UML

第1章　オブジェクト指向開発入門

1-1 オブジェクト指向開発とは

「オブジェクト指向」ということばを定義すると、「これから作成する（開発する）対象のもの、事柄をすべてオブジェクト（もの）としてとらえて表現する考え方」である

1-1-1 オブジェクト指向システムとは

オブジェクト指向システムとは、私たちが現実世界で「もの」と扱う様子をシミュレートしたシステムである。「もの」の扱い方を考えることは、私たちにとってごく当たり前の思考であり、オブジェクト指向システムを開発するということは、人間の思考に近いシステムを開発するということになる。

1-1-2 オブジェクト指向の考え方

例えば、「自動販売機」というシステムを開発することになったとする。

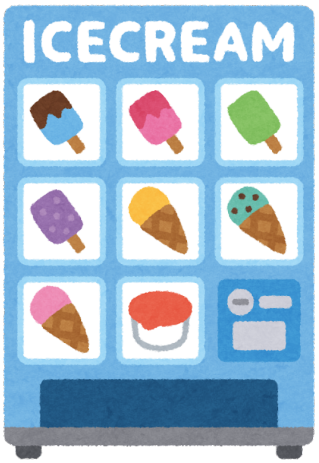
最初に行う作業は、「どんな商品を販売するのか」「商品んの価格はいくらにするのか」「どんな機能を持っているのか」「利用者はなにをするのか」といった項目を分析することになります。

▼オブジェクトの性質を洗い出す

自動販売機のもつ性質を整理して考える。自動販売機の持つ性質をデータ（属性）と機能（操作）にわけて洗い出す。

オブジェクト指向では、オブジェクトが持っている数値や文字列のようなデータを「属性」という。自動販売機の持っているデータは、「商品名」や「価格」などの項目になる。

オブジェクトのもっている機能は「操作」という。自動販売機の機能といえば、「支払金を受け取る」「おつりを計算する」といった機能になる。ほとんどのオブジェクトは、その性質を「属性」「操作」にわけて表現することができる。



自動販売機
【属性】 ・商品名 ・価格 ・支払金額 など
【操作】 ・支払金を受け取る ・商品を出す ・おつりを算出する など

図　オブジェクトの性質を洗い出し「抽象化」

このように、オブジェクトを持つ性質を一定の基準で抽象的に洗い出すことによって、開発対象のオブジェクトを把握し易くするのがオブジェクト指向の特徴である。

▼クラスを定義する

ここまでの分析結果は、これからのオブジェクトの作る際の指針になる「設計図」のようなものである。オブジェクトの分析で導きだされた「属性（フィールド）」「操作（メソッド）」に整理された設計図のことをクラスと呼ぶ。

自動販売機
【属性】 ・商品名 ・価格 ・支払金額
【操作】 ・支払金を受け取る ・商品を出す

・おつりを算出する

図　抽象的に定義された「自動販売機」クラス

▼クラスをもとにオブジェクトを生成する

オブジェクト指向では、クラスからオブジェクトを生成することを「インスタンス化」という。インスタンス化の際、属性である商品と価格は「オブジェクトが出来上がった時に決まっていなければならない属性値」であり、初期設定が必要な項目となる。

例えば、商品名はコーヒー、価格は140円といった値を設定する。支払金額は「オブジェクトが出来上がった後に決まる属性値」であり、自動販売機を使う利用者が設定する項目なので、オブジェクトの生成時には設定しない。

利用者は、出来上がった自動販売機オブジェクトの操作を利用することができる。

例えば、「支払金を受け取る」操作を利用すれば、自動販売機オブジェクトの属性「支払金額」に代金を設定することができる。

「商品を出す」操作を利用すれば、自動販売機オブジェクトを生成した際に初期設定さらた商品名の商品が出てくるだろう。

「おつりを算出する」操作を利用すれば、自動販売機オブジェクトを生成した際に初期設定された価格と支払金額の差額、つまり、おつりを得ることができるはずである。「おつりを算出する」操作は、「商品を出す」操作が内部的に利用することになるかもしれない。

このように、自動販売機オブジェクトの内部では、操作によって属性の値を利用している。なお、利用者が操作を利用する・起動することを、「メッセージ」を送信するという。

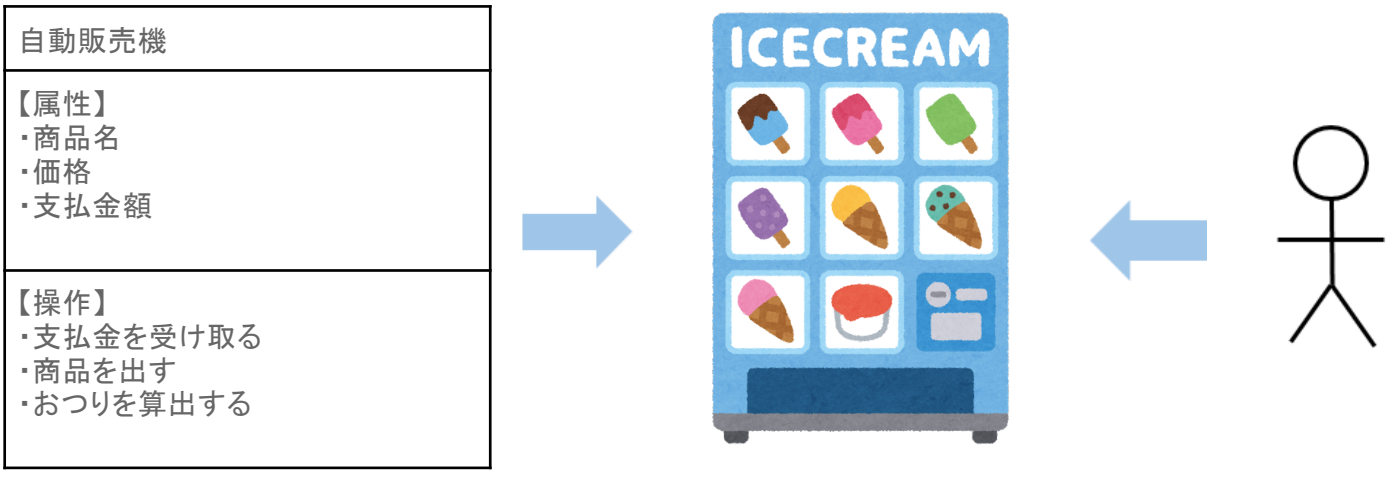


図 クラスからオブジェクトを生成する

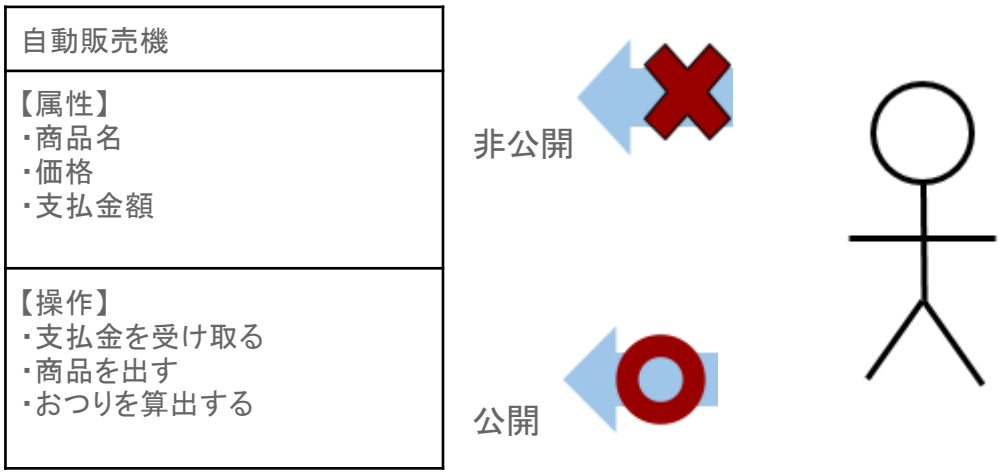
▼カプセル化

これまで見てきたように、オブジェクト指向では、「属性」と「操作」が一体化して無駄なく振舞うよう、対象のオブジェクトを整理する。利用者は、属性の具体的な値とを意識することなく、オブジェクトを利用することができる。

属性と操作が一体となっていオブジェクトとして扱われることを「カプセル化」とよぶ。

▼情報隠ぺい

カプセル化されたオブジェクトでは、操作のみを「公開」して属性を「非公開」にする。これを「情報隠ぺい」とよぶ。情報隠ぺいにより、属性に対して不正にアクセスすることを防ぐことができる。



▼変更の容易性

カプセル化されたオブジェクトは、内部で属性値や操作の詳細に変更があったとしても、利用者がオブジェクトを扱う手続き影響を与えない。また、属性に関係する操作が同じオブジェクト内であるため、修正する範囲も狭くなる

▼最大のメリットは「再利用性」

オブジェクト指向の最大のメリットは、再利用性である。

自動販売機クラスに定義した「自動販売機」というクラス名や、「商品名」などの属性名は、非常にあいまいで抽象的である。つまり、このクラスは「アイスの自動販売機」や「ジュースの自動販売機」など、「自動販売機」と名の付くものならなんでも流用できる。

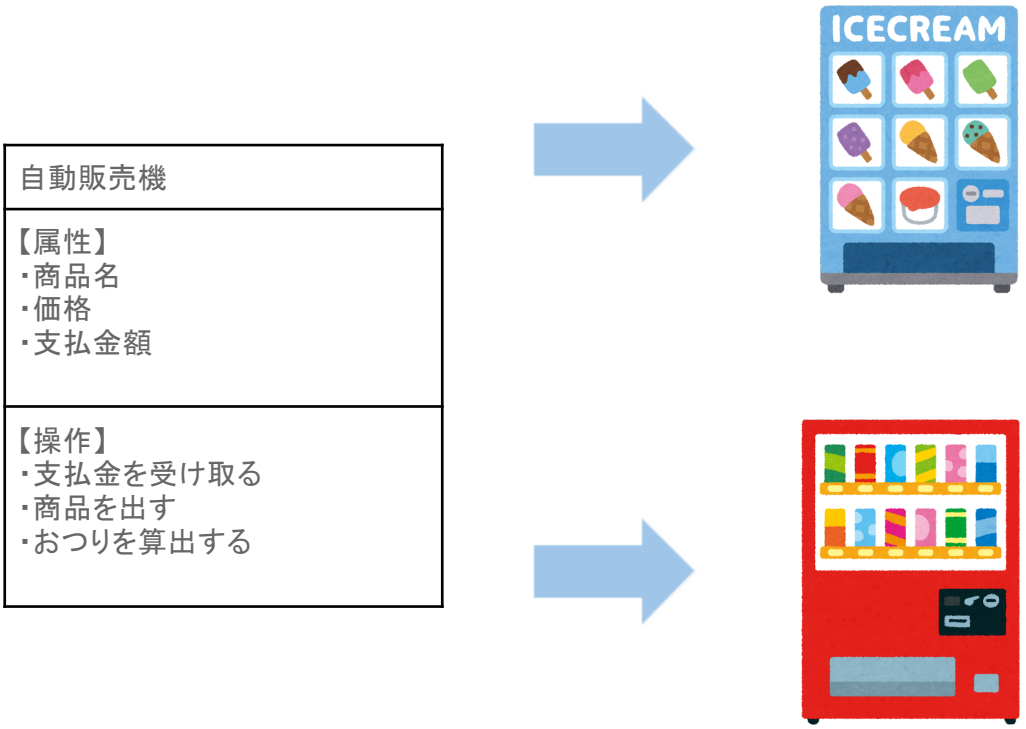


図　再利用性

▼クラスを細分化する

オブジェクトに対する分析をさらに進めていくと、より、汎用的なクラス定義を見出すこともできる。

自動販売機の例の場合、「商品」というクラスを新たに抽出して自動販売機クラスから参照するように定義することもできる。

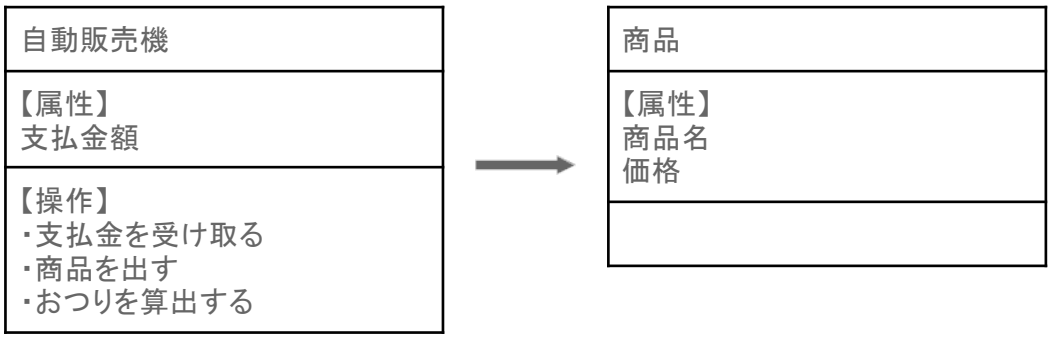


図　汎用的なクラス定義を見出す

このようにすることで、「商品」の情報に変更があった場合に自動販売機に与える影響が軽減できる。また、「商品」クラスは、他のシステムでも利用しやすくなる。自動販売機クラスも、商品の詳細を気にすることなく本来の役割に徹することができる。

▼「品質」を向上させることができる

様々なシステムで利用されたオブジェクトは、発見された不具合がフィードバックされて洗練されたオブジェクトに進化していく。

さらにそのオブジェクトを再利用することで、システム全体の品質を向上させることにつながる。

1-1-3 オブジェクト指向のキーワード

▼継承

継承は、既存クラスの属性や操作を新たに作成するクラスで引き継ぐことによって、同じ定義をしなくても利用できるようにする仕組みである。クラス間に「**is - a**」関係があることを表現する。

新たなクラス（サブクラス）は既存クラス（スーパークラス）にはなかった新しい操作の定義（差分定義）に集中できる。

汎化は、複数のクラスから共通点を抜き出し、抽象的かつ汎用的にスーパークラスを定義する考え方である。

（例1）操作を継承する

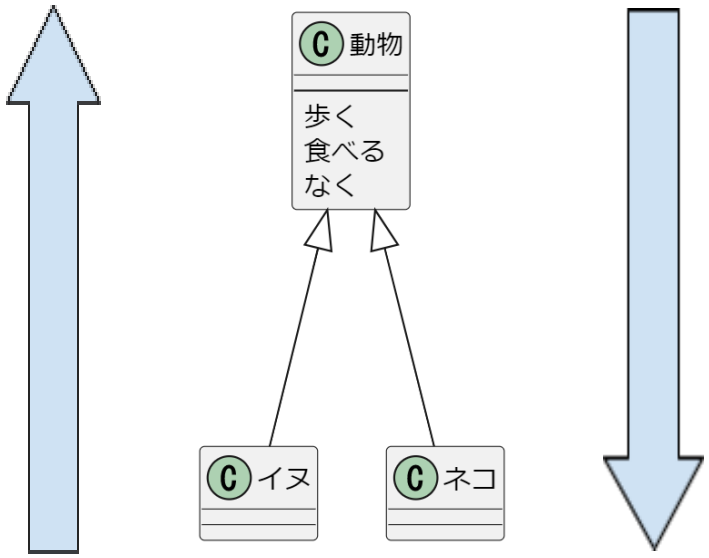


図　操作を継承する

（例2）差分を定義する

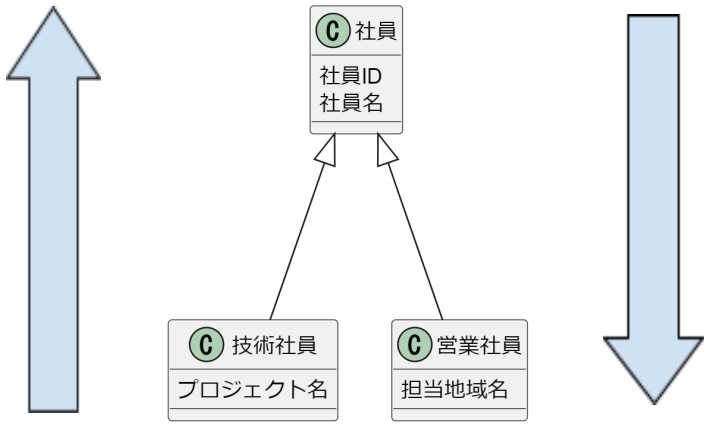


図　差分を定義する

▼集約

集約は、「全体 - 部分」の構造や、「所有する側 - 所有される側」の関係を明確に表現する場合に使用する。クラス間に「**part - of**」または「**has - a**」の関係があることを表現する。

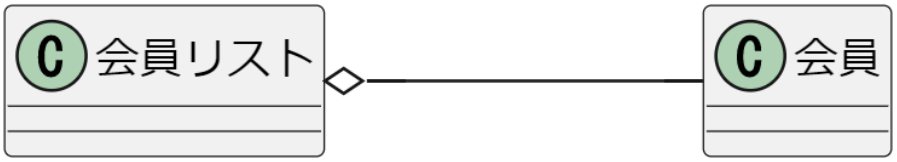
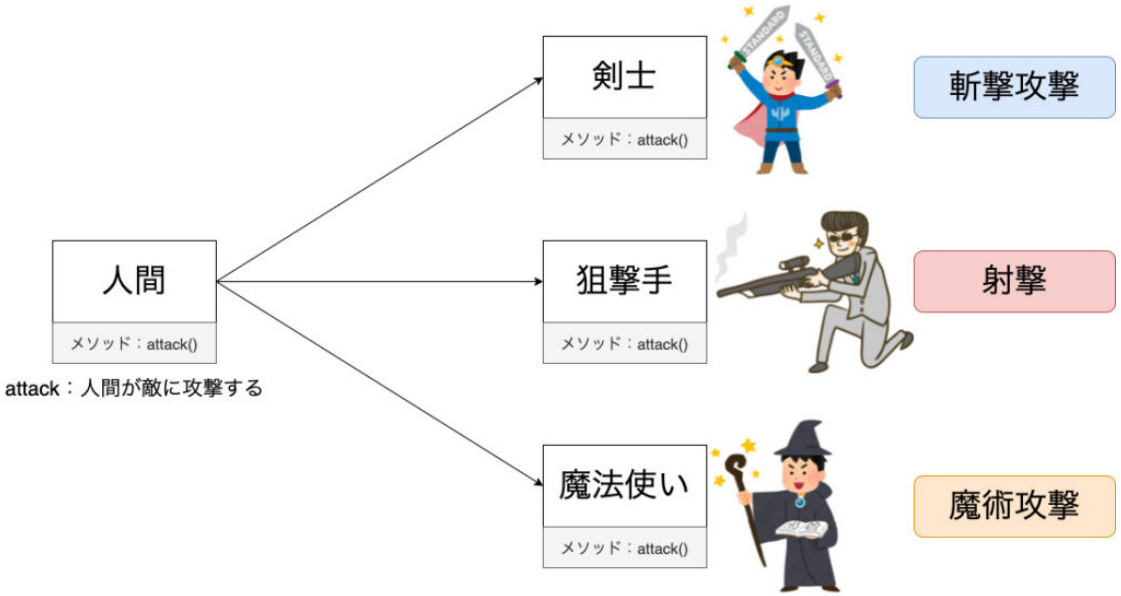


図　集約

▼ポリモーフィズム

一つの操作で様々な機能が利用できることを、ポリモーフィズム（多態性）と呼ぶ。ポリモーフィズムに則って実装された操作は、特に利用者にとって非常に使いやすいものになる。



1－2　オブジェクト指向による開発

ウォーターフォールモデルは、大規模なシステムを効率よく開発する目的で考案された手法である。ウォーターフォール（滝）の流れのように、後戻りなく各工程の成果物を完成させていく。

ウォーターフォールモデルには、次のような問題点がある。

- 最終段階まで動くシステムが見えないため、ユーザと開発者の間の認識の違いを発見しにくい。
- ある工程で遅れが生じると、テスト工程の期間が短縮するため品質が低下する。また、リリースを延期することで開発費用も増大し、顧客からの信頼も低下する
- 各工程で使用するドキュメント形式が異なるため、工程から工程へ移行する際に不具合が発生しやすい。
- システムに仕様変更が発生すると、費用や開発期間面でプロジェクト全体に大きな影響を与えてしまう

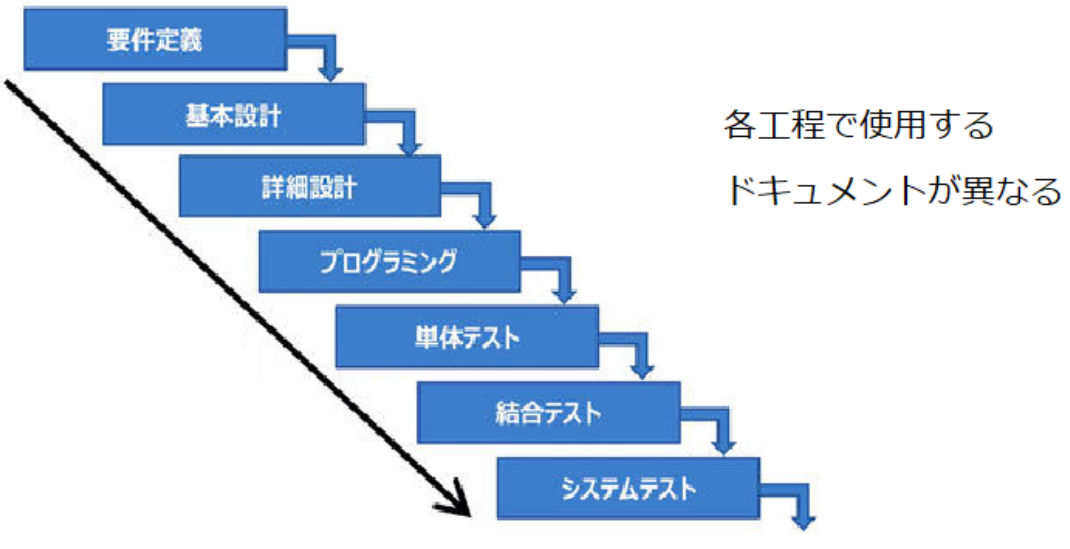


図　ウォーターフォールモデル

これらの問題点を解消するために、要件定義からテストまでの流れを繰り返しながらシステムを開発していく手法「スパイラルモデル」が考案された。

スパイラルモデルは、1回の繰り返しで要件定義からテストまでの作業を行う。次の繰り返しは、前回の繰り返しを評価して行う。

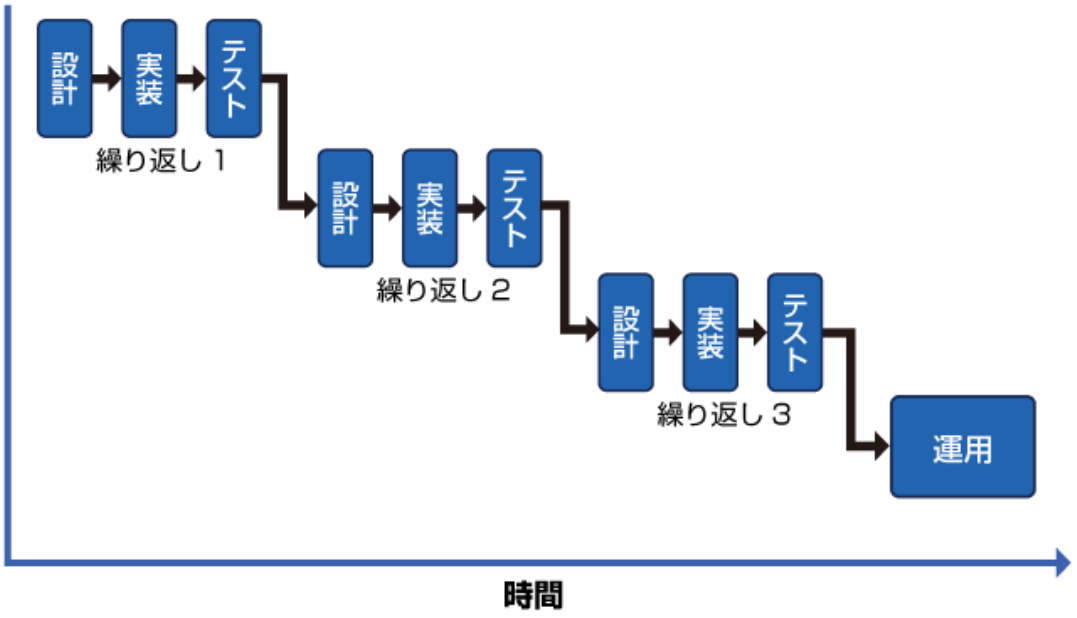


図　スパイラルモデル

- 1回の繰り返しごとに動くシステムが構築されるため、開発したシステムとユーザーの要求の食い違いが早期に発見できる
- 主要な機能から開発に着手できるため、主要機能の選考リリースができる
- 1回の繰り返しの結果を評価して次の繰り返しにそれを反映させるため、設計の不備を次の繰り返しで解決することができる

オブジェクト指向開発では、スパイラルモデルや統一されたドキュメント仕様（UMLなど）を用いることによって、各工程で発生する不具合を減らすことができる。

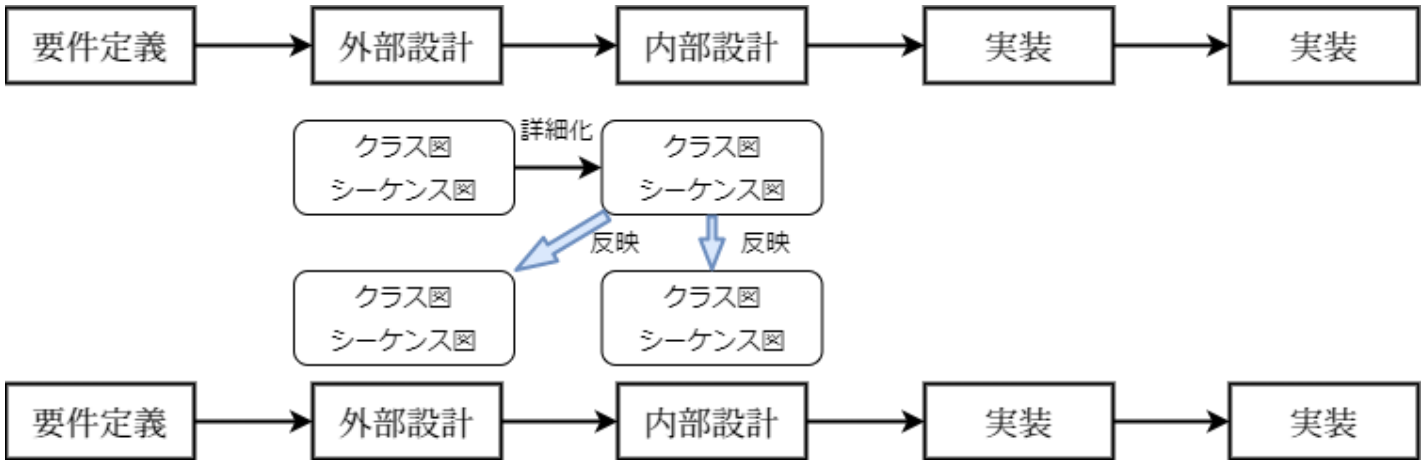


図　スパイラルモデルと統一されたドキュメント

オブジェクト指向開発における各フェーズの概要は次の通りである。

要件定義　・利用者から見たシステムに必要な機能を定義する

⇩

オブジェクト指向分析　・各機能の実現に必要なオブジェクトを詳細化する

⇩

オブジェクト指向設計　・実装を考慮して分析の結果を詳細化する

⇩

実装　・プログラミングを行い部品の動作を検証する

⇩

テスト　・システムが要求どおりの状態かを検証する

顧客の要求を把握し、それをいかにプログラマに伝えるかが、システム構築の中で最も難しい作業の一つであり、多くのシステムエンジニアが頭を悩ませてきた部分である。

従来の手法では、顧客と打ち合わせを重ね、作成した設計書をレビューして認証を得るという作業を繰り返していた。そして、作業の後にはプログラマに仕様を伝えるという作業が待っている。

システム構築をよく理解している顧客や、システムエンジニアの作業を補うことができるプログラマもいないわけではないが、そういった恵まれた環境は滅多にない。顧客はシステム構築について何も知らないし、ほとんどのプログラマは言われたとおりにしかプログラムを書かない。

このような顧客や、対プログラマで問題が発生する要因の一つが、「要求定義書」や「設計書」を自然言語で記述するという点である。仕様書をことばだけでは説明するのは非常に難しく、後からよく読んでみると、あいまいで、どちらの意味にも取れるものになってしまいがちである。

１－３　オブジェクト指向とUML

UMLとは、「Unified　Modeling　Language」の略で、日本語では、「統一モデリング言語」となる。言語といっても、C/C++やJavaのようにプログラムコードで表現するのではなく、その表現形式は「図」と「記号」を使ったものである。UMLは、顧客とシステムエンジニア（SE)、SEとプログラマーの橋渡しをするツールとして位置づけられる。システムの振る舞いや構造をオブジェクト指向で分析したり設計したりする際、図を用いることで視覚的に把握できるようになり、効果的に表現できる。

1—3－1　オブジェクト指向フェーズとUML

オブジェクト指向開発におけるフェーズとUMLダイアログの関係は次の通りである。

表　オブジェクト指向開発におけるフェーズとUMLの各ダイアログの関係

分類	名称	説明	フェーズ
動的	ユースケース図	システムに要求される機能をユーザ支店で表現する	要求定義
静的	クラス図	システムの静的な構造（クラスが持つ属性、操作、およびクラス間の関係）を表現する	分析・設計
静的	オブジェクト図	システムのある時点におけるオブジェクト間の関係を表現する	任意
動的	シーケンス図	オブジェクト間のメッセージのやり取り（相互作用）を時間軸に沿って表現する	分析・設計
動的	コミュニケーション図	オブジェクト間のメッセージのやり取り（相互作業）をオブジェクトの構成に沿って表現する	分析・設計
動的	ステートマシン図	オブジェクトの状態遷移を表現する	分析・設計
動的	アクティビティ図	ユースケースを補う目的で、システムの動きを様々な粒度で表現する	任意
物理的	コンポーネント図	コンポーネントの構成や依存関係を表現する	設計
物理的	配置図	システム実行時のノートの構成、コンポーネントの配置を表現する	設計

UMLは「図」と「記号」を使ったわかりやすいものだといっても、前提知識のない人が見て直観的に理解できるものではない。実際の開発では、文章で書かれた仕様書を併用するなど、UMLのみに頼らない方法を考える必要がある。