

オブジェクト指向は難しくない 第2回 オブジェクト指向技術を分類/整理する

(日経 BP 日経ソフトウェア2001年1月号より2002年1月号に
「オブジェクト指向は難しくない」として連載された)

[平澤 章]

仕様や製品を支える四つの技

“わかる”オブジェクト指向を目指したこの連載。いよいよ2回目です。今回は、オブジェクト指向についての素朴な疑問を五つ取り上げて、「オブジェクト指向とは何か」を考えてみました。「高品質で、わかりやすく、保守しやすいソフトウェアを作るための技術やノウハウの総称がオブジェクト指向だ」という全体イメージはつかめてもらえたでしょうか？皆さんの中には「イメージは何となくつかめたけど、具体的な技術やノウハウにはどんなものがあるの？」という次の疑問を抱いた人もおられるかもしれませんね。

そこで今回は、オブジェクト指向の全体像をもう少し具体的に眺めてみましょう。まずオブジェクト指向技術と一くりにされている技術や製品を、四つに分類/整理します。そのうえで、最も重要な中核技術について構成要素を紹介しましょう。「オブジェクト指向にはどんな技術や製品があって、それらがどんな関係にあるのか、さっぱりわからない」なんていう人も、これでスッキリすると思います。

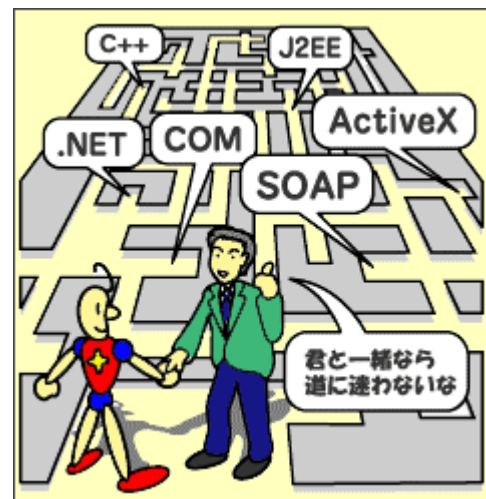
10年たっても基本はほとんど変わらない

コンピュータ技術の歴史を振り返ると、それこそ星の数ほどの製品や技術が次々と登場してきました。そして、それと同じぐらいスピードで、多くの製品や技術が陳腐化し、消えていきました。しかも、現在主流となっている技術であっても、いつまでその座を占め続けられるかはまったくわかりません。

こうした状況は、非常に新鮮で刺激的ではありますが、一方でせつかく身に付けた技術がすぐに陳腐化してしまうのですから少々困ったものです。陳腐化が繰り返されると、そのうち「新しい技術を覚える」という意欲そのものが失せてしまいます。実際、筆者自身も、覚えた技術が陳腐化してしまう悲哀を過去に何度も味わいました。

では、現在持てはやされているオブジェクト指向技術も、“歴史は繰り返す”のことわざ通り、すぐに陳腐化していつてしまうのでしょうか？筆者の考えは「ノー」です。なぜなら、オブジェクト指向技術の本質的な部分は10年前からほとんど変わっ

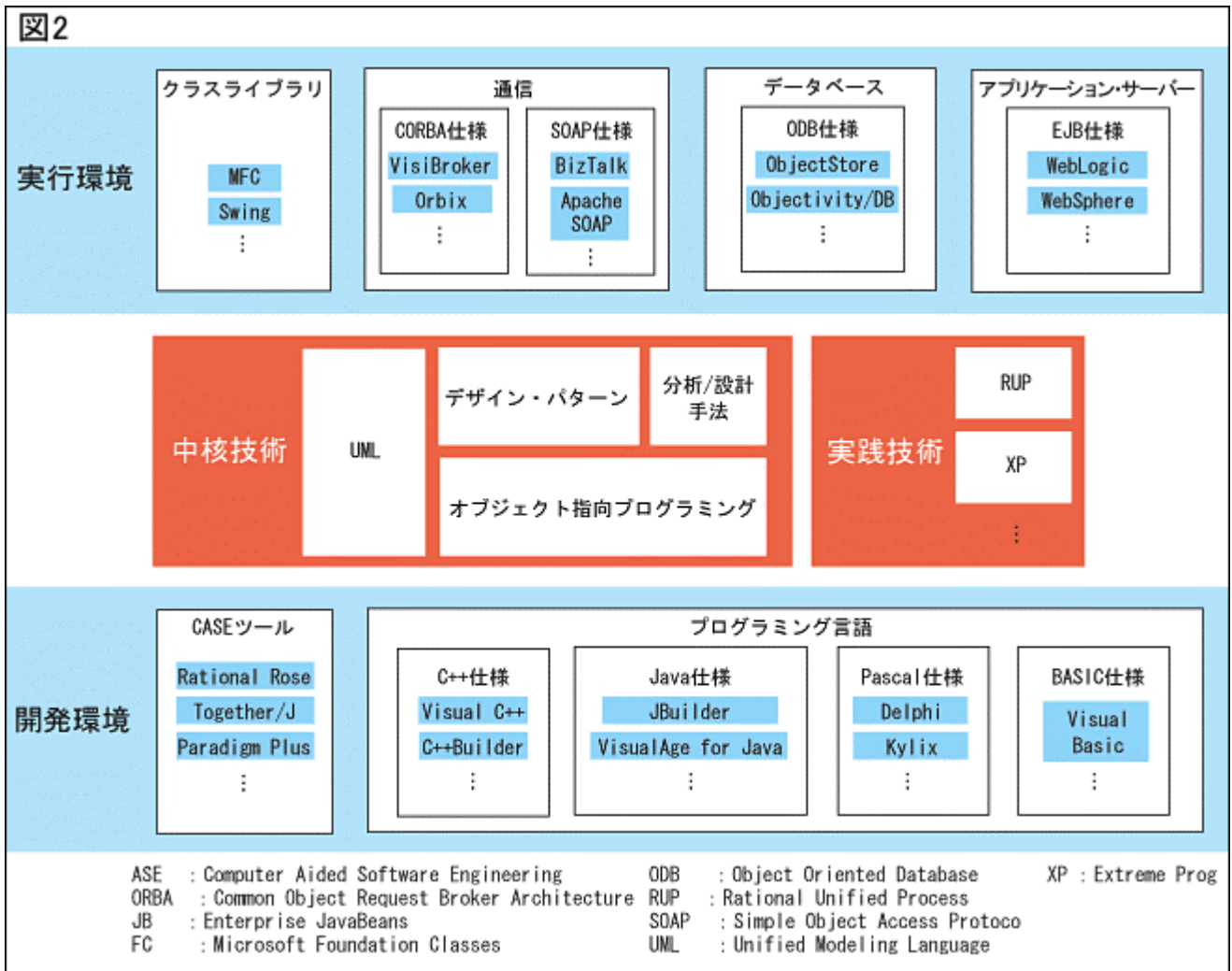
ていないからです。現在 Java が備えている機能は、10年前にすでに実現されていました¹。汎用モデリング言語 UML (Unified Modeling Language : 詳細は後述) も、その前身となる開発方法論が発表されておりました²、デザイン・パターン (詳細は後述) のアイデアも10年前に発表されておりました。製品やツールだけを見ていると、激しい競争の中でオブジェクト指向技術も変わってきたように思えますが、実は変化しているのは実装のほうだけで、オブジェクト指向技術そのものはほとんど変わっていないのです。さらに、次の10年でオブジェクト指向技術が大きく変わってしまう可能性もありません³。「オブジェクト指向技術は、この10年で開花し、次の10年もまだ現役でいられる」、筆者はそう確信しています。したがって、オブジェクト指向技術を理解していれば、コンピュータ技術の目まぐるしい変化も怖くはありません (図1)。



¹ 1970年代に登場した Smalltalk や、1980年代に登場した C++ はすでに、オブジェクト指向言語の三大要素と言われるクラス (カプセル化)、継承、ポリモーフィズムを備えていた。

² UML の前身である OMT 法 (Object Modeling Technique) や Booch 法 (名称は考案者の Grady Booch 氏に由来) などが発表されていた。

³ オブジェクト指向の次に来る技術は、エージェント指向、アスペクト指向などと言われている。しかし、実用段階に入るのはかなり先のことになるとの見方が多い。仮に実用化されても、それらの技術はオブジェクト指向の延長として登場すると思われる。



中核/実行/開発/実践の四つに分類しよう

では、オブジェクト指向技術のうち、「何が変化し」「何が変化しない」のでしょうか。それを把握するために筆者がオススメしたいのが、オブジェクト指向技術の全体像を「中核技術」「実行環境」「開発環境」「実践技術」の大きく四つに分類することです(図2)。変化しない(変化の少ない)技術は、「中核技術」ですね。一方、「実行環境」「開発環境」「実践技術」の三つは、実装をともなうために変化しやすいと言えます。もし「オブジェクト指向技術を採用した～」というフレーズの技術や製品が新たに登場したら、この分類のどこに当てはまるのかを考えてみましょう。そうすることで、その技術/製品の位置付けが理解できるはずですよ。

1) 中核技術

中核技術に含まれるのは、オブジェクト指向プログラミング、UML、デザイン・パターン、分析/設計手法の四つです。これらは現在主流となっている多くの製品や標準仕様のもとになっており、今後も大きく変わることはないでしょう。この連載でも、中核技術を重点的に解説していきます。

2) 実行環境

実行環境に含まれるのは、ソフトウェアの動作に必要な基本ソフトウェアやミドルウェアです。具体的には、MFC⁴やSwing⁵などのクラスライブラリ、米 Borland Softwareの VisiBroker、Microsoftの

⁴ MFC (Microsoft Foundation Classes) は、米 Microsoft が提供する、Windows アプリケーションを開発するための C++用クラスライブラリ。

⁵ Swingは、Javaアプリケーションを開発するためのグラフィックス用クラスライブラリ。米 Sun Microsystems が提供する。

BizTalk Server といった、CORBA⁶や SOAP⁷仕様の通信機能を備えた製品、米 eXcelon の ObjectStore など ODB⁸に代表されるデータベース、米 BEA Systems の WebLogic など EJB⁹仕様のアプリケーション・サーバーが挙げられます。これらのソフトウェアは、オブジェクト指向言語で書かれることが多く、また、何らかの形でオブジェクト指向のインタフェースが用意されています。

この領域は技術の進歩が速く、かつベンダー間の競争が激しいので、製品はそれぞれ雨後のタケノコのように次々と登場しています。同じ分野に複数の標準仕様が存在することが珍しくなく、その標準仕様も次々とバージョンアップされています。

3) 開発環境

実行環境と同様に、たくさんの製品が存在するのが開発環境です。プログラミング言語では、現在は Java や C++、Visual Basic (VB)¹⁰が主流ですが、Microsoft の C#¹¹のような新しい言語も登場しています。言語仕様についても、C++は ANSI 標準がありますが、Java や VB のように提供ベンダーが仕様を決めているものもあります。

CASE ツール¹²は、ツールの価格が比較的高いことなどから、現時点ではさほど普及していません。しかし UML で描く図(ダイアグラム)を、いちいち紙に手で書いていたのでは手間がかかりますし、修正も大変です。オブジェクト指向技術を使ったシステム開発が一般的になれば、UML の利用も拡大するでしょうから、CASE ツールはもっと安価になり、広く利用されるようになるでしょう。

⁶ CORBA (Common Object Request Broker Architecture) は、標準化団体 OMG (Object Management Group) が定めた、分散オブジェクト通信技術の仕様。

⁷ SOAP (Simple Object Access Protocol) は、XML (Extensible Markup Language) ベースの通信プロトコル。

⁸ ODB (Object Oriented Database: オブジェクト指向データベース) は、オブジェクト指向技術を使ったデータベース。

⁹ EJB (Enterprise JavaBeans) は、サーバーで利用される Java 部品プログラムの仕様。

¹⁰ 前回の繰り返しになるが、筆者は Visual Basic (VB) をオブジェクト指向言語と考えている。現在の VB 6.0 でも、クラスやポリモーフィズムが実現されているし、次期版にあたる Visual Basic.NET では、Java に引けを取らないオブジェクト指向言語に進化することが明らかになっているからだ。

¹¹ C# は、Microsoft が開発したオブジェクト指向言語。同社の開発スイートの新版 Visual Studio.NET に開発環境が実装される予定。

¹² CASE (Computer Aided Software Engineering) ツールは、システム構築の効率化を支援するソフトウェア。

4) 実践技術

実践技術とは、実際のソフトウェア開発をどのように進めていくと効果的かという運用方法やルールをまとめたものです。従来も、各組織やプロジェクトの単位で、開発標準としていろいろな決めごとや工夫をしてきましたが、最近では業界全体での統一や、優れたノウハウを共有しようという動きが出ています。

特に従来から行われてきた、いわゆるウォーターフォール型開発¹³には、批判が強くなっています。開発の各フェーズごとに仕様書をまとめたり、成果物を引き継ぐ作業によって、システムを実際に稼働させるまでに長い時間がかかることが多く、仕様が陳腐化しやすいからです。そこで最近では、要求定義、分析/設計、実装、テストの各作業のサイクルを複数回実施し、ソフトウェアの機能を段階的に発展させていく、繰り返し型開発¹⁴と呼ばれる開発手法へのニーズが高くなっています。繰り返し型開発手法では、いったん作ったソフトウェアに対しても必要に応じて修正や拡張を行います。オブジェクト指向は、保守性を高くできるため、こうした開発手法も採用しやすいわけです。

OOP を理解することから始めよう

さて、オブジェクト指向技術の四つの分類がイメージできたところで、最も重要な「中核技術」について、少し詳しく見て行きましょう。最初は「オブジェクト指向プログラミング (Object Oriented Programming: OOP)」です。

オブジェクト指向技術の基本は、何といたってもプログラミングです。ソフトウェアは最終的にプログラミング言語を使って記述するのですから、これをきちんと理解せずして、ソフトウェアを語ることはできません。おや、「プログラムなら知ってるさ。自分はこれまで COBOL や C で何万行もプログラミングをしてきたからね」という声も聞こえてきそうですね。でもちょっと待ってください。

オブジェクト指向プログラミングとは、言い換えれば、「オブジェクト指向言語」を使ってプログラムを作成することです。したがって、オブジェクト指向言語の特徴を理解しなければ、オブジェクト指

¹³ ウォーターフォール型開発は、要求定義、分析/設計、実装、テストの各段階を確実に 1 回だけ実施する開発手法。水の流れ (waterfall) のように後戻りがないことからこの名前がある。

¹⁴ スパイラル型開発とも呼ばれる。

向技術の効果は出ません。オブジェクト指向言語には、それまでの C や COBOL といった言語¹⁵には存在しなかった重要な機能があるからです。

それは「クラス(カプセル化)」「継承」「ポリモーフィズム¹⁶」と呼ばれる三つの機能です。これらが具体的にどのようなものなのかは次回以降で詳しく説明しますが、これらをまとめてオブジェクト指向言語の三大要素と呼ぶこともあります。オブジェクト指向以前の言語、例えば C 言語では、ソフトウェアの最小単位は「関数」でした。しかしオブジェクト指向言語では、結びつきの強い関数とデータを一つにまとめた「クラス」が最小単位になります。このとき、クラスの中の情報に対して外部から直接アクセスできないようにすることを「カプセル化」と呼びます。またオブジェクト指向言語には、似たようなロジックを一つにまとめる「継承」や「ポリモーフィズム」といった仕組みも用意されています。

要するにオブジェクト指向言語には、ソフトウェアを整理整頓して、ムダを徹底的に省くための仕組みが備わっている、と考えればいいでしょう(図 3)。



フレームワークやコンポーネントと呼ばれる、より大きなソフトウェア部品を作り上げることができる点も、オブジェクト指向言語ならではの特徴です。こうした芸当は従来のプログラミング言語ではちょっと真似ができません。したがって、C や COBOL をきちんと理解している人でも、一度はオブジェク

¹⁵ オブジェクト指向以前の言語は「手続き型言語」と呼ばれることもある。

¹⁶ 日本語では「多態性」「多相性」と呼ばれる。

ト指向言語を使ってオブジェクト指向プログラミングを体感することを強くオススメします。

ただし勘違いしないでほしいのは、言語仕様を覚えることが必ずしも重要ではない、ということです。大切なのは、オブジェクト指向言語の備える機能の意味や役割、目的を理解することです。それさえわかれば、C++から Java に変わったとしても、あるいは C#や Visual Basic.NET が今後の主流になったとしても、何も恐れることはありません。

UML はソフトの構造を図で表現

中核技術の中で、オブジェクト指向プログラミングの次に重要なのは何でしょうか。筆者は UML だと考えています。UML は、ソフトウェアの構造を図式表現するための表記法で、標準化団体の米 OMG (Object Management Group) が 1997 年に標準として認定しています。Language (言語) と言っても、C や Java のようなプログラミング言語ではありません。要するに、ソフトウェアの構造を図(ダイアグラム)を使って描くための決まりごとです。例えば UML を使ってクラスを設計するときには「クラス図」を描きます(図 4)。

もともと UML は建築図面や電気配線図などのように、ソフトウェアの“設計図”も描き方を統一しよう、ということから生まれました。「ソフトウェアの構造を図で表現するってどういうこと?」と思う人もいるでしょうが、そんなに難しいことはありません。私たちが何か複雑なことを相手に伝えようとするときには、紙やホワイトボードに図を描いて説明しますよね。人間が物ごとを直感的に理解しようとするとき、簡潔にしかも雄弁に物事を語ってくれるという点で、図は非常に有効な表現方法です。

「このプログラムの中はどうなっているの?」と思ったとき、「コードをすべて読んでください」と言われたらどうでしょう。短いコードならまだしも、現実に動いている何千、何万行にもなるソフトウェアであれば、コードをすべて読むのは実際には不可能です。そんなときに UML を使って表記された図を見ることができれば、何千行にもなるソフトウェアの構造をコードを読まずに理解することが可能になります。

「でも図の描き方を覚えるなんて面倒だな」と思った人もいそうですね。心配はいりません。UML には、クラス図のほかに、フローチャートによく似たアクティビティ図や、制御系のソフトウェアでよく使う状態図やシーケンス図など、従来使われてきた図式表現も取り込まれています。ソフトウェア開発

にかかわってきた方なら、どこかで見た図がきっとたくさんあることでしょう。

デザイン・パターンはノウハウのかたまり

中核技術の三つ目は、「デザイン・パターン」です。初耳の方だと、「ソフトウェアの話なのにデザインとはなんぞや?」と思ったかもしれませんね。ここでの「デザイン」とは、ソフトウェアの設計を意味します。デザイン・パターンとは大ざっぱに、ある機能を実現するためのソフト部品(クラス)の組み合わせ方や使い方を規定したもので、ということができます。

例えば、ディレクトリとファイルの情報を画面に表示するエクスプローラのようなアプリケーションを作るときに、ツリー構造を表現するクラス構造があらかじめわかっていたら便利です。しかも、このクラス構造は、他のアプリケーションでもよく使われるものかもしれません。デザイン・パターンは、このような典型的な設計情報を、いくつかのパターン(解決方法)としてまとめたものです。

デザイン・パターンを再利用すれば、毎回すべてゼロから自分で考えてコードを書く必要がなくなります。他のプログラマなどと会話するときも「

パターンを使った」と言えば、どのような実装なのかを説明することができます¹⁷。いわば、オブジェクト指向設計のノウハウ集と言うわけです。従来も「ハッシュ」や「バブルソート」など汎用的なデータ構造やアルゴリズムが、様々なアプリケーションで使われていますが、これのオブジェクト指向版と考えればよいでしょう。

筆者は以前から新人エンジニアに対して、まずは代表的なパターンの名前とクラス図を覚えることを勧めてきました。とりあえず形だけを先に覚えておき、実際にそれを使ったデザインに出会ったときに「あのパターンって、これのことだったのか」と実感してもらおうという意図からです。

デザイン・パターンを解説した書籍で最も有名なのは、「オブジェクト指向における再利用のためのデザインパターン(改訂版)」(ソフトバンクパブリッシング発行)でしょう。この書籍には、

¹⁷ 最近ではデザイン・パターンだけでなく、システム全体の大きなソフトウェア構造を表現するためのアーキテクチャ・パターン、業務分析で有効なアナリシス・パターン、特定のプログラミング言語に依存したノウハウのイディオムなど、ソフトウェア開発のさまざまな分野でパターンが提案され、カタログ集として整備されてきている。

Composite(コンポジット)、Observer(オブザーバ)、Iterator(イテレータ)などと名前がつけられた、全部で23種類のパターンが紹介されています。少々とつきづら内容ではありますが、オブジェクト指向によるソフトウェア設計のバイブルですから、ぜひチャレンジして読んでみてください。

特にJavaのクラスライブラリであるJDK(Java Development Kit)には、デザイン・パターンがあちこちで活用されています。これはおそらくMicrosoftの提唱する.NET¹⁸でも同様です。JDKや.NETを理解して使いこなしたいなら、パターンについての知識は不可欠と言えるでしょう。

最後は分析/設計手法に行き着く

オブジェクト指向に限らず、ソフトウェアは最終的にプログラミング言語で記述します。しかし、そのシステムでやりたいことがわかったからと言ってすぐにプログラムが書けるほど簡単な場合は多くありません。特に数人以上のチームで、ある程度の期間をかけてソフトウェアを開発するケースでは、ユーザー(顧客)がどんなソフトウェアを必要としているのかをきちんと分析し、実際のプログラムが正しく要求を実現できるように設計する、きちんとした「分析/設計」作業が必要です。

こうした分析/設計手法は従来も数多く提案されており、「構造化アプローチ」「データ中心アプローチ」「オブジェクト指向アプローチ」といった流れで発展してきました。このことから分かるように、オブジェクト指向アプローチによる分析/設計は、他の手法とまったく異なっているわけではありません。それぞれの手法の優れたところを生かしています。詳しくはこの連載の後半で解説しましょう。

さて、2回にわたってオブジェクト指向の全体像を眺めてきましたが、いかがだったでしょうか。次回からはいよいよオブジェクト指向の中核技術について、詳細を解説していきます。まずは「オブジェクト指向プログラミング」です。カプセル化、継承、ポリモーフィズムなどがなぜそんなに重要なのか、今回の説明では今ひとつピンと来なかった方も、きっと納得していただけると思います。

¹⁸ .NET(ドットネット)は、Microsoftが提唱する、インターネットをベースにした次世代のソフトウェア運用/開発環境。

オブジェクト指向を代表する二つの実践技術

オブジェクト指向技術を使ったソフトウェア開発には、さまざまな実践技術があります。中でも有名なものは、米 Rational Software が提唱する「RUP (Rational Unified Process : ラショナル統一プロセス)」と、最近米国を中心に急速に注目されている「XP (Extreme Programming : エクストリーム・プログラミング)」です。

RUP は、ソフトウェア開発を複数のフェーズに分割する際の指針に加えて、開発に参加する人々の役割、作業内容、標準的な成果物などを詳細に定義した開発方法論です。「開発プロセスの百科事典」と言ったところでしょうか。筆者も実際のシステム開発で利用した経験があります。ユースケース駆動、アーキテクチャ中心、リスク駆動、四つのフェーズ分割というコンセプトはわかりやすく、実務にとっても役に立ちました。RUP は、「Rational Unified Process」という製品名で同社がHTML ベースのドキュメントを販売していますが、「ラショナル統一プロセス入門」(ピアソン・エデュケーション発行)などの参考書もあります。一方、XP は RUP のようなベンダー系のもではなく、米国のソフトウェア技術者 Kent Beck らが提唱している、新しい方法論です。RUP が管理者側から見た方法論だとしたら、XP はエンジニア(主にプログラマ)の主体性を重視した方法論と言えます。XP には、形式的な手順や文書による成果物はほとんど定義されていません。その代わりに、システム開発を成功させるための12の実践項目が強調されています。例えば、1台のパソコンを二人で共有しながらコーディングを行う「ペアプログラミング」、コーディング後に再設計をして構造を改善していく「リファクタリング」、顧客をフルタイムでチームに参加させる「オンサイト顧客」などです。従来のソフトウェア開発の常識を覆す概念も多く「方法論と言えば形式主義で、成果物を重視するもの」というイメージからするとまさに異色です。一貫してプログラマの自主性やメンバーの円滑なコミュニケーションを重視している点は、人間の知的作業の集約であるソフトウェア開発の本質を突いていると言えるでしょう。