オブジェクト指向による実装実験の評価

妻木俊彦, 古村哲也

1. はじめに

優れたプログラムを作成するには，優れた仕様が必要である．特に，オブジェクト指向は，再利用性や拡張性など，ソフトウェアの保守における効果については論じられているものの，システム開発時における意義については，シームレスと言うこと以外に論じられることが少ない．そうしたこともあって，今回は，オブジェクト指向技術のソフトウェア開発における効果について，実証実験を試みた．

2. 実験の狙い

システム開発では，最終的に作成されたプログラムが要求仕様を十分に満たしているかどうかが評価の対象となる．作成されたプログラムが期待通りのものであるかどうかは，仕様の完全性とともに，システム開発技術者の技術力や，技術者間で交換される情報の種類などに影響を受けることになる．

今回の実験では，特にモデル作成者と実装者の間での情報交換に焦点を当て，次の三つの検証項目を設定した．

1) オブジェクトモデルは，プログラムを作成するのに十分な記述能力があるか．

モデルや仕様は，プログラムを作成するのに十分な情報が正確に記述されていることが重要である．情報が欠如している場合，実装作業に入ってからその情報の不足が露呈するはずである．そのため，本実験では，モデル作成者と実装者の間の情報交換を電子メールに限定し，その内容をトレースすることにした．そして，プログラム完成時に，メールの数量と内容を評価することにした．但し，使用するモデル記述言語とプログラム言語の種類によって授受される情報の量は多少変化するかも知れない．

2) プログラムを作成するためにモデルには最低限どのような情報が記述されていればよいか．

モデル作成者から実装者に渡される情報は多ければ良いというものではない．冗長な設計情報は開発作業の効率化を妨げるだけではなく，実装環境に関する専門家である実装者の独自性をも抹殺しかねない．最低限必要な情報の限界を探るために，今回の実験では，各クラスの個々の操作の定義を行うことを省略した．したがって，属性に関してもその定義は不完全である．クラス図やシーケンス図，状態図，シナリオなど，他のモデルから実装者がどこまで必要な情報を獲得できるかを評価する．この評価は，モデル作成者によるプログラムの評価とモデル作成者と実装者の間のメールの解析によっておこなうことにした．

3) 実装者が保持していなければならない技術は何か．

優れたプログラムを作成するには，プログラム言語をはじめとする実装環境に関する知識とモデル解読のための技術があれば十分なのだろうか．暗黙の内に前

提としている技術があれば，それを明示化することは，不特定多数の実装者へのプログラム発注を考えた場合重要である．今回の実験では，中華人民共和国のシステムハウスに実装をお願いして，文化の共有性を排除することにした．また，問題の解析は，モデル作成者によるプログラムの評価をもとにおこなうことにした．

3. 実 験 方 法

本実験を実施するに当たって，モデル作成者と実装者の間の暗黙の了解を極力排除するために，モデル作成者は当社内からオブジェクト指向技術者を無作為に選定し，実装は中華人民共和国のソフトウェア・ハウスにお願いすることとした．使用するモデル記述言語は統一モデリング言語 UML，実装言語にはオブジェクト指向言語である Java，あるいは C＋＋のいずれかを指定した．

実際の作業に当たって，設計は UML，オブジェクト指向言語のいずれにも十分な知識と経験を持った技術者が担当したが，実装者の側はこれらの言語に関して基礎的な知識しか持っていなかった．

また，本実験では各種情報を採集できるように，次の環境条件を設定した．

1) 複数の例題の設定

問題領域の特性によって実装者が必要とする情報の種類を区別するために，複数の異なった種類の例題を設定することにした．いわゆるビジネス・アプリケーションと呼ばれる例題を 2 題，エンジニアリング・アプリケーションに属する例題を 1 題，計 3 題の例題を用意した．

2) 提供情報の制御

設計者から実装者へ提供する情報の種類を制御するために，最初は提供する情報の種類を制限し，設計者が満足するプログラムができあがるまで，その情報を徐々に増やしていくことにした．具体的には，シーケンス図や状態図から操作が設計できるかどうかを試してみるために，各クラスの詳細な操作の設計については，基本的に実装者に任せることにした．

各設計者が実装者に提供した初期情報と実装プログラミング言語は表 1 の通りである．

3) 交換情報のトレース

設計者と実装者の間の情報交換を制御することである．このために，両者の間での通信は電子メールのみを使用することとし，後で解析が可能なように，通信内容を記録することにした．

表 1　モデルと実装プログラミング言語

提供された情報	実装プログラミング言語
例題 1　クラス図，ユースケース図，シーケンス図，シナリオ，モデルの説明	Java
例題 2　クラス図，ユースケース図，シーケンス図，シナリオ	Java
例題 3　クラス図，状態図，シーケンス図，モデルの説明	C++

4. 実 験 結 果

実験は，表 1 に示した通りの情報の提供によって開始された．それぞれのモデルの構造は，後に述べるような独自の構造をしたものであったが，実装されたプログラムは，三つの例題とも，データ処理部と画面制御部をクライアントとし，データ部をサーバとする典型的な 2 階層のクライアント・サーバ型のものであった．このことは，実装者がオブジェクトモデルを，ソフトウェアの構造まで規定してはいない，いわゆる外部仕様と見なしていたことを意味している．外部仕様はプログラムの実装方法を規定はせず，それ故，作成されたプログラムの構造についても制約はないものである．一方，オブジェクトモデルは，一般に，プログラムの構造まで規定しており，そういう意味で内部仕様の一部までをも規定したものと捉えることができる．これまで，内部仕様の作成は実装者の役割とされてきたが，オブジェクト指向システム開発では，モデル作成者と実装者の役割の見直しが必要であることを認識させられた．しかし，オブジェクト指向システム開発のプロセスと役割分担は方法論ごとに異なっているのが現状であり，実際には，各開発プロジェクトが決定しなければならないだろう．

この実験のプログラムはサイズが小さいため，大規模分散開発時の問題については検証はできていない．また，実際のアプリケーション・システムでは障害などの例外事象に多くの処理が費やされるが，今回の実験ではそうした処理は省略してある．永続オブジェクトのデータベース化についても実施していない．

それぞれの例題における実験結果とその評価は，以下の通りである．

4.1 例 題 1

例題 1 は，典型的なビジネス・アプリケーションの一つである受注システムであるが，発注者と納品先が異なるという条件を付け加えてある．

モデリングのための方法論は BOAD 法を使用した．したがって，モデル作成者から実装者に渡されたモデルは，BOAD 法のインタフェース－コントロール－ドメインという 3 階層のアプリケーション・アーキテクチャの構造に準拠したものであったが，作成されたプログラムは，クライアント・サーバ型の 2 階層モデルであった．しかし，この課題では，実装者と同様，モデル作成者もオブジェクトモデルをシステムの仕様と見なし，プログラムの構造上のチェックは行わず，機能の検証のみを実施し，要求したプログラムとしての妥当性を認めた．

モデル作成者と実装者の間の情報交換は，モデル作成者から，仕様違反の指摘が 1 件と，ユーザインタフェースに関する改良依頼が 3 件であった．

4.2 例 題 2

例題 2 も，いわゆるビジネス・アプリケーションである．この課題のポイントは，プログラムの実行中に顧客が新規顧客から既存顧客へと状態遷移を行う所にあった．

本例題のモデルも，モデル作成者が提示した構造は BOAD 法のインタフェース－コントロール－ドメインという 3 階層モデルであったが，当初作成されたプログラムは例題 1 と同様の 2 階層のクライアント・サーバ型のものであった．モデル作成者は実装者に，モデルの構造とプログラムの構造の一致を要求し，それに伴い，実装言語を Java アプレットから Java アプリケーションへの変更も要求した．そのため，モデル作成者と実装者の間での膨大な情報交換が行われた．

モデル作成者は，クライアント・サーバプログラミングではなくモデルで提示した通りの構造をしたプログラミングを期待している旨伝えたのだが，実装者は，使用している属性をどのようにデータベース化するか，ということで悩み，クラスをまとめられなかったようだ．そこで，モデル作成者の方から，いくつかの追加情報を提示することにした．最初に提示したのは，プログラミングのサンプルであったが，それでも問題は解決しなかった．次に，モデル作成者からアプリケーション・アーキテクチャとしてのBOAD法の参照モデルが提示され，この結果，ほぼ満足できるプログラムの構造が入手できた．その後，実装者の方から，タイマーやレターの取り扱い，及びシステム化範囲の確認についての質問が行われた．

問題となったのは，デザイン・パターンの適用についてである．インスタンスの検索処理にデザインパターンを適用すべきとの指摘がモデル作成者から出されたが，デザインパターンだけがアルゴリズム実装の唯一の方法ではなく，また，特定のアルゴリズムに関しては，実装者が実装環境や実行効率を考慮してデザインすべきものという反論もあり，実装者による独自のアルゴリズムを認めた．

4.3 例　題　3

例題3は，制御という，エンジニアリング・アプリケーションの一種である．したがって，BOAD法は採用できなかった．OMTなどの方法論の採用も検討したが，制御系のアプリケーションに関するアプリケーション・アーキテクチャについて言及している方法論はなく，モデル作成者が独自に，クラス図と状態図を使ってモデルを記述した．

初めに納められたプログラムは，C++によって作成されていたが，その構造は他の例題と同様，クライアント・サーバ型のプログラムであった．モデル作成者から，プログラミングのサンプルを提示することによって，ほぼ満足できるプログラムが入手できた．センサー類のクラスがPublicな属性になっていたり，クラスの状態がクラスの操作でなく，メッセージハンドラで変更されている類の改善点が多かったが，それらも簡単な指摘で修正が可能であった．

5. ま と め

今回の実験を通して得られたオブジェクト指向システム開発に関する知見は，次に示すとおりである．

1) オブジェクトモデルを外部仕様と見なすか，プログラムの構造をも規定したモデルと捉えるかは，それぞれのプロジェクトの判断によるだろう．しかしながら，モデルからプログラムへのトレーサビリティを考えるなら，モデルが示している構造を反映したプログラムの作成が望まれる．そのことが，オブジェクト指向システム開発のシームレス性ということであろう．

2) 今回の実装者が，何故クライアント・サーバ型のプログラミング・スタイルにこだわったかについては，これまでの構造化手法による仕様書ではプログラムの構造に対する明確な注文が行われることは少なく，実装者は開発コストという視点からプログラムの構造を決定していたという習慣があったからと考えられる．また，市販されているオブジェクト指向プログラミングの学習書のほとんどが，

ユーザインタフェースの作成事例に費やされており，その結果，ユーザインタフェースとデータ処理を同一モジュールの中で処理するというプログラミング・スタイルが身についてしまうということも原因の一つと考えられる．本格的なアプリケーション・プログラムの作成を志すには，さまざまなプログラミング・スタイルの学習が必要であるだろう．

3） 例題 2 と例題 3 で，求めるプログラムへの到達に差が出たのは，モデルがプログラムの構造を明示的に表現しているかどうかの違いによると考えられる．エンジニアリング・アプリケーションでは，対象世界の構造が明確で，システム作成者に対象世界に対する正しい理解があれば，プログラムの構造にも曖昧性が無いと言える．それに対し，自然発生的に行われているビジネス活動には明確な構造はなく，モデリング段階で特定のアプリケーション・アーキテクチャが設定され，それに合わせたモデルが作成されるが，モデルそのものは，アーキテクチャを明示的に示してはいない．ビジネス・アプリケーションの開発に当たっては，モデル作成者と実装者の間で，アプリケーション・アーキテクチャに関する情報の共有が必須である．

4） エンジニアリング・アプリケーションのモデル化にあたっては，問題領域を抽象化しただけのモデルでは役に立たない．解析，設計，制御などシステムの目的に合わせた機能構造が必要となる．しかし，そのようなシステムの機能を前提としたアプリケーション・アーキテクチャを提示している方法論は見あたらなかった．現段階では，システム開発をするそれぞれのプロジェクトが独自にアーキテクチャ設計を行わなければならないのが現状である．エンジニアリング・アプリケーションのアーキテクチャは，CAD や通信制御など，問題領域ごとにそれぞれ独自のアーキテクチャが研究されている．普段からアーキテクチャの研究を行っておくことが重要である．

5） 今回の実験では，あえてクラスに操作の定義を記述することを避けたが，そのことが本質的な問題になることはなかった．シナリオとシーケンス図，あるいは状態図によって実装者は十分に操作の設計を行うことが可能であり，むしろ，その方が実装者の自由裁量を認めることとなり，意欲的なアルゴリズムの実現が期待できる．ただし，そのようなプログラムを期待するには，前提として，実装者がアプリケーション・アーキテクチャを理解していることや，さまざまなプログラミング・スタイルとアルゴリズムに関する知識をもっていることが前提となることは言うまでもない．実装者の技術力に疑問がある場合は，十分な教育と共に，最低限必要な属性や操作の定義を行うことが必要となる．