情報システム 44－5
（1993．5．18）

オブジェクト指向再利用部品の有効活用と構築に関する考察

春木良且
富士ゼロックス情報システム株式会社
東京大学大学院博士課程先端学際工学専攻

中谷多哉子
富士ゼロックス情報システム株式会社
筑波大学大学院修士課程経営システム科学専攻

クラスライブラリに対しては、内部状態を利用する開いた再利用と、ブラックボックスとして機能のみを利用する、閉じた再利用を分類することができる。これらは、各々その評価基準や構築手法が異なっており、システム分析･設計において考慮しなければ、システム開発の生産性に寄与しない。

本稿では、実際の開発プロジェクトにおける経験に基づいて、「共有」と「再利用」の区別、システム開発工程とライブラリ構築工程の分離の必要性について、明らかにする。

A Consideration in effectively reuseable Object-Oriented class-library development.

Yoshikatsu Haruki
Fuji Xerox Information Systems CO., LTD.
Research Center for Advanced Science and Technology, University of Tokyo

Takako Nakatani
Fuji Xerox Information Systems CO., LTD.
Graduate school of Systems Management, University of Tsukuba

It is important to classify "Reuse in Open" and "Reuse in close" in Object-Oriented Class-library to increase the productivety. "Reuse in Close" is the method to use the functional result of library component, as black-box. And "Reuse in Open" is to use internal state of glass-box library component.

We report that the difference between "reuse " and "common use", or "system constructing process" and "system refinement process" must be separated in Object-Oriented system development.

1. はじめに

オブジェクト指向技術の利点としては、まず上流工程において、高度なモデル記述能力に基づく、開発者間での仕様情報のギャップの低減を指摘することができる[中谷92].オブジェクト指向に基づくシステムのモデル化に関しては、ソフトウェア分析・設計における新しいパラダイムとして、近年盛に研究されている[Rumbaugh 92J, Shlaer 92J, Coad 91].下流工程においては、開発効率を上げる事による生産性向上と、情報隠蔽機能による信頼性の向上が注目されている[Goldberg 88J].さらに、開発の後工程においても、モジュール性の高さを用いた保守性の高さに着目した研究も行われている[Jacobson 92].

本稿では、こうしたオブジェクト指向技術の長所を生かしたシステム開発プロセスにおいて、特に再利用に基づく生産性に焦点を当てる.オブエクト指向技術に基づいた、ソフトウェア部品の構築とその利用に関して、実際の開発プロジェクトにおける経験に基づいて、その実効性を上げるための方法論にいて、検証を行うものである.

2. ソフトウェアの再利用と生産性

2.1 ソフトウェアの生産性

ソフトウェアの生産性とは、以下に示すように定義される[菅野 87].

生産性 = 品質 * 増幅率 * 開発効率

品質とは、そのソフトウェアの持つ価値であり、増幅率とは、生産量と開発量の割合を表す.また開発効率とは、開発量と作業量の割合を示す.ここでは、生産量と開発量、そして作業量の3つの概念を区別する.作業量とは主に開発者の物理的な生産行為の量であり、開発量とは作業の結果製造されたソフトウェアの量を示す.そして生産量とは、最終製品となったソフトウェアの量である.ゆえに、前述の生産性を示す式は、以下のように表される.

生産性 = (価値 / 生産量) * (生産量 / 開発量) * (開発量 / 作業量)

ソフトウェアの生産性は、これら3つの要素の積で表されるため、各要素を向上させることで、それを高めることが可能となる.ここで言う品質とは、ユーザのニーズに対するソフトウェアの適合性を意味し、信頼性を含んだ概念ではない.これは、ユーザの満足度に左右されるため、ソフトウェアの生産工程のみに留まる問題ではなく、定式化することが困難な要素である.又、開発効率とは、新たに開発するソフトウェアに対する作業の効率を意味し、主に開発作業工程の問題である.このように、ソフトウェアの生産性には、開発効率によって表される開発工程と、増幅率によって表されるソフトウェアそのものの、2つの側面が存在する.

以降に、オブジェクト指向技術に基づき考察する生産性とは、ソフトウェア製品そのものを対象とするものであり、主にここで言う増幅率を示す.増幅率とは、前述したように、作業の結果として製造されたソフトウェアが、最終製品に占める割合を意味する.特に、既存の製品を利用することにより実作業以上の量を製造する事が可能となるため、この増幅率とは、既存ソフトウェアの再利用の割合を示すものと考えることが出来る.

2.2 オブジェクト指向システムにおける再利用の形態

オブジェクト指向システム(Object-Oriented System)とは、以下に示すように定義される.

Object-Oriented System = Object + class + inheritance

オブジェクトをシステム要素として含むものを、オブジェクト主導システム(Object-Based System)と呼び、それにクラス概念を包含することにより、オブジェクトの型定義やデータ抽象を実現することを可能とするものを、クラス主導システム(Class-Based System)と呼ぶ.そしてさらに、クラス階層と継承概念を含むものを、狭義のObject-Oriented Systemと呼ぶ[Hu 90].

ソフトウェアの再利用対象には、データ型、関数モジュール等が含まれるが、オブ

ジェクト指向システムにおいては、オブジェクト自体を再利用対象として考えることができる.これらのオブジェクト部品に対しては、部品の内部状態をユーザが利用するか否かにより、(1)閉じた再利用(Reuse in Open)と、(2)開いた再利用(Reuse in close)の、2種類の利用形態を実現することができる.閉じた再利用に用いられる内部状態を持たない再利用部品はブラックボックス部品と呼ばれ、開いた再利用に使われる、内部状態を持った部品は、グラスボックス部品と呼ばれることがある[菅野 87].

2.1.1. 閉じた再利用とその限界

閉じた再利用とは、既存のプログラムを完結したものとして利用する方法であり、その代表例としては、手続型言語における、関数ライブラリの利用が挙げられる.関数ライブラリとは、汎用性のあるプログラムを、ユーザプログラムから呼び出す事が可能なようにした、再利用部品群であり、コンパイルをした後のオブジェクトコードとして提供される、ブラックボックス部品である.

オブジェクト指向システムにおいては、

- 既存のクラス定義に基づくインスタンスの生成とメッセージ通信
- 既存のインスタンスに対するメッセージ通信

の2つの手段によって、この形態の再利用を実現することが可能となる.これは、オブジェクト指向分析・設計手法においては、オブジェクトの集約構造として仕様化されるが[Rumbaugh 92J]、Object-Basedシステムの範疇で実現される再利用である.

ブラックボックス部品は、内部状態を持たないため、この形態の再利用においては、ユーザにはライブラリプログラムの実行結果のみが提供される.つまり、既存のプログラムの機能を、その結果、または副作用として利用するもので、その用途が限定されたものである.そのため、この閉じた再利用は、再利用の形態としては非常に単純なもので、それほど生産性に寄与するものでは無い.

オブジェクトに対するメッセージ通信は、オブジェクトの持っている機能の再利用に該当する.その意味では、これは手続き型言語における関数ライブラリの利用と異なるものではなく、生産性に大きく寄与するものではない.むしろ、厳密な情報隠蔽機構が働いているため、通常の関数ライブラリと異なって、副作用を積極的に利用することは出来ないため、再利用の効果はそれほど高いものではない.これは、オブジェクト主導システムの範疇にある再利用であるため、情報隠蔽機能を中心とした、プログラムの信頼性を実現するといった性格の強い手法と捉えるべきであろう.

2.1.2 開いた再利用と事例

閉じた再利用のそうした欠点を補うものとして、開いた再利用を挙げる事が出来る.開いた再利用とは、部品の内部状態をユーザが利用する形態をとるものである.この開いた再利用を実現するには、既存のプログラムにソースコードをつけ加えて、新しいプログラムを作り上げる必要がある.それによって、プログラムの機能のみではなく、既存プログラムが持っている変数を利用することが可能となり、さらに、実行結果ではなく、プログラムの問題解決プロセス、アルゴリズムも利用することが可能になる.変数の再利用を、関数を利用する機能の再利用と対比して、状態の再利用と呼ぶ事もある.

オブジェクト指向システムにおいては、狭義のオブジェクト指向システムにおけるクラスの階層構造と継承機構によって、既存の変数の参照を実現することが可能となる.これは、変数をも再利用する状態の再利用でもある.また、既存のメソッドを継承し、オーバライドを実現するため、アルゴリズムの再利用でもある.前述した閉じた再利用と、開いた再利用を、ひとつのシステム構造の上に実現することが可能であるという点が、オブジェクト指向システムの、再利用に関する機能的な特長であると言える.

実際のクラスライブラリにおいても、多

くがこの2種類の再利用形態を予想している.例えば、C+において、実質標準的に扱われているクラスライブラリであるNIHCL(National Institute Of Health Class Library)は、クラス単位でファイル化され、(1)クラス定義ファイル(拡張子.h)、(2)メンバ関数定義ファイル(拡張子.c)、(3)オブジェクトファイル(拡張子.o)の、3種類の構造をとる[Gorlen 90].
クラス定義ファイルは、クラスの階層構造とメンバのインタフェースを定義し、メンバ関数定義ファイルは、メソッド本体を定義する.これらのファイルはソースコードで提供される.既存のクラスを用いて生成したインスタンスや、あるいは既存ライブラリ中に含まれるインスタンスに対して、メッセージを送信するためには、オブジェクトファイルのみが存在すれば可能である.しかし、ソースコードが含まれているため、これにより、開いた再利用をも予想したライブラリ構造をとっていると考えることができる.

2.1.3 開いた再利用の問題点

部品の内部状態を利用する、開いた再利用を実現する場合、いくつかの問題点を指摘することができる.ここでは特に、

(1)生産性と信頼性とのトレードオフ
(2)学習時間の必要性
(3)資産の管理

を指摘する.これらのうち、特に(2)と(3)は、ライブラリの利用が生産性を疎外するマイナス要因としての可能性も持ったものである.

(1)生産性と信頼性とのトレードオフ

オブジェクト指向システムにおいては、閉じた再利用のみならず、開いた再利用をシステム構造において実現することが可能となる.開いた再利用は、閉じた再利用に比べ、多くのプログラム要素を再利用の対象とするため、生産性に対する寄与の度合いは大きい.しかし、付加するプログラムは、既存の変数をも参照することになるため、システム構造は、構造化技法にいうモジュール間病的参照(Pathological Reference)構造となる.つまり、クラス階層を用いた開いた再利用は、構造化技法においては、信頼性を低下させるモジュール構造として排除されるものである.

生産性を開いた再利用によって高めようとすれば、そのプログラムの信頼性は必然的に低下し、厳密な情報隠蔽を実現して信頼性を高めるならば、再利用は閉じた再利用のレベルで留まるため、関数ライブラリ程の生産性向上しか望めない事になる.ここに、オブジェクト指向技術おける、生産性と信頼性のトレードオフ関係を見ることができる.信頼性を保ちつつ、高度な再利用により生産性を向上することが、クラス階層を用いた再利用に関する重要な課題である.

(2)学習時間の必要性

開いた再利用を行うためには、ユーザが、既存の部品プログラムのアルゴリズムや変数、関数等の持つ意味を理解していなければならない.他人の手によるプログラムを再利用するためには、そのソースコードを解析する必要がある.そのため、ある程度学習時間を必要とするし、再利用可能なライブラリを実現するための方法論も必要となってくる.実際の開発プロジェクトにオブジェクト指向技術を導入し、クラスライブラリの再利用によって、生産性を上げようと試みても、それほど早急に効果が上がるものではない.むしろ生産性に関して言えば、自ら開発したか、あるいはシステム開発に使用した経験のあるクラスライブラルでなければ、生産性は向上しない.開発開始時点においては、再利用可能なクラスライブラリは、零と考えたほうが無難であり、開いた再利用を行うための学習時間を含め、通常のシステム開発よりも、多くの工数を見積もるべきであろう.

(3)資産の管理

さらに、C++のように、ファイルによってクラスライブラリを管理する、コンパイル型のプログラミング言語では、NIHCLのように、クラス単位でファイル化するのが一般的である.そして、開発チームの学習や

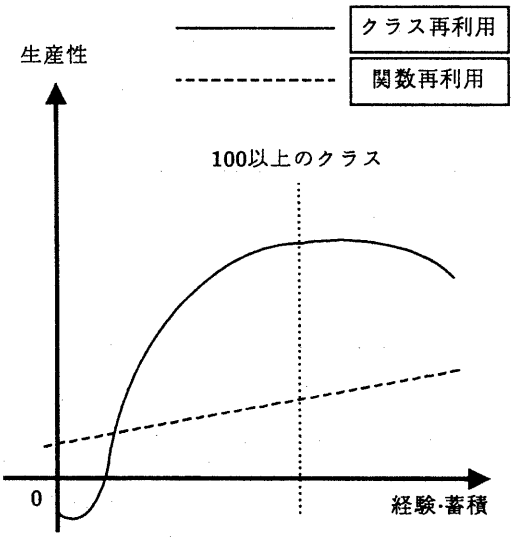
経験によって、クラスライブラリの蓄積は増加する.その場合、各クラスが前述のような3つのファイルに展開されるとするならば、クラスの3倍のオーダーで、ファイルが増加することになる.

プログラミング環境を全てオブジェクト化したSmalltalk-80は、概ね350から400程度のクラスによって構成されている.そのため、C++においても、プログラミング環境をオブジェクト化してライブラリの蓄積を行った場合、その3倍即ち1000から1200程度のファイルを管理しなければならなくなる可能性がある.さらに、それはプログラミング環境のみであり、アプリケーション依存のクラスが含まれてはいない.そして、それらクラスに対する開いた再利用を行うためには、ソースコードの参照によって、既存定義情報を把握しなければならないため、そのファイルを参照する労力も、莫大なものとなる.

一般には、クラスライブラリの蓄積は、生産性に結び付くと捉えられているが、実際には、ファイル化した資産の管理という課題が、蓄積に比例して、重要な問題となって来ることが多い.筆者らの経験では、クラスが100を超えるかあるいは、管理すべきファイルが数百のオーダになると、その再利用のための労力が、急激の増加するようである.オブジェクト指向システムにおいては、ライブラリ資産の蓄積は、生産性の向上をそのまま意味するということではないと言える.

これらが、開いた再利用に纏わる問題点である.特に、(2)(3)を鑑みて、クラスライブラリを用いた開いた再利用による生産性を、関数ライブラリの利用による生産性曲線と比較すると、概ね図2-1に示したようになるようである.(2)の学習時間の問題から、開発初期段階では、関数ライブラリの利用よりも、生産性は低いが、経験と資産の蓄積によって、生産性は急激に向上する.しかし、(3)の資産の管理の問題から、生産性の伸びは低くなって行くといった傾向が見られる.

(図2-1 生産性曲線)



3. 再利用の実態

以下に、筆者の所属する企業において、オブジェクト手法を用いてシステム開発を実践したプロジェクトを例に、その生産性との係わりについて考察し、特に開いた再利用の実態について明らかにすることにする.

本論文で取り上げるプロジェクトの規模は、開発技術者8名により構成され、開発対象は、ワークステーションにより稼働する、ユーザインタフェースを重視した開発支援ツール群である.開発言語及び環境は、Smalltalk-80を使用し、クラスライブラリ管理は、処理系が持つ変更履歴管理機構を利用して行われた.プロジェクトは、2、3人の技術者による3つのサブシステム開発チームから構成され、CASEツール、リポジトリツール、GUIの開発に役割分担をして開発を進めたものである[皆川 92,中谷 92].

表2-1に示すように、プロジェクト内で開発されたクラス数は400あり、そのうち198クラスが、プロジェクト内で共有されているものである.また、各タスクに関するクラス数は、各々CASEツール116、リポジトリツール48、そしてGUIが38クラスである.

(表2-1タスクと開発クラス数)

タスク	クラス数
共有クラス	198
CASEツール	116
リポジトリ	48
GUI	38
合計	400

そのうち共有クラス198を、UI部品である、View、Controller、演算実体のModel、さらにそれらを結び付ける表示Model、その他のクラスに大きく分類した.そして各々のクラスに関して、クラスの数と、Smalltalk-80上の最も抽象度の高いスーパクラスObjectからの、最大の階層距離を求めた.それを表2-2に示す.

(表2-2共有クラスの分析)

クラス種別	クラス数	階層の深さ
View系	32	12階層
Cotroller系	28	8階層
表示Model系	42	4階層
演算Model系	30	3階層
その他	66	8階層

また、それらの共有クラスを用いて開発したもののうち、最もクラス数の多いCASEツールに関して、同様の分類を行ったものを、表2-3に示す.これも共有クラスと同様に、各クラスに関して、クラスの数とクラス階層の最大の深さを、共有クラス及びクラスObjectからの距離によって求めた.

(表2-3 CASEツール構成クラスの分析)

クラス種別	クラス数	階層の深さ (共有クラスから/Objectから)
View系	1	1/7階層
Cotroller系	2	1/5階層
表示Model系	80	3/7階層
演算Model系	20	2/4階層
その他	13	2/5階層

各共有クラスは表2-1に示すように、その他のクラスを除きほぼ30前後である.それらについて、クラス階層の深さを見た場合、UI部品である、View、Controller系は、他のグループよりも非常に深く階層化されている.これらに該当するクラスは、表2-3に示すように、新たに開発されたクラス数が、Viewは1クラス、Controllerは2クラスと非常に少なく、さらにViewは1階層、Controllerは1階層と殆ど階層化されていない.これは概ね、閉じた再利用を中心に行われているという事を示している.

演算実体のModel、及び表示Modelは、旧来のプログラミング言語における、アプリケーション本体的な性格を持ったクラス群である.これらは、表2-2に示すように、表示Modelが42、演算実体Modelが30という共有クラスの数に比べて、表示Modelは4階層、演算実体Modelは3階層と階層が浅い.そのため、表2-3に示すサブシステムおいて、表示Modelが80、演算実体Modelが20と、非常に多くの数のクラスが追加定義され、さらにその階層も若干深いものとなっている.これは、演算実体のModel、及び表示Modelでは、既存のクラス定義に対して、多くのクラスが開いた再利用を行っているという事を示している.

他のサブシステムに関しても、この傾向値はほぼ共通している.このように、実際の開発プロジェクトにおいては、開いた再利用と閉じた再利用を混在して用いており、開発者はクラスの大まかな分類に従って、その再利用形態を使い分けていると考えることができるであろう.

4. 部品構築における問題点とその解決

前項で述べたように、実際のシステム開発においては、共有クラス群とアプリケーションクラス群で、その利用形態が異なっており、それらに対する評価基準も異なったものとなる.つまり、クラスの部品化に関しては、その構成原理自体が単純な抽象化原理として把握することが出来ないと言わざるを得ない.

オブジェクト指向分析・設計等の手法においては、モデル化の対象となるオブジェクトは、統一的に把握され、最終的にシステム全体を抽象化し、システム全体を再利用の対象として把握する[Rumbaugh 92J, Shlaer 92J, Coad 91].しかし、再利用の形態が相違して現れるといった点は、実際のシステム開発においては歴然とした事実であり、全てのシステム要素が、全てのシステム開発において再利用の対象となるべきものではない.以下に、クラスライブラリの再利用を効果的なものとするための、筆者らの経験則を明らかにする.

4.1 再利用と共有の区別

前述したように、共有クラスは一つのプロジェクト全員が共通に用いるものであったが、各サブシステム開発タスク内で構築されたクラス群は、他の開発タスクでは再利用されることがない.さらに、開発プロジェクトを観察すると、各開発タスク内で開発されたクラス自体も、開発タスク内で共有されるものと、各プログラマ個人が再使用するものとに大別されている.また、他の開発プロジェクトやさらに企業外部のコミュニティ等において、類似の再利用部品を見つけることが出来る.これは、再利用部品は、その利用範囲、利用主体によって、その構造や利用形態が左右されるべきであるという事を示している.

そこで、再利用部品の利用範囲別に、利用主体を中心に、再利用のレベルを分類した.(1)個人レベルの再利用、(2)プロジェクト・レベルの再利用、(3)企業レベルの再利して(4)社会レベルの再利用の4レベルである.

(1)個人レベルの再利用

開発者個人では、部品の仕様情報や、利用場面、部品再利用上の問題点等が、特にプログラマのレベルで蓄積されている.一般にこのレベルの再利用は、アルゴリズムを単位としたものが多く、オブジェクトという形態で組織化することは、比較的困難なものが多い.そのため特に、ポリモフィックなメソッド選択や、ハードウェアやシステムに依存した部分に関する、断片的なものが多く存在する.これらは一般にはノウハウと呼ばれるものであるが、実際のシステム開発では、多くの有効性を持っているのは否定できない.

(2)プロジェクトレベルの再利用

一つの開発プロジェクトにおいては、プロジェクトメンバーの価値観や技術背景は共通したものとなる.そのため、クラス構造は比較的共通したものとなることが多く、開いた再利用を行う場合においても、信頼性を低下する可能性は、比較的低いものとなる.そのため、クラス定義の追加が多用されることが多く、クラス部品自体は完成度の低いものとなる.前述の例では、表示Model、演算実体Modelは、共有クラスにおいては階層の浅い、未完成なものであり、各タスクで開いた再利用により完成させるという傾向を持っている.これらは、プロジェクトレベルの再利用部品であると言えよう.

(3)企業レベルの再利用

企業内では、各技術者の価値観や技術背景はかなり相違する.その上で、再利用の目的は企業の競争力をつけるといった、経営戦略的な側面をも持つようになる.そのため、閉じた再利用を中心に行うことにより、信頼性を保ちつつ、生産性の向上を実現せねばならない.こうした再利用部品は、モデルとして一般性を持ったものが多く、さらに完成度の高い部品を提供することにより、増幅率を増大することが可能となる.前述の例では、View、Controller系のUI部品は、共有クラスにおいて深く階層化

され、ほぼ完成されており、各タスクでは閉じた再利用を行っている.これらは、プロジェクトを超えた、企業レベルの再利用部品であると言えよう.

(4)社会レベルの再利用

クラス構造が、一般性を持つようになると、企業という範疇を越えた再利用部品となる.これが、特定の団体により洗練される場合を標準化と呼ぶが、オブジェクト指向言語の場合、Smalltalk-80が社会的に完成度の高いシステムとして認知されているため、これらが実質標準となって行くであろう.実際に、商用、PDSを含め相当数のクラスライブラリが、言語を越え、Smalltalkライクなクラス構造をとっているのは、この社会レベルの再利用であると言える.これらを実現するためには、著作権、共有部品管理、異機種間接続、分散環境等の問題を解決する必要があるが、この問題は本稿の考察範囲から外れるため、ここでは論じない.

以上の、4レベルの再利用は、相反するものではなく、ひとつのシステム内に混在することになる.しかし、そのレベルにより、再利用の目的や部品の完成度等は異なったものとなり、クラス構造を抽象化原理に従って洗練していく、多くのオブジェクト指向分析・設計手法では、各レベルにおいて有効なクラス構造を構築することができない.特に、開いた再利用と閉じた再利用という形での、再利用形態の相違点が表面化する、プロジェクトレベルの再利用と企業レベルの再利用は、明確に区別してとらえる必要があると言えるだろう.

プロジェクトレベルの再利用においては、前述の通り、表示Model、演算実体Model等、アプリケーション依存のオブジェクトにつき、浅い階層によって、ひとつのプロジェクト内で完成させることが多い.これは、通常共有部品と呼ぶものであって、ひとつのアプリケーションで完結しており、プロジェクトを越えて利用されたり、あるいは標準化に向かうことは考えられない.しかし、企業レベルの再利用においては、主にアプリケーションに対して、透過なUI部品等がその対象となっている.これは、単独のプロジェクトだけの問題ではなく、抽象度を高めて、次のプロジェクトに適用されたり、さらに標準化に向かうといった可能性を持つ.これらは、本来的な意味における、再利用部品の範疇に含まれるであろう.

以上から、クラスライブラリによる生産性の向上のためには、このプロジェクトレベルの再利用と企業レベルの再利用、すなわち共有と再利用を明確に区別する必要があるということが明らかになる.これにより、部品の完成度の評価や再利用の形態が相違することになる.

特に、クラスライブラリの完成度は、抽象度と他のオブジェクトからの参照の度合いにより、統一的に定量評価する方法が提起されている[青木 93].しかし、ここで述べたように、クラス階層はその再利用のレベルにより評価を行うべきであり、絶対的に「正しい」、あるいは「唯一の」クラス階層、クラス階層は存在しない.クラスの抽象度の評価と生産性は、全く別の問題であり、これらの評価基準は、全てのクラス階層に対して有効なものではないと言えるだろう.

実際のオブジェクト指向開発においては、特にC++のような言語では、あまり抽象度の高いクラスは、その目的が分かりにくくなってしまい、それほど機能しない事が多い.前述のNIHCLクラスライブラリでは、最も抽象度が高いクラスとして、NIHCLというクラスが存在する.これは、階層の初期化機能を持つといったコメントが付記されているが[Gorlen 90]、ユーザプログラムでは直接サブクラッシングを行うものではなく、その制御の範囲外にある.つまり、プロジェクト内共有のレベルでは、あまり極端な抽象化は必要ないと思われる.

4.2 仕様化と部品化の区別

開いた再利用に纏わる問題点を前記に指摘したが、特に、企業レベルでの再利用を実現するためには、以下の問題点が明確化

する.

(1) 再利用者·開発者の心情的な問題
- 他プロジェクトのために部品を提供する技術者は少ない
- いわゆるNot Invented Hereシンドローム

(2) 該当部品の信頼性

(3) 部品の仕様情報を解析する学習時間

プロジェクトレベルにおいては、部品の再利用による利害が一致しているため、組織内で部品情報を提供する動機づけが行われているために、これらの問題、特に(1)の問題は起こりにくい.そのため、目的やドメィン等を異にする業務間での再利用を進める組織作りが必要となり、さらに再利用を企業内に拡大する場合には、先に述べた問題から、以下に示す点を考慮する必要があると考えられる.

(1) 学習時間短縮のための情報提供環境
- リポジトリ
- 部品情報検索機構等

(2) 再利用に関する意識改革

特に、再利用を支援するための組織については、[Ruben 91]に述べられている、GTEデータサービスが挙げられる.そこでは、部品管理組織(Asset Management Groupe, Librarian)と、部品利用者の組織を完全に分割し、さらに管理組織とは別に利用者教育支援グループ(Help Support Groupe)を存在させて、対応する.

しかし、企業レベルでの再利用を行うことを可能とする完成度の高いクラスや、開いた再利用を予想する抽象度の高いクラスなどによる再利用部品は、分析·設計といったトップダウン工程によって開発されることは殆どなく、実際には開発されたシステムの洗練作業といった、ボトムアッププロセスを経由しなければ、そのシステム構造は明らかになって来ない.さらに、多くの類似タスクを包含するアプリケーションによって、より再利用度が高いクラス構造が明らかになって来る.そこで、3点めとして

(3)部品構築工程とシステム開発工程の分離

を指摘する.旧来のオブジェクト指向分析·設計手法においても、このボトムアッププロセスの必要性は明らかになっていた.そのため、クラス階層の洗練のためには、ウォーターフォール型の開発プロセスにおいて、各工程を行き来する、ラウンドトリップ型、あるいはスパイラル型の開発プロセスが提唱されていた[青木 93,中谷92].

しかし、実際のシステム開発においては、少なくともワークステーションという開発リソースを用いなければ、この各工程を行き来する開発は不可能である.さらに、ワークステーションにおけるシステム開発においては、ひとりの開発者が分析から設計、プログラミングまでをカバーしており、実質的にひとりで開発を行う、分散開発とは言い難い開発形態をとっていることが明らかになった.そのため、システム規模に限界があり、さらに工程別の分散開発やメインフレームを用いた開発には、ラウンドトリップ型の開発プロセスは適用することができない.

これは、システムの仕様化とクラス構造の洗練を統一的に捉えていることに問題がある.システムの仕様化とは、クラス構造からできるだけ冗長性を排除する方向性の作業であるが、クラスライブラリの洗練は、可読性や再利用の可能性を高めるために、冗長性を高める方向の作業である.例えば、ポリモフィックなメソッドの実現は、まさにその例として指摘できる.これらの作業は、方向性が異なっているため、効果的な再利用部品を見つけ出すためには、システム開発作業を、工程として明確に分離する必要があると言える.

5. 結論

オブジェクト指向システムにおいて、クラスライブラリを用いて生産性を上げる事は、それほど短期間に、劇的な効果を上げることはできない.勿論、開いた再利用という強力な手段を実現する機能を持つが、そのためには、多くの学習や、手法の洗練、標準化などを必要とする.

現実に、多くの開発プロジェクトにおいて、再利用の必要性や価値があり、プロジェクトレベルの再利用から、企業レベル、社会レベルへと認知され、標準化されて行くのは、データ構造とユーザインタフェース、特にグラフィカルなもの、そして基本ソフトの一部程度なのではないかと思われる.それ以外は、個々のシステムが持つ目的に依存する要素が非常に強いため、ライブラリという形での洗練を行うことが困難であり、個人レベルからプロジェクトレベルでの、「共有」で留まるものであろう.

むしろ、オブジェクト指向技術による再利用の形態は、オブジェクト群を用いたシステム構造、つまりオブジェクトを用いたシステムのモデル自体を、プログラミング以前の上流工程での再利用に変質するようになるのではないかと思われる.例として上げた、NIHCLクラスライブラリも、Smalltalk-80ライクなクラスライブラリと言われているが、プログラムコード自体というよりも、Smalltalk-80の開発プロセスにおいて洗練されてきた、データ構造などのモデルを、再利用するといった方向性をもっていると考えることができる.結論として、今後ライブラリはプログラムライブラリではなく、モデルライブラリとしての可能性を持つと思われる.そのためには、問題類型という観点が、オブジェクト指向分析・設計作業に必要となるであろう.
今後は、システム対象の類型化を、オブジェクト指向分析・設計作業に包含する方向での研究を進める予定である.

参考文献

- [菅野 87]菅野友文:ソフトウェアの信頼性,日科技連出版,1987
- [Goldberg 88J]Goldberg,A. & Robson,D..相磯秀夫監訳:Smalltalk-80対話型プログラミング環境,オーム社,1988
- [Gorlen 90]Gorlen,E.&Plexico, S.&Orlow,M. :Data abstraction and Object-Oriented Programming in C++,Wiley,1990
- [Hu 90]Hu, D.:Object-Oriented Environment in C++」,MisPress,1990
- [Coad 91]Coad,P.&Yourdon,E.:Object-Oriented Analysis, Prentice Hall, 1991.
- [Ruben 91]Ruben, P. "Implementing Faceted Classfication for Software Reuse", Communication of ACM, Vol34,No5, p88-97.May 1991
- [Rumbaugh 92J]Rumbaugh/Blaha/Premerlani/Eddy/Lorensen.羽生田栄一監訳:オブジェクト指向方法論OMT,トッパン,1992
- [Shlaer 92J]Shalaer,S&Mellor,S.本位田真一外訳:続オブジェクト指向システム分析,啓学出版,1992
- [Jacobson 92]Jacobson, I:Object-Oriented Software Enginering, Addison-Wesley,1992
- [皆川 92]皆川誠他:"Smalltalkによるグループ開発事例",ソフトウェアシンポジューム'92論文集,情報処理学会,1992
- [中谷 92]中谷多哉子:"オブジェクト指向によるシステム分析",第44回全国大会チュートリアルセッション資料,情報処理学会,1992
- [青木 93]青木淳:オブジェクト指向システム分析設計入門'ソフトリサーチセンター,1993
- [春木 93]春木良且:オブジェクト指向システム構築(初稿),啓学出版,1993