特集1 Java勝ち組プログラミング オブジェクト指向を味方につけろ!

5章 オブジェクト指向開発のポイント

プロセスからパターンまで

ソニー㈱
橋本隆成
Takanari HASHIMOTO

Javaは「オブジェクト指向言語」です．しかし，Java言語を理解すれば，それだけで優れたオブジェクト指向プログラムを開発できるわけではありません．本章では，Java言語で優れたオブジェクト指向プログラムを開発するには何が必要か，何が重要なポイントであるかを考えていきたいと思います．

優れたプログラムを開発するために

優れたオブジェクト指向プログラムを開発するには，Java言語の習熟はもちろん，オブジェクト指向分析設計の知識，OSやデータベースの知識など，非常に多くの技術や経験が要求されます．今回は誌面の都合もありますので，特に重要なものの中から，「開発プロセス」「アーキテクチャ」「パターン」の3点をピックアップし，解説していこうと思います．

開発プロセス

ソフトウェア開発にとって，プロセスは大変重要です．最近はプロセスについての研究や紹介が雑誌の記事を賑わせていますし，多くの書籍も出版されています．プロセスには大きく分けて，技術的な作業のフローを記述した「エンジニアリングプロセス」と，プロジェクトを効果的に管理，推進していくための「プロジェクト管理プロセス」があります．今回は，このうち「エンジニアリングプロセス」について解説することにします．

なぜプロセスが重要なのでしょうか．ソフトウェア開発に限らず，ハードウェア開発やビルや橋などの建築物の開発でも，プロセスはもちろん重要です．しかし，ソフトウェアの場合は“開発対象が目に見えない”上に，他の分野と比較して非常に急激に技術が進歩したために，ソフトウェア開発を開発者が完全に制御しきれていないからです．このことから，「要求定義」から「設計」工程で作業すべき項目を明確にし，慎重にレビューを行いながらソフトウェアの品質を作りこんでいくという作業が不可欠になり，プロセスが大変重要となるのです．

生産性の視点からも，上流工程での欠陥は，下流工程で発見され修正される場合には，上流工程で発見され修正される場合と比較して，コストと時間が大きくなることが定量的に示されています．つまり，ソフトウェアの品質と生産性は，プロセスによって決まってしまうといっても過言ではない状況にあります．特に，ソフトウェア開発では，「実装」工程前の上流工程の作業が極めて重要です．ソフトウェア開発を「要求定義」から「設計」工程に分割して，レビューやインスペクションを行いながら進めていくことで，確実に要求定義書とプログラムまでの技術的な大きな隔たりを効果的に埋めて行くことが可能になります．

大規模なソフトウェアシステムでは，多数のエンジニアが開発に関わり，複数のテームによる開発が行われるのが普通です．各作業工程を異なるエンジ

T1-5 02.10.25 0:40 PM ページ2

ニアが担当することも多く，優れた開発プロセスか否かで開発されるソフトウェアの生産性，品質が大きく変わってくることになります．

◎ウォーターフォール型と反復型

ソフトウェアの大規模化，複雑化に伴い，新規性の高いソフトウェア開発では，ウォーターフォール型ではなく「**反復型開発**」と呼ばれる開発プロセスが主流になってきています（図1，図2）．ソフトウェアはハードウェアと違って直接目に見えないために，各工程間の作業成果物のレビューで完全に見落としや間違いを発見することが困難です．また，実行速度やマルチタスクなどのタイミング，リソースの競合の検証は，机上のチェックやレビューでは限界があり，実際にソフトウェアを作成して動作させてみないと検証が困難な項目もあります．

そこで，実現する機能を少しずつ作成し，動作確認をしながら開発を進めることが不可欠になってきました．反復型開発では，機能の一部分を実現したソフトウェアを実際に開発して動作させることで，要求されている機能がユーザの要望と合致しているか，設計上の問題点はないかなどを確認しながら，機能を徐々に実現していきます．

反復開発では，ソフトウェア開発のライフサイクルを，いくつかの反復開発に分けて開発を行います．この反復のことを，「イテレーション」と呼ぶことがあります．各イテレーションは，「ウォーターフォール型」開発の流れで作業が進められていきます．

以下では，開発プロセスの各工程の意味と注意点を簡単に解説していくことにします．

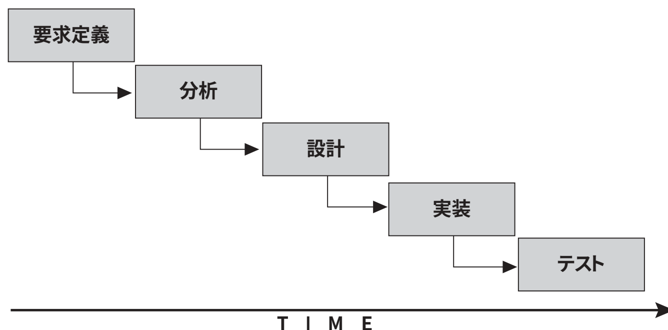
◎要求定義

「要求定義」とは，開発するソフトウェアの機能要求や制約事項などを洗い出し，定義する作業を指します．「要求定義」の作業の成果物が「要求仕様書」とか「要求定義書」と呼ばれるものであり，開発するソフトウェアの機能要求や制約事項などを記述したドキュメントです．

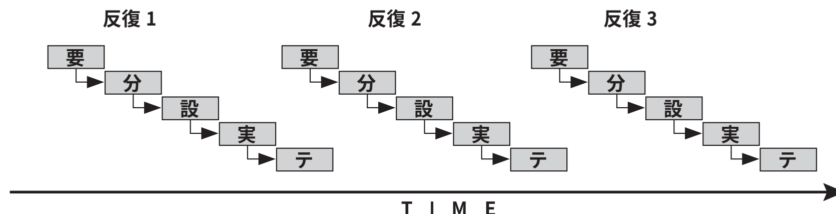
ソフトウェア開発において，「要求仕様書」はとても重要です．「要求仕様書」に記述された内容を基に以後の開発作業は進められていきますから，「要求定義」作業が十分に行えず，不完全あるいは誤った「要求仕様書」が作成されれば，それを基に行われる分析，設計そして実装（プログラム作成）もすべて誤ったものになってしまいます．

近年，ソフトウェアが複雑になるにつれて，「要

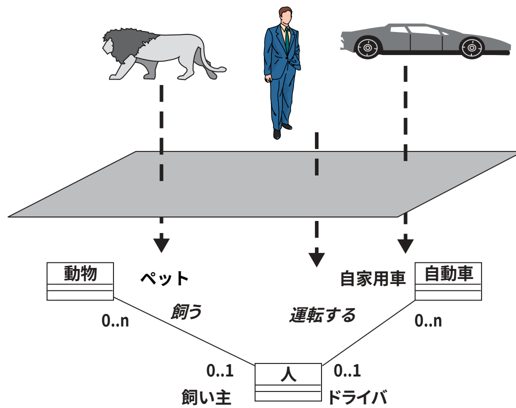
●図1　ウォーターフォール型開発の流れ



●図2　反復型開発の流れ



●図3 分析では，抽象化の作業を行う



求定義」が非常に困難になってきました．複雑かつ大規模化するソフトウェアの機能や制約事項を，矛盾や過不足なく正確に記述することは大変な作業だからです．現在では，UMLにも定義されている**ユースケース**を用いた要求定義が一般化しつつあります．ユースケースによる要求定義は，「プログラムのユーザを要求定義に巻き込みやすく，ユーザの視点から要求定義をまとめられる」「ユースケース単位で反復型開発を行え，プロジェクト管理もやりやすい」など，多くの優れた点を備えているからです．

◎分析

「分析」工程の作業は，開発するシステムの要求機能や制約事項を整理し，ソフトウェアで実現する対象の構造や振る舞いの特徴を洗い出し，抽象化する作業です（図3）．ソフトウェアで実現する問題対象領域（ドメイン）を，論理的に「静的な構造」と「動的な構造」の両面に形式化します．

オブジェクト指向開発の場合，「静的な構造」を表現する中心的な成果物は，UMLではクラス図やパッケージ図などになります．一方，動的な構造は，状態図やコラボレーション図などが中心となります．分析工程作業で重要なのは，開発の実行環境（OSやハードウェア環境）や開発言語などを考慮しないで，問題対象領域の要求機能や制約事項を形式化することです．具体的な実行環境や開発言語への最適化は，「設計工程」で行うことになります．分析工程の作業が不十分なまま設計工程の作業に進んだ場合，優れた設計を行うことは困難となってしまいます．システムの優れた保守性，拡張性をもつアーキテクチャを設計するには，「分析」工程の作業で，しっかりと問題対象領域の構造や振る舞いの特徴を形式化し，把握しておかなければなりません．

◎設計

分析工程の作業結果から，開発の実行環境（OSやハードウェア環境）や開発言語などを考慮し，問題対象領域の要求機能や制約事項を最適化します．

設計工程の大きな作業の中心は，「アーキテクチャ」の決定と構築です．アーキテクチャの決定では，プログラム実行時のパフォーマンスやコードサイズ，利用するミドルウェア，ライブラリなども重要な課題となります．Java言語のメカニズムである「カプセル化」や，「継承」「インタフェース」による「ポリモーフィズム」，さらにはデザインパターンやアーキテクチャパターンを利用して，保守性と拡張性をアーキテクチャ上に実現させることを検討するのも設計工程作業です．

当然，OS，ミドルウェア等の専門知識も必要とする，きわめて高度な作業になります．アーキテクチャを設計するエンジニアは「アーキテクト」と呼ばれ，大変重要な責任を持つことになります．

設計工程での作業は，オブジェクト間の協調関係を詳細に定義し，クラスのメソッドの引数の数や型も検討していきます．ただし，一度の多くの技術的課題や機能を設計し，アーキテクチャを構築するのは困難です．そこで，反復型開発を用いて，少しずつ実現する機能設計の妥当性を検証していきます．

◎実装

設計工程の作業結果を受けて，開発言語でプログラミングを行います．Java言語でプログラミングをするのはこの工程です．

オブジェクト指向開発の場合，設計工程の成果物であるクラス図や状態図が重要な入力になります．最近では，開発環境であるCASEツールの機能が充実してきており，たいていのCASEツールであれば，クラス図や状態図からJava言語のスケルトンを自動

生成することが可能です．

◎テスト

Java言語で実装したプログラムのテストです．テストは大変重要な作業です．テストには，「単体テスト」「コンポーネントテスト」「システムテスト」があります．

要求仕様書が作成されれば，並行してシステムテストの計画や作業準備を始めることが可能です．実装工程の作業が終了してから慌ててテストの準備をするのは避けるべきです．

アーキテクチャ

アーキテクチャは大変重要ですが，優れたアーキテクチャを構築するのは難しい作業です．その理由の1つは，アーキテクチャにはさまざまな要素が関係しており，大変複雑でデリケートな課題を解決しなければならないからです．

アーキテクチャの構築で扱う事項には，以下のようなものがあります．

1. ソフトウェアの静的なアーキテクチャ構造：クラス，パッケージ，階層化（レイヤ）
2. CPUスペック，CPU数，CPU間構成，バス構成
3. 各物理ノード上に配置するソフトウェアコンポーネント構成
4. ソフトウェアの動的なアーキテクチャ構造：オブジェクト間のイベントの送受信方法，タスク間通信設計
5. 例外処理設計
6. 分散性，永続性の実現方法や製品の決定 etc．

パターン

パターンは，現在最も注目され，研究されているものの1つです．パターンの価値は，世界中のエンジニアたちがソフトウェアを開発する際に遭遇した課題や問題を解決した「定石」「手筋」に該当するものです．読者のみなさんがソフトウェア開発で遭遇する課題や問題について，すでに世界中のエンジニアが似た課題や問題に遭遇し，すでに解決していることは十分考えられることです．パターンを利用すれば，そのような問題についてみなさんが新たに試行錯誤しながら検討しなくてもすむということになります．ここでは特に代表的なパターンを紹介していきます．

アナリシスパターンは，分析のパターンで，開発対象領域の分析を対象にしたものです．抽象度が高く，汎用的なパターンが多く見られます．最近では，ビジネス分野の業務内容ごとに整理された「ビジネスパターン」なども存在しています．

アーキテクチャパターンは，アーキテクチャのためのパターンです．システム全体あるいは骨子を構成するためのパターンという点で，次に紹介するデザインパターン（設計パターン）とは異なっています．アーキテクチャは，ソフトウェアの性能，保守性，拡張性，可読性に大きな影響を与えますので，アーキテクチャパターンは重要なパターンです．

デザインパターンは，設計のためのパターンで，パターンの粒度が比較的小さく，いろいろなところで利用できるようになっています．主にクラス間の構造や振る舞いなど，かなり具体的なメカニズムを解説しています．

実装パターンは，パターンと呼ばれるよりは「イディオム」と呼ばれることが多いようです．他のパターンとは異なり，特定のソフトウェア言語によるアルゴリズム集，実装例となっています．

変り種のパターンとして，“落とし穴”パターンとか“反面教師”パターンを解説したのが，アンチパターンです．

最後に

Javaで優れたオブジェクト指向開発を実現するために何が重要であるかを，駆け足で解説してきました．誌面の都合上，ポイントだけを列挙した形になりましたが，オブジェクト指向開発入門者の方に，開発を進める上でどのようなことがポイントであるかを理解していただければ幸いです．J