第 17 章　オブジェクト指向技法

オブジェクト指向技法のスタート

我々が一般に持っている印象では、構造化技法はもうカビが生えるほど古いものであり、一方のオブジェクト指向技法は新しく、これから大きな花が開くソフトウェア開発方法論である、というものだろう。しかしこの両者が実は同い年であるということから、話を始めたい。

1968 年という年は、ソフトウェア工学の世界に 3 つほどの画期的なことが起きた年である。1 つ目は、当時の西ドイツで開かれた NATO 主催の「ソフトウェア・エンジニアリング会議」で、「ソフトウェア工学」と「ソフトウェア危機」という言葉が作られたことである[1]。2 つ目は、当時オランダの大学に勤務していたエズガー・ダイクストラ（Edsger W. Dijkstra）教授がアメリカのコンピュータ関係の学会誌（Communication of the ACM）の編集長に手紙を書き、「プログラムの中で GOTO ステートメントを使うことは良くない」と主張したことである[DIJ68]。編集長は彼の雑誌にこの手紙をそのまま掲載して、その後構造化プログラミングに、さらに構造化設計と構造化分析を包含した構造化技法に発展する議論のきっかけを作った[2]。

そして 3 つ目は、ノールウェーのクリステン・ニガード（Kristen Nygaard）とオーレ・ヨハン・ダール（Ole-Johan Dahl）の二人が Simula というコンピュータ・シミュレーション用言語を発表したことである。Simula は今でもヨーロッパでは汎用のプログラム言語として使われているとのことだが、この Simula が、オブジェクト指向プログラム言語の特徴であるクラスの考え方、継承などを既に持っていた。つまりオブジェクト指向技法は、ここからスタートしたということができる。したがって構造化技法とオブジェクト指向技法は、共に 1968 年生まれの同い年ということになる。

構造化技法は構造化プログラミングから 10 年以上をかけてゆっくりと上流に向かって進んでいったが、オブジェクト指向技法はこのオブジェクト指向プログラミングから、もっとゆっくりと上流に向かっていった。

オブジェクト指向ユーザ・インタフェース

ゼロックス社はシリコンバレーの中のパロアルトに、パロアルト・リサーチ・センター（Palo Alto Research Center：PARC）を持っている。この PARC はこれまでコンピュータや情報処理に関するいろんな分野で、すばらしい功績を挙げてきた。例えばいまでも LAN の代表選手であるイーサネットは、ここで開発された。イーサネットに加えてこの PARC のもう一つの功績に、オブジェクト指向ユーザ・インタフェースの開発がある。

この頃 PARC には、アラン・ケイを中心とするオブジェクト指向の言語や機器を開発するチームがあった。このチームは 1973 年に、アルト（Alto）と名付けたワークステーションの原型を開発した[KAY77]。そのユーザ・インタフェースを、当時は「オブジェクト指向ユーザ・インタフェース」と呼んだ。

このユーザ・インタフェースは、今はグラフィカル・ユーザ・インタフェース（GUI）と呼ばれていて、我々にとって、マイクロソフト社の Windows やアップルのマッキントッシュ（mackintosh）で、すでにおなじみのものである。

データやプログラムをアイコンで表して、それをクリックすることでそのデータの処理をス

---

[1] ソフトウェア・エンジニアリング会議については、第 1 章で述べた。

[2] 構造化技法については、第 15 章で述べた。

タートさせたり、プログラムを稼働させたりする方式は、まさにオブジェクト指向の方式そのものであるというところから、この名前が付けられた。

1973 年に完成した後 Alto のユーザ・インタフェースは、しばらくの間は特に誰から注目されることもなく、PARC の片隅に眠っていた。ところが 1980 年に当時アップルの CEO だったスティーブ・ジョブス（Steve Jobs）が PARC を訪れて、このユーザ・インタフェースを見て一目惚れをしてしまった。

アップルに戻ったジョブスは、早速このユーザ・インタフェースを当時アップルで開発していたリサ（Lisa）というコンピュータのユーザ・インタフェースに取り込んだ。しかしリサは高価な上に処理速度が遅く、売れ行きはもう一つだった。その後アップルはマッキントッシュを開発し、そのユーザ・インタフェースに Alto のユーザ・インタフェースを再び取り込んで発売した。これが、大当たりした。

マイクロソフトもこのユーザ･インタフェースに似たものを､1985 年に発表した Windows1.0 で実現した。しかしマイクロソフトの Windows が評判になったのは、米国では 1991 年に発売された Windows3.0 から、日本では 1994 年の Windows3.1 からである。

スティーブ・ジョブスはマイクロソフトを、Windows がマッキントッシュのユーザ・インタフェースを盗んだとして、著作権侵害で訴えた。しかしビル・ゲーツは、Windows のユーザ・インタフェースのオリジナルは Alto であると主張し、これが裁判所に認められた。つまりマッキントッシュも Windows も、ともに Alto を親とする兄弟であるというわけである。

ちなみに PARC はオブジェクト指向プログラム言語として、SmallTalk も開発した。SmallTalk にはいくつかの版があるが、1980 年に開発された SmallTalk80 が、今の最新の言語仕様である。

オブジェクト指向データベース

コンピュータの中でプログラムが稼働している時、コンピュータの中にデータが作られる。このデータは、何もしなければそのプログラムが処理を終了した時、コンピュータの中から消えてしまう｡それでは困る場合､プログラムはそのデータをファイルやデータベースに書いて、次の使用に備える。

このプログラムがオブジェクト指向プログラム言語で書かれたものであっても、状況は変わらない。オブジェクト指向のプログラムがコンピュータの中に作り出すものを、オブジェクトと呼ぶ。そしてこのオブジェクトを記録するデータベースを、オブジェクト指向データベースと呼ぶ。最初のオブジェクト指向データベースは、1987 年に発表された ONTOS と呼ばれるものだった。

オブジェクト指向データベースは一時期、そのうちすぐにリレーショナル・データベースに取って代わるのではないかと期待されていた。しかしオブジェクト指向の要素を随分取り入れてはいるものの、未だにリレーショナル・データベースの時代が続いている。

オブジェクト指向分析と設計

オブジェクト指向の考え方は後で述べるが、それをプログラムの設計や情報システムの分析に使用しようという機運が生まれた。そのきっかけになったのは、1986 年にグラディ・ブーチが書いた論文[BOO86]である。

その後はまさに雨後の竹の子のように、いくつものオブジェクト指向による分析と設計の方

法論が発表された。その多くのものは提唱者の名前をそのまま付けて、例えばブーチ法（グラディ・ブーチ（Grady Booch）が提唱したもの）[BOO94]、シュライアー・メーラ法（シュライアーとメーラが提唱したもの)、コード・ヨードン法（コードと、エドワード・ヨードンが提唱したもの）などと呼ばれた。個人名がついていないものには、ジェームズ・ランボー（James Rumbaugh）などのグループの OMT（Object Management Technology）[RUM91]、イアン・ヤコブソン(Ivar Jacobson)などのグループの OOSE(Object Oriented Software Engineering)[JAC92]などがある。それぞれの方法論ごとに、モデルの表現方法、開発の方法などが異なっていた。

このような状況を受けて、毎年秋に IEEE のコンピュータ・ソサイエティと ACM が共同開催している OOPSLA（Object-Oriented Programming, Systems, Languages, and Applications）という国際学会の主催者が、1992 年秋の OOPSLA でそれぞれの方法論の紹介を行うことを考えついた。ただしそれぞれの方法論の紹介者はその提唱者ではなく、必ず別の人に当てることにした。つまりこの会議で、それぞれの方法論の善し悪しについて、大論争が起きるだろうと期待された。西部開拓時代の有名な「OK 牧場の決闘」[3]にちなんで、「OO 牧場の決闘」が起きると期待された。

しかしこの「OO 牧場の決闘」は、結果として起きなかった。むしろ発表者たちは、個々の方法論にはその違いよりも共通するところの方がはるかに多く、大同団結してオブジェクト指向技法をより良い方向に進め、普及させることの方が、お互いに利益が大きいことを認識した。

UML と RUP

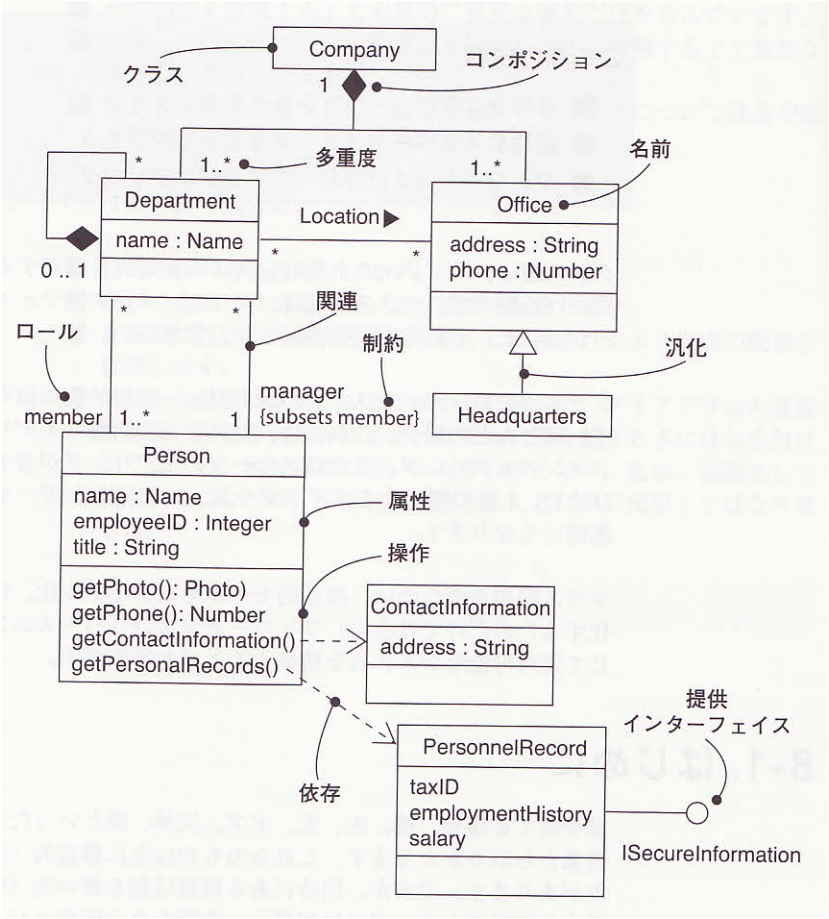
アメリカに、オブジェクト・マネジメント・グループ（Object Managing Group：OMG）という団体がある。非営利の民間団体で、オブジェクト指向技法の普及・拡大と定着をその目的としたもので、これまでに CORBA（Common Object Request Broker Architecture）の仕様などを発表して、大きな効果を挙げている。

上で述べたようなオブジェクト指向技法改善の風潮を受けて、この OMG がオブジェクト指向によるソフトウェアの開発方法の統一に乗り出すことになった。しかし結果として開発方法の統一は難しく、その手前にあるモデルの表記法の統一から着手するのが現実的という結論になった。

それで OMG はモデルの記述方法の統一について、推進者を公募した。ブーチは当初からラショナル社に勤務していたが、ジェネラル・エレクトリック社に勤務していたジェームズ・ランボーがまずラショナル社に移り、さらにスエーデンのエリクソン社を退職して自分で会社を運営していたイアン・ヤコブソンもその後でラショナル社に移って、ラショナル社からこの 3 人が共同でモデルの記述方法統一を進めるという提案があった。OMG はこれを承認し、この 3 人が進めるモデル記述方法の統一を OMG としての正式の活動と位置づけた。この結果作成されたものがユニファイド・モデリング・ランゲージ（Unified Modeling Language：UML）

[3] OK 牧場の決闘とは、アリゾナ州ツムストン近くにあった OK 牧場で、名保安官の誉れが高いワイアット・アープとクラントン兄弟などとの間で 1881 年 10 月 26 日朝に起きた銃による決闘である。ジョン・フォード監督、ヘンリ・フォンダ主演の「荒野の決闘」と、ジョン・スタージェス監督、バート・ランカスター主演の「OK 牧場の決闘」の 2 つの名作映画が、いずれもこの決闘をテーマにして作られている。

である[4]。UML は 1995 年に最初のバージョンが発表され、2005 年 7 月にバージョン 2.0 が発表された[5]。この原稿を修正している時点（2017 年 1 月）での最新バージョンは、UML2.4.1 である[6]。UML からの 1 つのサンプルとして、クラス図を図表 17-1 に示す。



図表 17-1　クラス図の例（[BOO05]より）

一方正式には一度ギヴ・アップした開発方法の統一についても、この 3 人を核にしたラショナル社の技術者たちが 1 つの方法を編み出した。それはラショナル統一プロセス（Rational Unified Process：RUP）と呼ばれている[KRU03]。

ちなみに、グラディ・ブーチ（Grady Booch）、ジェームズ・ランボー（James Rumbaugh）、アイバー・ヤコブソン（Ivar Jacobson）の 3 人は「3 人の偉人（Three Amigos）」と呼ばれている。また米国のラショナル社は 2002 年 12 月に IBM 社に買収されて、今は IBM 社の一部になっている。

---

[4] UML については、第 22 章で述べる。

[5] UML に関する資料等は、次の URL からダウンロードできる。
http://www.uml.org/

[6] UML 2.4.1 は ISO と IEC で規格化されて、ISO／IEC 19505-1：2012 と ISO／IEC 19505-2：2012 になっている。

Java

オブジェクト指向プログラム言語として、Smalltalk に加えて、いくつかの既存の言語にオブジェクト指向の処理方式を取り入れたものがある。その中で著名なものに、C 言語をベースにした C++がある。

その後 1992 年 9 月に、サン・ソフトウェアが Java を開発して発表した。当初の Java はインターネットのブラウザの上で動く小さなプログラム（「アップレット」と呼ばれる）開発用のものといっても良かった。しかしその後各種の拡張と整備が進み、今ではオブジェクト指向プログラム言語の代表格になっている。

Java には、1 つ大きな特徴がある。Java のソース・プログラムはコンパイルされて、「バイト・コード」と呼ばれる特種なコードに変換される。このバイト・コードは、Java 仮想コンピュータ（JavaVM）と呼ばれる特種な稼働環境の下で、一種のインタープリタ用の言語として稼働することができる。この JavaVM は、現実の多くのプラットフォーム（ハードウェアとオペレーティング・システムの組み合わせ）の下で稼働するように準備されており、これによって Java はあるコンピュータで一度コンパイルされると、後は JavaVM が用意されている任意のプラットフォームの下で稼働することができる。したがってインターネットのブラウザが JavaVM を内蔵していると、既に用意されたバイト・コードをダウンロードするだけで、どのブラウザの上でも Java で作った小さなプログラムが動くということが実現した。

Java は、純粋にプログラム言語としての側面だけを見ると、明快な、平易なプログラム言語といって良い。別のところで既に述べたが、「GOTO ステートメント」がその語彙に含まれていない数少ないプログラム言語の 1 つでもある。したがって構造化プログラミングの方式に従って、容易にプログラミングできる。しかし Java を使ってオブジェクト指向プログラムを作成しようとすると、内部にあらかじめ用意されているオブジェクトからの継承などを十分に理解しなければ対処できず、突然難解なものに変わる。これが Java のもう 1 つの特徴ともいえるものである[ARN00]。

オブジェクト指向技法の考え方

オブジェクト指向技法をよく知っている人から見ると異論があるかもしれないが、私から見るとオブジェクト指向技法の考え方は、「人を中心とした実社会で行われている仕事の進め方を、コンピュータの内部でも実現しようとするもの」のように見える。つまり実社会のオブジェクトをソフトウェアの中にも作って、実社会をソフトウェアの世界に移し替えようとするもののように見える。

もちろんコンピュータの世界は実社会に比べるとはるかに簡単で、その面での制約がある。しかしもしこれが実現するなら、情報システムの構築に当たって、より素直に対応できるようになるのかもしれない。

オブジェクト指向技法のキーワード

オブジェクト指向技法には、次の 10 個のキィワードがある。[TAY97]。

- オブジェクト
- クラス
- クラス体系、スーパー・クラス、サブ・クラス
- インスタンス

- 継承（インヘリタンス）
- 多相性（ポリモーフィズム）
- 情報隠蔽（インフォメーション・ハイディング）
- カプセル化（エンキャプシュレーション）
- メッセージ
- メソッド

以下でこれらのキーワードについて、簡単に触れておきたい。

よく「オブジェクト指向は難しい」といわれる。その理由としていくつかを挙げることができるが、その 1 つは「オブジェクト」と「クラス」の概念に混同があるということにある。

「オブジェクト」をオブジェクト指向技法について書いた本を紐解いて調べてみると、「オブジェクトとは『データ』とそれに対する『処理』を一体にしたもの」で、「『クラス』と同じ」と書いてある。これだけではよく分からないのでもう一段調べると、「『クラス』に属する個々の存在（インスタンス)」と書いてあり、それで「クラス」を調べると「『オブジェクト』の集合体（オブジェクトのテンプレート)」という言葉が見つかる。

ここで、不幸なことに無限ループに陥る。つまり「オブジェクト」が分からなければ「クラス」が分からず、「クラス」が分からなければ「オブジェクト」が分からないことになる。

そうなら、次のように割り切ってみると良いかもしれない。つまり「クラス」とは、今話題にしている情報システムで「重要なデータ」である。データ中心アプローチで「実体」と「関連」と呼んだものが、全部クラスになる。だから「学生」はクラスであり、その個々の存在である A さん、B さんが「オブジェクト」である。しかしオブジェクト志向の書籍で、「クラス」というべきところをよく「オブジェクト」と書かれているので、注意が必要である。この場合「オブジェクト」が A さん、B さんを意味していない時、「オブジェクト」という言葉を自動的に「クラス」に置き換えて見ると、理解が容易になるかもしれない。

構造化技法時代の「実体」や「関連」は、全て独立した存在だった。例えば実体には、親もいなければ、兄弟もいなかった。しかし「クラス」は、そうではない。親もいれば、兄弟も子供もいる。例えば「自動車」というクラスの子供には、「トラック」というクラスや「乗用車」というクラスがいる。「乗用車」の下には「セダン」や「クーペ」、「ワゴン」がある、というわけである。

親クラスと子クラスの間には、「is a」の関係と「part of」の関係がある。「is a」の関係では、「乗用車 is a 自動車」となり、「part of」の関係では、「エンジン is a part of 自動車」となる。

この「is a」と「part of」の両方の関係を含めてクラス間の親子の関係を、「クラス体系」と呼び、親のクラスを「スーパー・クラス」、子のクラスを「サブ・クラス」と呼ぶ。サブ・クラスの親が常に 1 つしかない場合、その構造を階層構造といい、複数の親を持って良い場合は、ネットワーク構造という。

これを図に描くと「クラス図」になる。クラス図は「実体関連図」に近いが、クラス体系を書きこむことができる分だけ図に深みができる。

クラス体系のクラス間の関係が「is a」の場合、親のスーパー・クラスの性格や振る舞いを子供のサブ・クラスが受け継ぐことができる。これを「継承（インヘリタンス、inheritance)」という。階層構造の場合常にスーパー・クラスは 1 つしかないので継承は「単一継承」となる。一方ネットワーク構造の場合は親が複数いるので、継承は「複合継承」になる。複合継承の場

合、それぞれの親から継承を受けることができる。ただし A の親から受ける継承と B の親からのものが矛盾する場合どちらの親からのものを受け入れるのかを宣言することで、この問題を解決している。プログラム上で実際に継承の対象になるものは、データ構造とプログラムである。

「図形」というスーパー・クラスがあって、そのサブ・クラスに「円」、「三画」、「四角」などがあり、さらにスーパー・クラスに「描く」という振る舞いが定義されているとする。この「描く」という振る舞いはそれぞれのサブ・クラスに継承されていて、「円」のサブ・クラスはこの「描く」という振る舞いによって円を描き、「四角」は四角を書くとする。

このような場合「図形」のサブ・クラスに「描く」という同じ振る舞いをさせても、それぞれのサブ・クラスで行うことが違ってくる。これを「多相性（polymorphism）」と呼ぶ。

またオブジェクトは、重要でないものは後で述べる「カプセル化」によってオブジェクトの内部に格納してしまい、外部に公開しないという性格を持っている。逆に外部に公開する重要なものは、オブジェクトが持っている機能と、その機能を働かせるためにどのように処理を依頼すればよいのかというインタフェースの情報だけである。この性格を「情報隠蔽（information hiding）」とよぶ。

「カプセル化（encapsulation）」は、前述のようにオブジェクトの内部にデータとそれを処理するプログラムを格納する、情報隠蔽を実現する手段である。

さらに、オブジェクトに対する処理の依頼を「メッセージ」と呼び、処理を依頼することを「メッセージをパスする」とか「リクエストする」とかという。そしてメッセージによって起動されるプログラムのことを、「メソッド」とよぶ。

オブジェクト指向技法の特徴

前にも書いたが、オブジェクト指向技法は一般に難しいといわれている。その難しさの 1 つの理由として、重要なキーワードである「オブジェクト」と「クラス」の言葉の上での混同を挙げた。それ以外に、さらにキーワードに難解な言葉が並んでいる。例えば「inheritance（継承）」とか「polymorphism（多相性）」とかという言葉を、私は他のところで見たことがない。さらに「メッセージ」という言葉も「メソッド」という言葉も、言葉そのものは難しくないけれど、ここで使われているような意味で使われているのを見たことがない。

クラスを実体や関連と同じものと考えると少し気が楽になるが、しかしデータ中心アプローチの実体や関連はデータだけを持っており、それを処理するプログラムは別のところにある。これが一体となってオブジェクト（クラス）を構成しているというところに、データ中心アプローチで育った人にも、大きな違和感があるだろう。

構造化技法では、モジュール間の結合度は弱い方が好ましいとされてきた。しかしオブジェクト指向技法で親子関係にある 2 つ以上のクラスにデータ構造やプログラムの継承があると、2 つ以上のクラスが渾然一体となって区別がつかないような状態になる。クラス間の結合度を構造化技法の立場から見れば、最悪の状態である。

またクラスの中のデータやプログラムに、private という外部からは見えない属性をセットすることができる。それはやはり情報隠蔽の 1 つの方式だが、構造化技法のテスト方法でいう「ホワイト・ボックス・テスト（またはグラス・ボックス・テスト）」の方式をこの private な属性を持つプログラムに適用するということができない。

モジュールとクラスは違うのかもしれないが、私は単純に構造化技法時代のソフトウェア工

学の成果は、オブジェクト指向技法の時代になってもそのまま受け継がれているものと期待していた。しかしこの期待は、どうも裏切られているらしい。ここには、注意が必要である。

オブジェクトの発見法

すぐにオブジェクト指向技法でソフトウェアを開発する場合の手順について述べるが、その中で最も重要な作業はこれから開発しようとする情報システムで、何が「オブジェクト」であるかを見つけ出すことである。これをエドワード・ヨードンは「オブジェクトの発見」と呼んでいる[YOU95]。

このオブジェクトの発見法に、いろんな人がいろんな方法を提案している。

ブーチは何がオブジェクトかという一覧を著書に掲載［BOO94］し、その類推でオブジェクトを発見するように提案している。レベッカ・ワーフブリックは、仕様書を読んで名詞の下にアンダーラインを引くと、そのアンダーラインが引かれた言葉が「オブジェクトの候補」になるので、その中からオブジェクトを発見するように提案している[WIR90]。

ヤコブソンはその著書[JAC92]の中で、オブジェクトには次の 3 つの種類があるといっている。

- コントロール・オブジェクト
- 実体オブジェクト
- インタフェース・オブジェクト

「クラスは実体と関連」と何度か書いたが、2 つ目の「実体オブジェクト」の発見には、データ中心アプローチのデータ分析の手順がそのまま使える[7]。インタフェース・オブジェクトを「出力」、コントロール・オブジェクトを「入力」と考えると、データ中心アプローチで育った人には、オブジェクトの発見はあるいは容易かもしれない。

いずれにしろオブジェクト指向技法による開発の経験を積んでくると、仕様書を読んだだけで、その情報システムでは何をオブジェクトにすると良いかが分かるようになる。

オブジェクト指向技法による開発手順

これまで述べたことと重複するが、オブジェクト指向技法で情報システムを開発する時の手順は、以下の通りである。

1. システムの特性から「オブジェクトを発見」する
2. そのオブジェクトを「性格」と「振る舞い」で整理して、クラス体系を作る
3. オブジェクトの「機能」を明確にする
4. 「メッセージ（処理の依頼の仕方）」を明確にする
5. 「メソッド（プログラム）」を設計する（この時に、併せてデータベースも設計する）
6. 「メソッド」を作成する
7. テストする

オブジェクト指向技法の将来

いくつかの難点はあるかもしれないが、しかし現時点では、オブジェクト指向技法はソフトウェアの開発方法論の本命である。ビジネス・アプリケーションではこれに、データ中心アプローチでのデータベースの設計がプラスされる。21 世紀のソフトウェア技術者やそれを目指す

[7] データ分析の手順については、第 16 章で述べた。

学生諸君は、まずオブジェクト指向技法を理解し、UML を使いこなし、Java などのオブジェクト指向言語で自由にプログラミングできるようになることが必要である。

キィワード

オブジェクト指向技法、パロアルト・リサーチ・センター、PARC、オブジェクト指向ユーザ・インタフェース、グラフィカル・ユーザ・インタフェース、オブジェクト指向プログラム言語、SmallTalk80、オブジェクト指向データベース、オブジェクト・マネジメント・グループ、OMG、ユニファイド・モデリング・ランゲージ、UML、ラショナル統一プロセス、RUP、C++、Java、オブジェクト、クラス、インスタンス、「is a」の関係、「part of」の関係、クラス体系、スーパー・クラス、サブ・クラス、クラス図、継承、inheritance、多相性、polymorphism、情報隠蔽、カプセル化、メッセージ、メソッド、Three Amigos

略語

PARC：Palo Alto Research Center
OOPSLA：Object-Oriented Programming, Systems, Languages, and Applications
OMG：Object Managing Group
CORBA：Common Object Request Broker Architecture
UML：Unified Modeling Language
RUP：Rational Unified Process

人名

クリステン・ニガード（Kristen Nygaard）、オーレ・ヨハン・ダール（Ole-Johan Dahl）、アラン・ケイ（Alan Key）、スティーブ・ジョブス（Steve Jobs）、グラディ・ブーチ（Grady Booch）、ジェームズ・ランボー（James Rumbaugh）、アイバー・ヤコブソン（Ivar Jacobson）

規格

UML2.4.1、ISO／IEC 19505-1：2012、

参考文献とリンク先

[ARN00] Ken Arnold 他著、柴田芳樹訳、「プログラミング言語 Java 第 3 版」、ピアソン・エデュケーション、2001 年.
この本の原書は、以下のものである。
Ken Arnold, James Gosling, David Holmes, “The Java Programming Language, Third Edition,” Addison Wesley Longman, 2000.
[BOO86] Grady Booch, “Object-Oriented Development,” IEEE Transactions on Software Engineering, Vol.12, No.2, IEEE, Feb. 1986.
[BOO94] Grady Booch 著、山崎明宏他訳、「Booch 法：オブジェクト指向分析と設計 第 2 版」、星雲社、1995 年.
この本の原書は、以下のものである。
Grady Booch, “Object-Oriented Analysis and Design with Applications Second Edition,” The Benjamin/Cummings Publishing Co. 1994.

[BOO05] グラディ・ブーチ他著、羽生田栄一監訳、越智典子役、「UML ユーザガイド 第 2 版」、ピアソン・エデュケーション、2010 年.
この原書は、以下のものである。
Grady Booch, James Rumbaugh, Ivar Jacobson, “The Unified Modeling Language Use Guide Second Edition. “Addison-Wesley, 2005.

[DIJ68] Edsger W. Dijkstra, “Go To Statement Considered Harmful,” Communication of the ACM, Vol.11, No.3, ACM, March, 1968.
このペーパーは、ACM のデジタル・ライブラリ（http://portal.acm.org/portal.cfm）からダウンロードすることができる。（ただし、ACM のメンバーであることが必要。）

[JAC92] I. ヤコブソン他著、西岡利博他監訳、「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-」、アジソン・ウェスレイ、1995 年.
この本の原書は、以下のものである。
Ivar Jacobson, Magnus Christerson, Patric Jonsson, Gunner Overgaard, “Object-Oriented Software Engineering　A Use Case Driven Approach,” ACM Press, 1992.

[KAY77] Alan Kay, Adele Goldberg, “Personal Dynamic Media,” IEEE Computer, Vol.10, No.3, pp31-41, IEEE, 1977.
なおこのペーパーは、次の本に収録されている。
アラン・ケイ著、浜野保樹監訳、鶴岡雄二訳、「アラン・ケイ」、アスキー、1992 年.

[KRU03] フィリップ・クルーシュテン著、藤井拓監訳、「ラショナル統一プロセス 第 3 版」、アスキー、2004 年.
この本の原書は、以下のものである。
Philippe Kruchten, ‘The Rational Unified Process - An Introduction, Third Edition,” Addison Wesley Professional, 2004.

[RUM91] J. ランボー他著、羽生田栄一監訳、「オブジェクト指向方法論 OMT モデル化と設計」、トッパン、1992 年.
この本の原書は、以下のものである。
James Rumbaugh, “Object Oriented Modeling and Design,” Prentice Hall, 1991.

[TAY97] David A. Taylor, “Object Technology A Manager’s Guide Second Edition,” Addison-Wesley, 1997.
この第 2 版の日本語訳は出版されていない。しかしこの本の最初の版の日本語訳はすばらしい本なので、以下に紹介しておきたい。
デビット A. テイラー著、増永良文監訳、寺島哲史訳、「オブジェクト指向アプローチ －その全貌－」、アジソン・ウェスレイ、1993 年.

[WIR90] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener, “Designing Object-Oriented Software,” Prentice Hall, 1990.

[YOU94] E. ヨードン著、松原友夫訳、「オブジェクト指向システム設計 新たな方法論の統合を提唱する」、プレンティスホール・トッパン、1995 年.
この本の原書は、以下のものである。
Edward Yourdon, “Object Oriented Systems Design An Integrated Approach,” Prentice Hall, 1994.

（2007 年（平成 19 年）6 月 5 日　初稿作成）
（2016 年（平成 28 年）4 月 14 日　一部追加）
（2017 年（平成 29 年）1 月 12 日　一部修正）