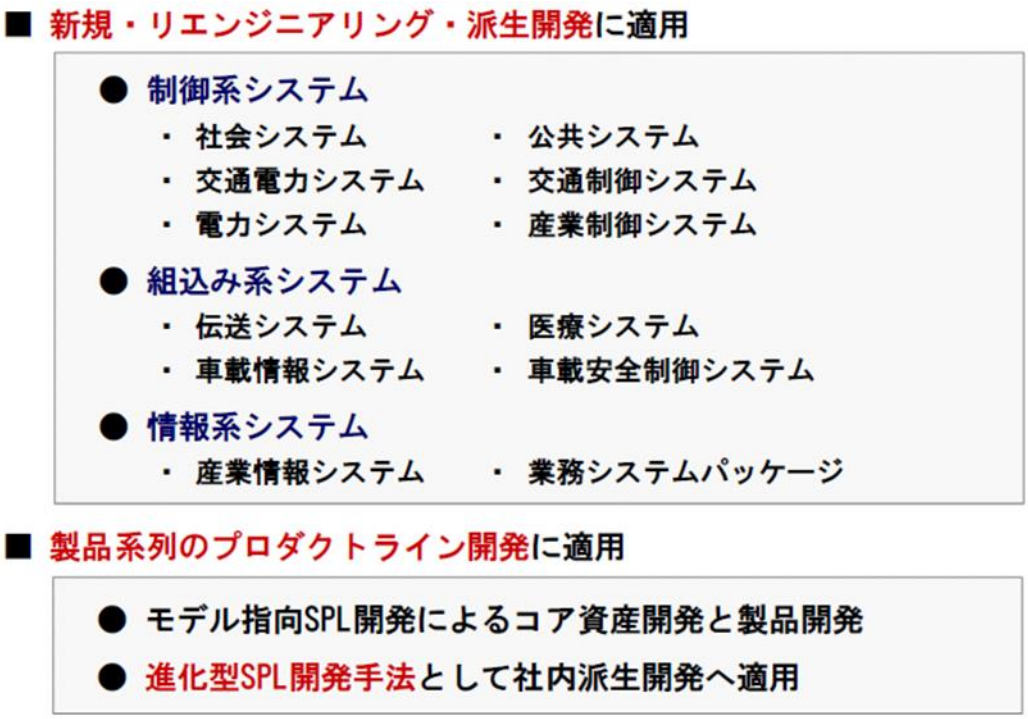


図 81-12 モデル指向開発の適用対象分野

(3)　構造化分析・設計手法

エドガー・ダイクストラによる構造化プログラミングから発展した構造化手法には、ソフトウェアエンジニアリングの基盤となる多くの重要な概念が存在する。仮想機械（virtual machine)による抽象化、モジュール性(modularity)、凝集度(cohesion)、結合度(coupling)、トップダウン・アプローチ、段階的詳細化、データ中心アプローチなどである。また、ソフトウェア検証の概念と課題に関する考察もエドガー・ダイクストラから始められているが、いまだに課題解決には至っていない。

(4)　オブジェクト指向分析・設計手法

Simula や Smalltalk などのオブジェクト指向プログラミングから始められたオブジェクト指向手法には、それまでに無かった重要な概念が多く存在する。カプセル化（encapsulation）と情報隠蔽（information hiding）、メッセージング、継承（inheritance）、多態性（polymorphism）などである。これらの優れた概念を深く理解して、開発現場に活かしていかなければならない。

(5)　責務割当パターン GRASP（General Responsibility Assignment Software Patterns）

クレーグ・ラーマンがオブジェクト指向設計でクラスやオブジェクトに責務を割り当てる際の基本原則を整理した 9 つのパターンである。パターンには、情報エキスパート（Information Expert）、クリエータ（Creator）、低結合度（Low Coupling）、高凝集度（High Cohesion）、コントローラ（Controller）、多態性（Polymorphism）、純粋架空物（Pure Fabrication）、間接化（Indirection）、バリエーション防護（Protected Variations）がある。これらは技法ではなく、ソフトウェア開発における課題解決のための設計に関する原則的な考え方である。

(6)　ロバストネス分析技法

イヴァー・ヤコブソンがオブジェクト指向開発方法論 OOSE（Object-Oriented Software Engineering）の中で提案したオブジェクト指向分析のためのアプローチである。本技法は、ユースケースモデルと概念モデルをインプットとして分析を行い、ロバストネス図を作成するものである。ユースケースモデルは、UML ユースケース図とユースケース記述ドキュメントからなるモデルである。概念モデルは、開発の対象となる業務における重要な概念と概念間の関係を識別するために作成する UML クラス図のモデルである。ロバストネス図とは、ユースケースから設計モデルを導出するための分析モデル図である。分析モデルの構造はクラス図で、振る舞いは相互作用図（シーケンス図、またはコラボレーション図）で表記される。ロバストネス図で表記されるクラスは、ステレオタイプ《boundary》で拡張した「バウンダリ」、ステレオタイプ《control》で拡張した「コントロール」、ステレオタイプ《entity》で拡張した「エンティティ」の 3 種類である。これらのクラスは設計要素の候補となり、ロバストネス図はソフトウェア設計のインプットとなる。

(7)　ソフトウェアモデリング言語 UML（Unified Modeling Language）技法

UML は、スリーアミーゴスと呼ばれている、グラディ・ブーチ、イヴァー・ヤコブソン、ジェームズ・ランボーの 3 人の技術者が策定し、非営利の標準化団体 OMG（Object Management Group）が標準化したオブジェクト指向開発向けの統一モデリング言語である。その後、国際規格化（ISO/IEC 19501）され、2009 年には日本工業規格化（JIS X 4170）された。UML は、オブジェクト指向の考え方に基づき、複数視点から構造と振る舞いを表現可能な 13 種類のダイアグラムから構成されている。

(8)　フィーチャ駆動開発手法

ジェフ デ・ルーカが考案した、別名ユーザ機能駆動開発は、アジャイルソフトウェア開発手法の 1 つであり、反復型ソフトウェア開発プロセスの一種でもある。その主な目的は、実際に動作するソフトウェアを繰り返し、適切な間隔でユーザに提供することである。本手法では、開発をカバーする 5 つのプロセスを設定している。それらは、全体モデルの作成とユーザ機能のリストアップ、ユーザ機能単位の計画立案、ユーザ機能単位の設計、およびユーザ機能単位の構築である。最初のプロセスは、ピーター・コードのオブジェクトモデル技法に大きな影響を受けている。2 番目のプロセスは、ユーザ機能単位の要求仕様と開発作業を管理するためのピーター・コードのユーザ機能リストの考え方を取り入れている。他のプロセスと全体としてのプロセスの統一性は、ジェフ デ・ルーカの開発経験に基づいている。

(9)　ドメイン駆動開発手法

エリック・エヴァンスの考案したドメイン駆動設計によるソフトウェア開発手法である。「ドメイン」とは、アプリケーションが対象とする業務領域のことであり、本手法では、ドメインを「知識、影響力、活動の一領域」と定義している。一般に業務アプリケーションのアーキテクチャは、プレゼンテーション層、ドメイン層、データソース層の 3 層に分けることができ、ユーザ業務に直接的に関わる部分となるドメイン層を正しく構築することが重要になる。ポイントは、ドメインモデルをソフトウェア開発の中心にすえ、コードやコミュニケーションをドメインモデルと一体化させながら、ドメインモデルを反復的に深化させることで、より価値の高いアプリケーションを生み出していこうとする考え方にある。そのため、ドメインモデルをドメイン知識者であるユーザと開発者間の共通言語とし、反復的な開発でドメイン知識を深めながら、ドメインモデルと実装コードが双方向に対応付けられるようにすることが必要となる。

(10) 責務駆動設計技法

オブジェクト指向ソフトウェア設計で最も難解な課題は適切なオブジェクトの特定である。レベッカ・ワーフスブラックが考案した責務駆動設計は、そのような背景から生まれたものであり、優秀なオブジェクト設計者が行ってきた経験に基づき、一連のプラクティス、テクニック、ガイドラインとして反映させた強力な設計技法である。その中で、オブジェクトは交換可能であるロール（責務の集合）の集合を具現化したものであると定義し、オブジェクトのロールステレオタイプとして、情報保持役(Information Holder)、構造化役(Structurer)、サービス提供役(Service Provider)、調整役(Coordinator)、制御役(Controller)、インタフェース役(Interfacer)の 6 つに分類している。そしてアプリケーションは、複数の責務からなるシステムを実装したものであり、責務はロールに割り当てられ、ロールは責務を果たすためにコラボレートするとして、アプリケーションは協調するオブジェクトの共同体であると考える。

（11）インタフェース指向設計技法

構造化手法の中で考え出されたモジュール性に関する凝集度と結合度は、開発手法によって解決できるものではないソフトウェア構造の本質的な概念であることが認識され、それはオブジェクト指向においても引き継がれている。バートランド・メイヤーが提唱した「契約による設計」は、インタフェースがオブジェクトやコンポーネントの構造について、高凝集度、低結合度を実現させる強力な手段の１つであることを示している。本技法は、そこに着目した設計アプローチである。

（12）派生開発プロセス手法 XDDP（eXtreme Derivative Development Process）

株式会社システムクリエイツの清水吉男氏が考案した、派生開発向けに特化した開発プロセスである。多くの企業では、派生開発を「新規開発崩し」のプロセス（清水吉男氏が呼んでいる新規開発と同様のプロセス）で対応している。本手法は、「新規開発崩し」のプロセスが派生開発の特性を無視した非効率的なものであり、品質低下を招いている原因であることに気付かせてくれる。XDDP は、変更要求仕様書（What と Why の観点）、変更 TM（Traceability Matrix）（Where の観点）、および変更設計書（How の観点）を主な成果物とする「差分」（before と after）を意識した無駄の無いコンパクトな開発プロセスである。XDDP における重要な概念として、部分理解の罠、スペックアウト、変更要求と変更仕様の特定（Specify）、影響箇所への気付き、PFD（Process Flow Diagram）がある。

3.3. モデル指向開発の方法論

ソフトウェア開発方法論の立場から、モデル指向開発手法がこれまでの開発方法論と異なる点を中心として、開発プロセスのフェーズ別に開発現場における有効性の根拠について述べる（図 81-5 モデル指向開発の開発プロセスと成果物を参照）。

（1） ドメインモデリング（ドメイン分析）フェーズ

ユーザ要求、または製品ニーズなど、ソフトウェアシステムに求められる要求事項を導出するための分析、およびソフトウェアシステムが要求事項を実現させる手段としての要求仕様（別名、略して要件と呼ぶ）や外部とのインタフェース仕様（外部インタフェースと呼ぶ）を定義する過程である。

① 要求事項導出

ユーザである顧客からの要求事項が未確定の場合は、顧客と共にドメイン分析を行い、顧客の業務ドメインをモデル化する。業務ドメインの振る舞いモデルには、ワークフロー図や BPMN（Business Process Modeling Notation）によるビジネスフロー図を活用する。また、業務ドメインの構造モデルには、アーキタイプクラスによるドメインクラス図を活用する。現状の業務ドメインの構造と振る舞い（As-Is のモデル）から、今後どのような業務ドメイ

ンの構造と振る舞いに変えていきたいのか可視化（To-Be のモデル）した上でソフトウェアシステムについての要求事項を導出する。

② 要求仕様（要件）定義

ユーザからの要求事項について、各々理由を理解し、分類、整理して要求間のコンフリクトやアンマッチが無いことを確認した上で、要求リストの形式にまとめる。次に各々の要求事項を実現させるソフトウェアシステムの振る舞いを要求仕様（要件）として具体的に記述する。要求事項と要件の表記には、USDM（Universal Specification Describing Manner）を活用する。この表記方法により、要求事項と要件を分類、階層化し、構造的に記述でき、抜け漏れや勘違いなどの誤りに気付きやすくなる。また、要求事項が大規模で複雑な場合には、UDDM 表記でまとめる前段階として要求間と要件間、および要求と要件間の関係を整理する目的で、SysML（Systems Modeling Language）の要求図を活用する。

③ 外部インタフェース仕様定義

要求仕様（要件）を定義する際に、ソフトウェアシステムが外部と相互作用するために必要となるインタフェース仕様（外部インタフェース仕様）について定義する。インタフェース仕様は、オペレータを対象にしたマンマシン・インタフェース（別名、ユーザインタフェース）仕様、他のコンピュータや装置を対象とした通信インタフェース仕様、他のソフトウェアシステムを対象としたプログラム・インタフェース仕様に分類できる。マンマシン・インタフェース仕様では、画面の種類や画面入出力フォーマット、および画面遷移などに関する仕様になる。通信インタフェース仕様では、通信メッセージの種類やメッセージフォーマット、および通信シーケンスなどに関する仕様になる。プログラム・インタフェース仕様では、呼出す関数の種類や入出力パラメータ、および戻り値などに関する仕様になる。

(2)　要求モデリング（要件分析）フェーズ

定義された要求仕様（要件)、および外部インタフェース仕様における振る舞いを複数の視点から詳細に分析することで、ソフトウェアの振る舞いを決定し、振る舞いに必要となるソフトウェアの責務構造を抽出して、ソフトウェア・アーキテクチャの設計につなげる過程である。

① ユースケース分析（ユーザの使い方の視点で分析)

オペレータを対象としたマンマシン・インタフェースを持つ場合、ユースケース分析が有効な分析方法となる。分析手順としては、まずソフトウェアシステムの使い方としてのユースケースを漏れなく洗い出して、ユースケース表（ユースケース図に表記する情報をすべて表形式で記述したもの）にまとめる。次に要求仕様（要件）とマンマシン・インタフェース仕様である画面の種類や画面入出力フォーマット、および画面遷移などに関する仕様を基にして、ユースケース単位にオペレーションフロー図を作成する。オペレーションフロー図は、

UML アクティビティ図を拡張したモデル指向開発手法における分析用ダイアグラムである。

② イベントケース分析（外部との相互作用の視点で分析）

マンマシン・インタフェース、通信インタフェース、およびプログラム・インタフェースなど、すべてのインタフェースについてイベントを通じての相互作用として分析する。すなわち、マンマシン・インタフェースの場合は、画面表示を出力イベント、操作を入力イベントとして捉える。通信インタフェースの場合は、受信メッセージを入力イベント、送信メッセージを出力イベントとして捉える。プログラム・インタフェースの場合は、他プログラムからの呼出しを入力イベント、呼出し元プログラムへのリターンを出力イベントとして捉える。分析手順としては、まず、扱う入力イベント、出力イベント、および入出力イベントを漏れなく洗い出して、イベントケース表にまとめる。次に要求仕様（要件）とインタフェース仕様を基にして、入力イベント、入出力イベントの単位にインタラクションフロー図を作成する。インタラクションフロー図は、UML アクティビティ図を拡張したモデル指向開発手法における分析用ダイアグラムである。

③ ステートケース分析（相互作用のための動作条件の視点で分析）

ソフトウェアシステムが相互作用行う対象の状態、またはソフトウェアシステム自身の状態によって、扱うイベントに対する振る舞いを変えなければならない場合がある。その際に必要となるのが、管理すべき状態と状態を変えるトリガとなるイベントについての分析である。分析手順としては、まず、要求仕様（要件）とインタフェース仕様を基にして、管理すべき状態と状態を変えるトリガとなるイベントを漏れなく洗い出し、ステートケース表にまとめる。次に状態と状態遷移、および状態遷移のトリガ、ガード条件を基にして、UML ステートマシン図を作成する。最後にステートマシン図の網羅性を確認する目的で、状態遷移マトリクスを作成する。ステートケース分析の成果は、イベントケース分析のインタラクションフロー図作成時にイベント単位の振る舞い表記の中で状態管理振る舞いとして反映される。

(3)　分析モデリング（方式設計）フェーズ

責務構造を詳細に分析して、ソフトウェアの動作条件（事前条件、事後条件、不変条件)、実行環境（タスク構造、割込構造)、およびデータ構造（テーブル、ファイル）と処理構造（関数構成）を決定し、インタフェース仕様に沿って要求仕様を実現するソフトウェア・アーキテクチャを設計する過程である。

① 責務抽出と責務分類

入力イベント、入出力イベントの単位に作成するインタラクションフロー図が完成後、そこから責務（役割）の観点で関数群とその入出力データである一時データ群を括り出し（カプセル化する)、責務としての名称を付けて分類する。責務の分類は、「バウンダリ」(境界)、

「コントロール」(制御)、「エンティティ」(実体) の 3 種類のいずれかとする。具体的には、イベント入出力の責務は「バウンダリ」、永続データの責務は「エンティティ」、内部データ編集や制御ロジックの責務は「コントロール」として分類する。

② タスク設計

要件とインタフェース仕様に基づき、ソフトウェアシステムの振る舞いに並列性の要求があるか否かを分析し、動作環境であるタスク構造を決定する。ユースケース分析で作成するオペレーションフロー図やイベントケース分析で作成するインタラクションフロー図、およびステートケース分析で作成するステートマシン図が具体的な分析の対象となる。タスクが複数必要となる場合は、タスク構成を具体化するためにコンポーネント図を拡張したタスク関係図を作成する。

③ コンポーネント設計

ユーザ機能や製品機能、および再利用の単位を独立した責務として捉え、ソフトウェアコンポーネントに割り当てる。モデル指向開発におけるソフトウェアコンポーネントとは、カプセル化と情報隠蔽による低い結合度と抽象化したインタフェース仕様による高い凝集度を持ち、交換可能で、かつ再利用可能なソフトウェアのことである。一般に呼ばれているコンポーネントと概念的には同じであるが、ソフトウェア設計上の厳密な定義として前述の要件を満たさなければならない。

④ クラスへの責務割当

各インタラクションフロー図にて抽出した責務をクラスに割り当て、インタラクションフロー図単位に分析クラス図を作成する。責務に含まれている関数群はクラスの操作群、責務に含まれている一時データ群はクラスの属性群として割り当てる。割り当てた操作については、可視性、パラメータ、戻り値を定義し、割り当てた属性については、可視性、タイプ、デフォルト値を定義する。責務に付けた名称は、そのままクラス名とし、責務の分類は、そのままクラスの分類とする。クラス間の関連は、インタラクションフロー図における責務間の関係から、責務の分類を考慮して決定する。クラス間において、バウンダリ同士、エンティティ同士、およびバウンダリとエンティティ間では、関連を持つことを避けなければならない。

⑤ データ構造分析

インタラクションフロー図の振る舞いに必要となる永続データについて、データ構造を定義する。永続データには、データベースのテーブル、ファイル、永続的なメモリデータなどがある。ここでは、永続データとして必要なデータ項目を洗い出し、テーブル設計、ファイル設計などのデータ仕様設計につなげる。

(4)　設計モデリング（詳細設計）フェーズ

コンポーネント、クラス、関数について、各々インタフェースと内部構造（データと処理の構造）を決定して、ソフトウェアの構造と振る舞いの詳細仕様を設計する過程である。

① クラス設計

インタラクションフロー図単位に作成済みの分析クラス図を基にして、コンポーネント、またはタスクの単位で設計クラス図を作成する。言い換えると、複数の分析クラス構成から一つの設計クラス構成にまとめる。同じ責務を割り当てられた複数の分析クラスについては、同一クラスにまとめ、クラス内の属性や操作を統合する。

② コンポーネント構造設計

コンポーネント設計にてユーザ機能や製品機能、および再利用の単位として責務が割り当てられたコンポーネントについて、インタフェースと内部構造を設計する。インタフェースについては、インタフェースクラスとして必要となる属性（内部変数）と操作（インタフェース関数）を定義する。内部構造については、クラス設計にてコンポーネント単位に設計されたクラス構成を割り当てる。

③ タスク構造設計

タスクが複数必要となる場合、クラス設計、コンポーネント設計の結果を基にして、クラスやコンポーネントの動作環境となる各タスクの構造を割り当てる。

④ 関数構造設計

クラス設計にてクラス内の操作として可視性、パラメータ、戻り値を定義した関数の処理構造（処理ロジック）と、クラス内の属性として可視性、タイプ、デフォルト値を定義した内部データのデータ構造を設計する。

⑤ クラス間処理シーケンス設計

クラス設計にて確定した設計クラスについて、イベントケース分析で作成したインタラクションフロー図の単位にシーケンス図を作成することで、クラス間の処理シーケンスを設計し、クラスの構造と振る舞いに関する論理的妥当性をシミュレーションする。ただし、コンポーネントとして括られた設計クラス構成については、インタフェースの呼出しイベントから開始されるコンポーネント内シーケンス図を作成することで、クラス間処理シーケンスを設計する。

⑥ コンポーネント間処理シーケンス設計

コンポーネント設計にてユーザ機能や製品機能、および再利用の単位として責務が割り当てられたコンポーネントについては、イベントケース分析で作成したインタラクションフロ

一図の該当するフローについて、コンポーネントをブラックボックスとしたコンポーネント外接続のシーケンス図を作成することで、コンポーネント間処理シーケンスを設計し、コンポーネントの構造と振る舞いに関する論理的妥当性をシミュレーションする。

⑦ データ仕様設計

データ構造分析で定義した、データベースのテーブル、ファイル、永続的なメモリデータについて、データベースのテーブル構成、ファイル構成、メモリデータ構成などの詳細なデータ仕様を設計する。

(5) 実装モデリング（実装設計）フェーズ

ソフトウェアをプログラムとして実装する上での動作条件（タイミングや状態)、実行環境（プロセスやスレッド)、プログラムの処理や構造を確定させる過程である。

① タイミング実装設計

イベントケース分析のインタラクションフロー図にて分析し、処理シーケンス設計にて設計した割込み処理や状態遷移処理について、実装するためのタイミングを設計する。

② 実行環境設計

タスクが複数必要となる場合は、タスクを割り当てる実装形態（スレッド、またはプロセス）を決定し、スレッド間、またはプロセス間の通信仕様を設計する。

③ プログラミング

各詳細設計に基づいて、コンポーネント、クラス、関数、およびデータに関する仕様をプログラミングする。

3.4. 課題解決の展開

前述の 2．1 で提示した 3 つの課題に対して、モデル指向開発手法の適用によって、2．2 で提示した課題解決のための目標と着眼点に基づき、どのように課題を解決していったのか、具体的な展開について説明する。

課題 1：ユーザが考えている機能の単位でソフトウェアを設計していく

［着眼点］ユーザ機能や製品機能は一般にフィーチャと呼ばれることが多い。ユーザ機能の導出は、要件定義の前段階である要求分析で行うか、要件定義の後に要件分析にて行うかのいずれかを考える。要求分析では、フィーチャ分析モデリングによってユーザ機能の導出を行う。要件分

析では、要求仕様をもとに機能分析モデリングによってユーザ機能の導出を行う。ユーザ機能の導出後は、外部インタフェース仕様から、ユーザ機能単位にインタフェース仕様を決めて扱うイベントを洗い出し、ユーザ機能の振る舞いと必要なデータ構造を明確にしながら、責務に着目してソフトウェアの構造と振る舞いを設計できるようにする。

［展開］ユーザからの要求事項を基にして、ユーザが考えている機能（フィーチャ）の単位に要求仕様をまとめ、定義した要求仕様からフィーチャ間におけるインタフェースの独立性について分析することにより、フィーチャ単位のコンポーネント設計の可能性を追求していく。具体的には、以下のアクティビティにより進めていく。

(1)　要求分析

ユーザが実現させたいと考えている要求を洗い出して整理し、要求事項としてまとめた上で、それらを実現させる手段としてのフィーチャを明確にする。そのため、ユーザが実現させたい業務や運用に関係している要素の構造とそれらの振る舞いをモデルにして可視化する。構造を分析して可視化するモデルとしては、ドメインクラス図を活用する。業務や運用の流れを分析して可視化するモデルとしては、ワークフロー図を活用する。

(2)　要件定義

ユーザからの要求事項とフィーチャとの関係が明確になっていれば、フィーチャ単位での要求事項とフィーチャをまたがる要求事項に分類した上で要求仕様（要件）を定義していく。要求事項と要件の表記には、要求事項や要件の抜け漏れや誤りに気付きやすい USDM 表記法を活用する。要求事項と要件はソフトウェア開発にとってのスタートラインであり、ここからソフトウェアの設計によって要件を実現するソフトウェアの実装へとつなげていかなければならない。性能や信頼性、機密性、操作性などの非機能要件についても該当するフィーチャとの関係を明確にした上でフィーチャへの要件として仕様化し定義する。また、要件と共にシステムが外部と相互作用するために必要となるインタフェース仕様も定義する。

(3)　要件分析

フィーチャ単位にソフトウェアの設計につなげていくため、定義した要件を 3 つの視点で分析する。開発対象のソフトウェアがマンマシン・インタフェースを持つ場合、ユースケース分析を行い、オペレーションフロー図とインタラクションフロー図によりユースケース毎の振る舞いについて詳細化する。マンマシン・インタフェースを持たない場合は、外部と接続しているインタフェースにおいて扱う入出力イベントとシステム内部で扱う入出力イベントを対象にイベントケース分析を行い、インタラクションフロー図によりイベント毎の振る舞いを詳細化する。他にシステム内部の状態を管理する必要がある場合には、管理すべき状態を対象にステート分析を行い、ステートマシン図と状態遷移マトリクスにより状態遷移の振る舞いを詳細化する。

（4） 方式設計

要件定義と要件分析の成果から、コンポーネントの単位とコンポーネントをカプセル化し情報隠蔽して外部と相互作用を行うためのインタフェース仕様を定義する。コンポーネントは、要件定義で確定できるフィーチャ単位のものと要件分析で抽出できる再利用可能な単位のものの 2 種類ある。モデル指向開発では、前者を外部コンポーネント、後者を内部コンポーネントと呼ぶ。外部コンポーネントのインタフェース仕様は、要件と共に定義したインタフェース仕様に基づいて定義する。内部コンポーネントのインタフェース仕様は、要件分析にて抽出した内部コンポーネントの責務と関係をもつ責務との相互作用に必要な接続仕様に基づいて定義する。このようにして、フィーチャをコンポーネントという独立させた括りで捉えることで、フィーチャ間の結合度を極力低くすることができる。

課題 2：トレーサビリティを確保できるようなソフトウェア開発を行う

［着眼点］各開発フェーズにモデル技術を導入し、開発フェーズの成果物であるモデルをキー情報として要件定義から実装までのトレーサビリティを確保することを考える。要件定義の要求仕様とインタフェース仕様から、要求モデリングによりユースケース分析モデル、イベントケース分析モデル、ステートケース分析モデルなどの要件分析モデルにつなげる。さらに要件分析モデルから分析モデリングによりタスク、クラス、コンポーネント、シーケンス、ステートマシンなどのアーキテクチャモデルにつなげる。次にアーキテクチャモデルからクラス設計、コンポーネント設計、処理設計、状態遷移設計などの詳細設計モデルにつなげる。最後に詳細設計モデルからプロセス／スレッド、タイミング、関数・メソッド、プログラムなどの実装モデルにつなげる。各モデルには、トレーサビリティを確保するためのキー情報が定義される。

［展開］各開発フェーズにおけるモデルを中心とした成果物の要素をキーとして、成果物間のトレーサビリティを確保することで、開発全体のトレーサビリティを確保していく。具体的には、以下の要素をキー情報として相互にトレースできるようにしていく（図 81-6 を参照)。

（1） 要件定義

要件定義においてトレースの対象となるのは、下記の項目である。

・要求：ユーザである顧客から発せられる情報
・要件（要求仕様）：要求を実現する手段としてのソフトウェア仕様

他に要件と共に定義しなければならないものとして、ソフトウェアシステムが外部と相互作用するために必要なインタフェース仕様がある。例えば、マンマシン・インタフェース仕様、通信インタフェース仕様、およびプログラム・インタフェース仕様などである。その際に定義したインタフェース仕様項目も各々トレースの対象となる。

(2)　要件分析

要件分析においてトレースの対象となるのは、下記の項目である。

① ユースケース分析

・ユースケース：ソフトウェアの使い方のパターン

・シナリオフロー：ユースケースの詳細な振る舞い

② イベントケース分析

・イベントケース：外部との相互作用のパターン

・インタラクションフロー：イベントケースの詳細な振る舞い

③ ステートケース分析

・ステートケース：ソフトウェアの振る舞いの前提となる状態

・トリガ：状態が別な状態に変化する契機となるイベント

・処理（関数）：ある状態で実行されるべき振る舞い

(3)　方式設計

方式設計においてトレースの対象となるのは、下記の項目である。

・コンポーネント：ユーザ機能、製品機能、または再利用単位

・インタフェース（コンポーネントインタフェース）：コンポーネントの接続機構

・分析クラス：インタラクションフロー単位のクラス構造

・関数（メソッド）：単一目的のソフトウェア最小単位

・データ：モジュール構造に必要な内部データ構造

(4)　詳細設計

詳細設計においてトレースの対象となるのは、下記の項目である。

・設計クラス：ソフトウェアシステムに唯一のクラス構造

・関数（メソッド）仕様：関数（メソッド）の詳細な振る舞い

・データ仕様：データの詳細な構造

(5)　実装設計

実装設計においてトレースの対象となるのは、下記の項目である。

・プログラムファイル：　クラス、関数（メソッド）の実装

・ヘッダファイル：　データ構造の実装

課題 3：適切なソフトウェア・アーキテクチャを中心に考えた開発を行う

［着眼点］新規開発では、要件定義の要求仕様と外部とのインタフェース仕様から要求モデリン

グによりアーキテクチャ決定の根拠となるユースケース分析モデル、イベントケース分析モデル、ステートケース分析モデルなどの要件分析モデルを作成する。この要件分析モデルから並列処理構造、責務構造、およびデータ構造を抽出して、ソフトウェアの実行環境となるタスク構造、ソフトウェアの構造としてのクラス、コンポーネント、振る舞いとしての処理シーケンス、ステートマシンなどのアーキテクチャモデルを作成し、アーキテクチャを決定していく。派生開発では、変更要件定義の変更要求仕様と既存の要件分析モデルから差分分析モデリングによりユースケース差分分析モデル、イベントケース差分分析モデル、ステートケース差分分析モデルなどの差分分析モデルを作成する。この差分分析モデルと既存のアーキテクチャモデルから変更によるアーキテクチャへの影響範囲を特定し、アーキテクチャの劣化防止を行う。また、アーキテクチャが既に劣化している場合には、アーキテクチャ改善のために依存関係の局所化を行っていく。

［展開］ソフトウェア・アーキテクチャは、本来要求仕様を根拠として設計されるものであるが、その具体的な方法や手順は確立されていない。そこで、開発現場での実証を行いながら、要求仕様とソフトウェア・アーキテクチャをつなぐための具体的な分析方法と手順を考えた。以下に分析方法と手順を示す。

(1)　コンポーネント設計

モデル指向開発におけるコンポーネントの分析方法には、いくつかの種類がある。ここでは、ユーザ機能の分析であるユースケース分析について述べる。他には、製品機能の分析であるフィーチャ分析や再利用単位の分析であるソフトウェア共通化分析がある。ユースケース分析では、要求仕様から洗い出されたユースケースについて、各々シナリオフロー図を作成する。シナリオフロー図には、操作フロー図、処理フロー図、データフロー図の 3 種類の形式があり、分析対象に応じて選択する。作成したシナリオフロー図から、ユーザ視点での機能を抽出して各々コンポーネントに割り当てる。ここで抽出したユーザ機能は、単一責務の機能となっていなければならない。

(2)　モジュール（クラス）設計

イベントケース分析で作成するインタラクションフロー図において、ステレオタイプ《boundary》の「バウンダリ」、ステレオタイプ《control》の「コントロール」、およびステレオタイプ《entity》の「エンティティ」と呼ばれる 3 種類の責務構造を抽出し、モジュールとしてのクラスに割り当てる。割り当てられた 3 種類のクラス構造は、責務に関するデータと単一目的の関数処理でカプセル化、および情報隠蔽され、結合度が最低レベルに下げられたモジュール構造となる。

(3)　ステートマシン設計

ステートケース分析において作成する UML ステートマシン図と状態遷移マトリクスに基づいた状態管理の責務をクラスに割り当てる。状態管理の責務を割り当てられたクラス構造

は、状態遷移を制御するためのデータとステートマシンを管理、実行するための関数処理でカプセル化、および情報隠蔽され、結合度が最低レベルに下げられたモジュール構造となる。

(4)　関数（メソッド)、データ構造設計

イベントケース分析におけるインタラクションフロー図作成の中で、関数とデータ構造の分析を行う。インタラクションフロー図は、UML アクティビティ図を拡張した分析用のダイアグラムである。インタラクションフロー図におけるオブジェクトノードの意味付けは、ステレオタイプ《temporarydata》の「一時データ」か、ステレオタイプ《datastoe》の「永続データ」のいずれかとしている。ただし、通常、「一時データ」のステレオタイプは省略することにしている。また、アクションノードの意味付けはステレオタイプ《function》の「関数」としている。通常、ステレオタイプ《function》は省略してよいことにしている。他には、エッジの意味付けとして、「フロー」以外に「アクセス」を表現する場合がある。「アクセス」には、内部（例えば、永続データ）へのアクセスと、外部へのアクセス（例えば、イベント出力）の 2 種類がある。「アクセス」を意味するエッジには、「フロー」と区別するため、エッジ名を付すようにしている。「一時データ」と「永続データ」には、できるだけ具体的なデータ名を付け、「関数」には「～を･･･する」という基本形式で単一の目的が判定できる関数名を付ける。この「関数」の命名基準には、関数の凝集度を最高レベルに上げるという狙いがある。

4. 適用手法の評価と確認できた効果

4.1. 課題解決に対する評価

2.2 で提示した 3 つの課題の解決に向けた目標の達成度について、その評価を以下に示す。

課題 1：ユーザが考えている機能の単位でソフトウェアを設計していく

［目標］ユーザ機能の導出は、要件定義、または要件定義の前後で行われるが、ユーザ機能の単位でソフトウェアの構造や振る舞いを設計していくことは簡単ではなく、具体的な方法論が必要になる。従来では構造化分析やオブジェクト指向分析などの技法があるが、ユーザ機能単位の設計という面からは、いずれも一長一短があり決定的ではないと認識している。そこで、ユーザ機能の導出から、ユーザ機能の単位を意識してソフトウェア設計へつないでいく方法論を確立させる。

［評価］モデル指向開発で定義する機能は、開発者視点の機能（ファンクション）ではなく、ユーザ視点の機能（フィーチャ）である。各開発フェーズでユーザ機能を単位としたソフトウェアの分析モデル、設計モデルを作成し、実装へと進めていく開発プロセスが確立できた。本成果により、ユーザが考えている機能の単位でソフトウェアを設計していくことが可能となった。

課題 2：トレーサビリティを確保できるようなソフトウェア開発を行う

［**目標**］新規開発や派生開発のトレーサビリティを確保するためには、要件定義から実装までの各開発フェーズの成果物について、開発フェーズ間をつなげるキー情報が必要になる。開発成果物のキー情報の候補としては要求仕様、機能仕様、設計仕様、実装仕様があるが、これらを相互につなげ、管理していくことは容易ではない。そこで、要件定義から実装までの開発成果物のキー情報を定義し、それらを論理的につなげ、管理していく方法論を確立させる。

［**評価**］モデル指向開発の各開発フェーズで作成するモデルに、トレーサビリティを確保するためのキー情報を定義し、そのキー情報により、すべてのモデルは双方向に論理的につながる構造とする方法、および確保したトレーサビリティをモデル・トレーサビリティマトリクスで管理する方法を確立できた。本成果により、上流の要求仕様から下流のプログラムまでトレーサビリティを確保できるようなソフトウェア開発を行うことが可能となった。

課題 3：適切なソフトウェア・アーキテクチャを中心に考えた開発を行う

［**目標**］ソフトウェア・アーキテクチャは、ソフトウェアに対する要求仕様と外部とのインタフェース仕様から決定される。これらの仕様は主にソフトウェアの振る舞いに関するものであるが、ソフトウェア・アーキテクチャは構造と振る舞いの両面で決定されなければならない。また、その決定は要求仕様と外部とのインタフェース仕様を満たしていなければならないが、その根拠を設計レベルで示すことは難しい。そこで、新規開発ではアーキテクチャ決定の根拠を示せ、派生開発では既存のアーキテクチャを劣化させない方法論を確立させる。

［**評価**］モデル指向開発において、要件分析フェーズの成果物である要件分析モデルをソフトウェア・アーキテクチャ決定の根拠として、方式設計フェーズにてソフトウェアの構造と振る舞いに関するアーキテクチャモデルを作成することで、アーキテクチャを決定していくプロセスと技法を確立できた。本成果により、適切なソフトウェア・アーキテクチャを中心に考えた開発を行うことが可能となった。

4.2. 定性的効果

これまで適用した開発プロジェクトにて確認できている共通的な定性的効果について述べる。適用後、最初に確認できたのは「品質の観点」における効果である。2 回目以降の適用で確認できたのが「効率の観点」における効果である。また、繰り返しの適用で確認できたのが「再利用の観点」における効果である。それらの内容について以下に示す。

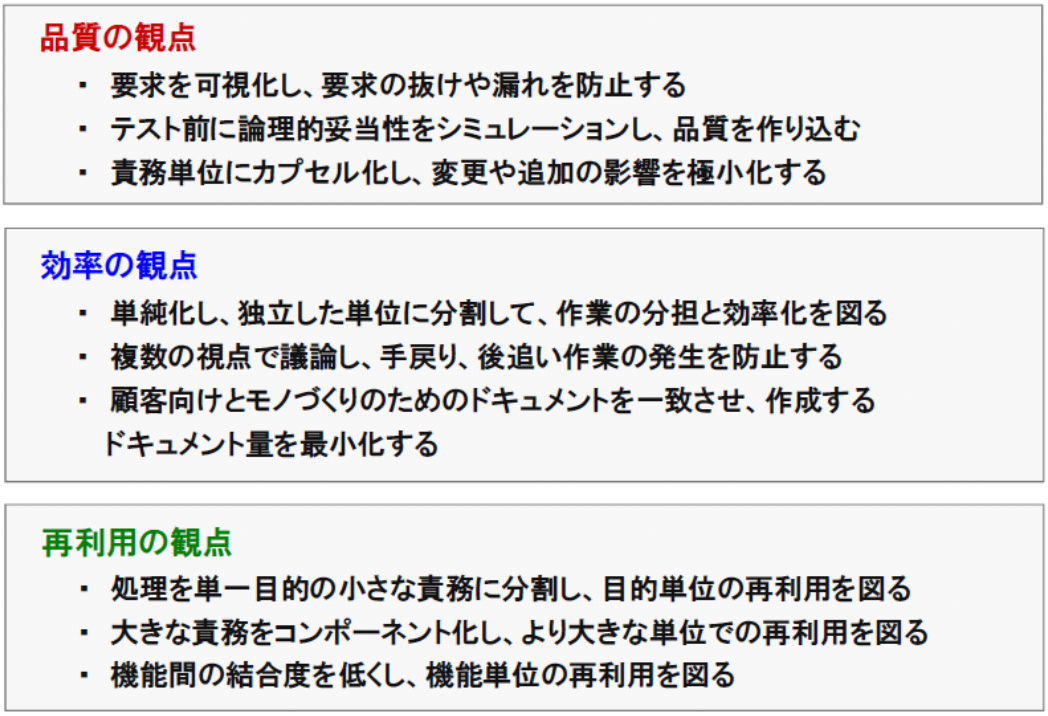


図 81-13 確認できた定性的効果の項目

品質の観点における「論理的妥当性をシミュレーションする」とは、設計モデリング（詳細設計）フェーズにて行う「クラス間処理シーケンス設計」と「コンポーネント間処理シーケンス設計」のアクティビティのことである。

なお、適用の定量的効果としては、これまで適用した開発プロジェクトにて開発されたソフトウェアシステムについて、納入後の不良発生件数を 0 にできることが確認されている。

5. 今後の取組みと課題

5.1. 取組みの目標

モデル指向開発の狙いと大きな効果を理解した上で、開発現場の状況に沿って段階的に導入していくことが重要なポイントとなる。また、モデル指向開発を開発現場で実践するためには、多くの要素技術を必要とし、そのための自己学習と開発現場での経験が不可欠になる。以下にモデル指向開発手法を開発現場に普及させていくための目標を挙げる。

（1） モデリング技法を普及させる

- UML、SysML による正確なダイアグラミング技法を指導する
- モデル図を活用した要件分析と設計の技法を指導する
- 開発の中でモデル図による要件分析、設計の機会を増やす

(2)　モデル指向開発の成功事例と技術者を増やす

- 効果を理解させ、部分的、段階的な導入を支援する
- 現状の課題を解決する対策の１つとして提案し技術支援する
- 開発現場で柔軟に適用できる技術者を育成する

(3)　派生開発への導入実績を増やす

- 派生開発 XDDP 適用の第２ステップとして導入を支援する
- モデル指向開発への派生開発プロセス導入を支援する
- モデル図を活用した変更要件分析と変更設計の技法を指導する

5.2. 今後の課題

ソフトウェアエンジニアリングにおける本質的、かつ究極の課題は、以下の 2 項目であると認識している。これらの課題について、モデル指向開発を基盤手法として解決を目指したい。

(1)　ソフトウェア仕様検証（validation：妥当性検証）

ソフトウェア仕様は自然言語で記述するのが一般的であるが、自然言語での表現には曖昧さがあり、その結果、仕様記述の正確さが失われ、勘違いや思い込みなどが発生してしまう。また、仕様の妥当性をチェックする上でも機械的仕組みの導入ができず、人間の判断に依存してしまう。

その対策として、曖昧さを排除し、仕様の妥当性を機械的にチェックできるようにする目的で、自然言語以外の言語で仕様を記述することが考えられ、導入する試みが進められている。

(2)　ソフトウェア検証（verification：正当性検証）

ソフトウェアが仕様通りに作られているかの検証には、ソフトウェアテストを用いるのが一般的であるが、ソフトウェアテストでは、欠陥が存在することを示すことはできるが、欠陥が存在しないことは証明できない。ここでの証明とは、ソフトウェアが仕様通りに振る舞うことを客観的に保証する行為である。

その対策として、ソフトウェアテストでは不可能な正当性の証明を可能にする目的で、ソフトウェアの意味を定義し、仕様で要求されるソフトウェアの性質を数学的に解析していくという試みが進められている。

参考文献

1. 要求の仕様化・表現技術 USDM 技法

[1] [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、2005.10

[2] [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? 改訂第 2 版清水吉男、技術評論社、2010.5

2. システムズモデリング言語 SysML 技法

[3] システムズモデリング言語 SysML サンフォード・フリーデンタール、アラン・ムーア、リック・スタイナー、西村 秀和他訳、東京電機大学出版局、2012.5

[4] A Practical Guide to SysML, Third Edition: The Systems Modeling Language Sanford Friedenthal、Alan Moore、Alan Moore、Morgan Kaufmann、2014.11

[5] OMG Systems Modeling Language Version 1.4 Object Management Group、Object Management Group、2015.9

3. 構造化分析・設計手法

[6] 構造化分析とシステム仕様 トム・デマルコ、高梨 智弘・黒田 順一郎訳、日経 BP 社、1994.8

[7] ソフトウェアの複合/構造化設計 G.J. マイヤーズ、国友 義久・伊藤 武夫訳、近代科学社、1979.1

[8] ソフトウェア構造化技法—ダイアグラム法による J. マーチン、C. マックルーア、国友 義久・渡辺 純一訳、近代科学社、1986.11

4. オブジェクト指向分析・設計手法

[9] オブジェクト指向入門 バートランド・メイヤー、酒匂 寛・酒匂 順子訳、アスキー、1990.11

[10] オブジェクト指向入門 第 2 版 原則・コンセプト バートランド・メイヤー、酒匂 寛訳、翔泳社、2007.1

[11] オブジェクト指向入門 第 2 版 方法論・実践 バートランド・メイヤー、酒匂 寛訳、翔泳社、2008.8

[12] オブジェクト指向システム分析—上流 CASE のためのモデル化手法 S. シュレィアー、S.J. メラー、本位田 真一・山口 亨訳、啓学出版、1990.2

[13] オブジェクト指向分析—OOA 第 2 版 P. コード、E. ヨードン、羽生田 栄一他訳、トッパン、1993.4

[14] オブジェクト指向設計「OOD」—Coad/Yourdon メソッド P. コード、E. ヨードン、小畑 喜一・中川 耕一・吉田 誠訳、プレンティスホール出版、1995.3

[15] オブジェクト指向方法論 OMT—モデル化と設計 ジェームズ・ランボー、ウィリアム・プレメラニ、ウィリアム・ローレンセン、マイケル・プラハ、フレデリック・エディ、羽生田 栄一訳、トッパン、1992.7

[16] オブジェクト指向ソフトウェア工学 OOSE—use - case によるアプローチ I. ヤコブソン、P. ジョンソン、M. クリスターソン、G. アーバガード、梶原清彦・西岡利博・渡邊克宏訳、トッパン、1995.9

[17] Booch 法:オブジェクト指向分析と設計 第 2 版　グラディ・ブーチ、山城 明宏・清水 洋子他訳、アジソンウェスレイパブリッシャーズジャパン、1995.11

[18] アジャイルソフトウェア開発の奥義　ロバート・C・マーチン、瀬谷 啓介訳、ソフトバンククリエイティブ、2004.6

5. 責務割当パターン GRASP 技法

[19] 実践 UML —パターンによるオブジェクト指向開発ガイド　クレーグ・ラーマン、依田 光江・依田 智夫、今野 睦訳、プレンティスホール出版、1998.12

[20] 実践 UML 第 2 版 —パターンによる統一プロセスガイド　クレーグ・ラーマン、依田 光江・依田 智夫、今野 睦訳、ピアソンエデュケーション、2003.10

[21] 実践 UML 第 3 版 —オブジェクト指向分析設計と反復型開発入門　クレーグ・ラーマン、依田 光江・依田 智夫、今野 睦訳、ピアソンエデュケーション、2007.11

6. ロバストネス分析技法

[22] ユースケース入門—ユーザマニュアルからプログラムを作る　ダグ・ローゼンバーグ、ケンドール・スコット、長瀬 嘉秀・今野 睦訳、ピアソンエデュケーション、2001.11

[23] ユースケース駆動開発実践ガイド　ダグ・ローゼンバーグ、三河 淳一・船木 健児・佐藤 竜一訳、翔泳社、2007.10

7. ソフトウェアモデリング言語 UML 技法

[24] UML による統一ソフトウェア開発プロセス—オブジェクト指向開発方法論　イヴァー・ヤコブソン、ジェームズ・ランボー、グラディ・ブーチ、藤井 拓訳、翔泳社、2000.3

[25] UML リファレンスマニュアル　ジェームズ・ランボー、グラディ・ブーチ、イヴァー・ヤコブソン、石塚 圭樹訳、ピアソンエデュケーション、2002.1

[26] UML2.0 仕様書 2.1 対応　Object Management Group、西原 裕善訳、オーム社、2006.11

[27] UML ユーザガイド第 2 版　グラディ・ブーチ、ジェームズ・ランボー、イヴァー・ヤコブソン、羽生田 栄一・越智 典子訳、ピアソンエデュケーション、2010.3

[28] UML2.0 クイックリファレンス　ダン・パイロン、ニール・ピットマン、原 隆文訳、オライリージャパン、2006.6

[29] リアルタイム UML—オブジェクト指向による組込みシステム開発入門　ブルース ダグラス、渡辺 博之他訳、翔泳社、2001.3

[30] 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

[31] OMG Unified Modeling Language Version 2.5　Object Management Group、Object Management Group、2015.3

8. フィーチャ駆動開発手法

[32] Java エンタープライズ・コンポーネント—カラーUML による Java モデリング　ピーター・コード、ジェフ デ・ルーカ、エリック・レイフェイブル、依田 光江・依田 智夫・今野 睦訳、ピアソンエデュケーション、2000.9

[33] アジャイル開発手法 FDD—ユーザ機能駆動によるアジャイル開発　スティーブン・R. パルマー、ジョン・M. フェルシング、今野 睦・飯塚 富雄・長瀬 嘉秀訳、ピアソンエデュケーショ

ン、2003.3
9. ドメイン駆動開発手法
[34] Domain-Driven Design: Tackling Complexity in the Heart of Software　Eric Evans、Addison-Wesley Professional、2003.8
[35] ドメイン駆動　ジミー・ニルソン、尾島 良司・長尾 高弘訳、翔泳社、2008.3
[36] 実践ドメイン駆動設計　ヴァーン・ヴァーノン、高木 正弘訳、翔泳社、2015.3
10. 責務駆動設計技法
[37] オブジェクトデザイン　レベッカ・ワーフスブラック、アラン・マクキーン、藤井 拓他訳、翔泳社、2007.9
11. インタフェース指向設計技法
[38] UML コンポーネント設計―コンポーネントベースソフトウェアのための開発プロセス　ジョン・チーズマン、ジョン・ダニエルズ、長瀬 嘉秀・今野 睦訳、ピアソンエデュケーション、2002.5
[39] インターフェイス指向設計―アジャイル手法によるオブジェクト指向設計の実践　ケン・パーク、角谷 信太郎・児島 修訳、オライリージャパン、2008.5
12. 派生開発プロセス手法 XDDP
[40] 「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、2007.10

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)