ソフトウェア工学 77－3
（1991. 2. 7）

オブジェクト指向言語のクラスの品質評価について

梁 海述† 辻野 嘉宏 都倉 信樹

江原大学校 韓国†
大阪大学 基礎工学部

本稿では従来の品質評価基準をオブジェクト指向言語で書かれたプログラム（オブジェクト指向プログラムという）へ適用する際の問題点を指摘し，よりオブジェクト指向プログラムに適した新しい二つの品質評価基準を提案する．一つはオブジェクト指向プログラムの構成要素であるクラスをモジュールと考え，そのモジュール強度を無強度，中間強度，理想型強度に区分し分類できる評価基準である．またクラス間の継承関係による結合度を無結合度，隠れた結合度，部分結合度，開かれた結合度に区分しより適した分類のできる評価基準を提案する．最後にオブジェクト指向言語C++のクラスの特性を考察し，提案する評価基準の適用結果を分析する．

Quality Evaluation of the Classes in Object Oriented Languages

Haesool YANG† Yoshihiro TSUJINO Nobuki TOKURA

Kangweon National University, Chuncheon, KOREA†
Faculty of Engineering Science, Osaka University, JAPAN

Some criteria have been proposed for quality evaluation of modules but they are not necessarily applicable to classes in object oriented languages. In this paper, three "strength" levels of classes are defined i.e. nil strength level, intermediate strength level and ideal strength level, and four "coupling" levels in terms of the inheritance relation between classes are defined i.e. nil coupling level, concealed coupling level, partial coupling level and open coupling level. These notion are defined by considering the language C++.

１．まえがき

ソフトウェアシステムの規模の大型化，複雑化に対応し，ソフトウェアシステムの開発と保守を効率的にするための技法として，分解技法と抽象化技法があげられる[1]．分解技法は大型ソフトウェアを独立的に扱える複数個のモジュールにわける技法であるが，このようにわけたモジュールもやはり高い複雑さをもつ可能性がある．一方，抽象化技法はソフトウェアシステムの増加する複雑さを扱うのに役に立つ強力な技法である．この技法では，可能なかぎり細かいことを隠し，モジュールがどのように実現されるかではなく，どのような機能を実現しているかに重点をおく技法である．

最近では，よりよい抽象化構造をもつ技法として抽象データ型（ADT）に基づくシステム構造が研究されている．つまり，よりよい抽象化を行うことによって，大きくて複雑な問題を解決することができると考えられている[1,2,3]．このADTのような新しい抽象化技法をADT指向技法と呼ぶ．ここでいうADTとは，値の集合とその集合の要素に適用される演算の集合をカプセル化したもので，ADT指向の技法はどの言語でも応用することができるが，Ada，Modula-2，CLU，C++などのようなオブジェクト指向言語ではADTをよりよく実現することができる．オブジェクト指向言語では，データをカプセル化し，ADTに関連するすべての情報を外部から参照可能にすべきか(以下，Export），外部からは参照不可能（以下，Hidden）にすべきかを言語機能を用いて定義することができる．ADTを使用するプログラムモジュールは，Exportされた情報だけを使わなければならず，隠された情報を参照できない．従来，品質のよいソフトウェアを生産するために，多様なプログラムの複雑度測定方法，モジュール強度，結合度基準などが研究されてきた．たとえば，Embleyは，AdaからADTの特性を抽出し，モジュール強度と結合度の基準を提示した[3]．しかし，このような基準を，データ抽象化と継承機構を言語機能として提供するオブジェクト指向言語で作成されたプログラムモジュールに適用するには多くの問題点がある．従って，オブジェクト指向言語の登場と普及によって，オブジェクト指向言語によるADT指向の高品質ソフトウェアを構成するための品質評価指針が必要になると考えられる．

本稿では，このような問題点を指摘し，よりオブジェクト指向プログラムに適した新しいふたつの品質評価基準を提案する．ひとつは，オブジェクト指向言語の構成要素であるクラスをモジュールと考え，そのモジュール強度を無強度，中間強度，理想型強度に区分し，分類できる評価基準である．また，クラス間の継承関係による結合度を無結合度，隠れた結合度，部分結合度，開かれた結合度に区分し，より適切に分類できる評価基準を提案する．最後に，オブジェクト指向言語の例としてC++を取り上げ，C++のクラスの特性と提案する評価基準がクラスにどのように適用されるかを考察し，その結果を分析する．

２．従来の品質評価基準

（１）モジュール強度

EmbleyがAda言語を対象として提案したADT指向ソフトウェアの品質評価基準を次に示す．ソフトウェア開発において，強く結合されたADTは，一つの抽象化を表現する単一化された構造と考える．強く結合されたADTは一つのドメインをもち，ADTのすべての演算はドメインに適用される．Embleyは，モジュール強度を，Separable強度，Multifaceted強度，Non-Delegation強度，Concealed強度，Model強度の５つの段階に区分し，定義した[3]．

文献（３）によると，Separable強度をもつモジュールは，たとえば，定義されたドメインを参照しない演算や，演算によって参照されないドメインをモジュール内に定義しているモジュールである，Model強度はただ一つのドメインを定義し，定義されたすべての演算がドメインを参照するように定義されているような場合である．この定義によると，Separable強度を持つモジュールは強度が弱く，よくないモジュールであり，逆に，Model強度のモジュールを最もよいものであると評価できる．

（２）結合度

モジュール間の結合性の観点からみれば，結合関係が少なく，また，単純な結合関係のモジュールからなるソフトウェアシステムがよい．Embleyは，結合度を，Nil結合度，Export結合度，Overt結合度，Covert結合度，Surreptitious結合度の５つの段階に区分し，定義した．

文献（３）によると，Nil結合度を持つモジュールは，ほかのモジュールで定義されたドメインとか演算をまったく使わない，もっともよいモジュールであり，Surreptitous結合度をもつモジュールは，ほかのモジュールで定義された内容をすべて参照し，モジュールを定義するのにこれを使うようになる場合で，最も結合度が高くなり，好ましくないモジュールであるとされている．

３．オブジェクト指向環境下のソフトウェアの品質評価基準

３．１　オブジェクト指向環境下のADT指向ソフトウェアシステム

ADTの考え方は，抽象化概念のない普通のプログラミング言語でも応用が可能であるが，抽象化概念を言語機能として支援するオブジェクト指向言語を用いることにより，よりよくADTを実現することができる．オブジェクト指向言語では，通常のプログラミングでの型と同様に，定義されたクラスを用いて変数を定義し使用することが可能である．これは，１９６０年代後半からのプログラミング言語に組み込みの型の概念から，基本言語にプログラマが新しいデータ型を追加することを許そうという努力の結果であると見ることができる．

オブジェクト指向の意味はいろいろと解釈されるが[4]，オブジェクトはADTであると単純にみなすならば，データと演算をそのオブジェクトにカプセル化し，モジュール化と情報隠蔽を行っていると考えることができる．また，オブジェクトとクラスをデータと型の概念としても見ることができるが，その場合には，各データ（オブジェクト）は型（クラス）の要素と考えることができ，型は，継承関係によって，下位型と上位型の関係として関連づけられる．このような観点から考えると，オブジェクト指向言語はADT指向のシステム実現に適切であるとみることができる．次に，オブジェクト指向環境下のADT指向ソフトウェアシステムで使われる用語を定義する．

［定義１］ADT(Abstract Data Types)は，対(D,P)で，Dはドメインの集合で，Pは演算の集合である．ここで，ドメインは演算の対象となる値の集合であり，演算は関数，手続き，定数である．□

このようなADTに関する定義の観点から見ると，言語から提供される“組み込み”型などもADTとして見ることができる．例えば，Pascalの論理型を見ると，ドメインとして {true,false} を，演算の集合として {not,and,or,<,<=,=,<>,>=,>,ord,succ,pred,false,true} をもつ．つまり，言語が提供する組み込み型を組み込みADTとしてみることができる．

3．2　Embleyが提示した基準との比較

Embley[5]は，ADTをExportされたドメインとExportされた演算との集合として定義した．従来の言語で実現されたADTは，外部からのそのドメインへの参照を言語機能として許している[7]．たとえば，CやPascalではもちろんのこと，AdaでもADTを表現するデータ構造に対するすべての情報がインタフェース部分に現れ，特に宣言しなければ，これをそのADTを利用しようとする他のモジュールで使用することが許される．しかし，C++，Flavor，Smalltalk-80などのオブジェクト指向言語で実現したADTのドメインでは，ふつうは，そのADTを利用するモジュールからの直接参照が許されないので，Embleyが提案したExportされたドメインの定義は，オブジェクト指向環境下のADT指向ソフトウェアシステムのクラスでは意味を持たない．また，Adaでは完全な意味の型階層を持たないが，オブジェクト指向言語で実現されたADTは，継承機構を用いて，上位クラスと下位クラスの階層構造を持つように実現するので，Adaでの制約された型の概念とは差があり，これによってEmbleyが提案した基準をオブジェクト指向言語のクラスに適用するには多くの問題点がある．したがって，オブジェクト指向言語の代表的な特性である抽象データ型と継承機構を言語機能としてもつオブジェクト指向言語で作成されたプログラムの一般的な品質評価基準が必要となる．以下，C言語の拡張としてよく用いられているC++のクラスを想定して議論を進める．

3．3　オブジェクト指向システムでのモジュール強度測定基準

オブジェクト指向システムでモジュール強度を測定することは一つのクラスがどの程度のデータ抽象化を行ったかに対する評価であると見ることができる．データ抽象化は，データのカプセル化を行い，定義されたインタフェースを介してのみオブジェクトを操作させることによって，達成される．高い水準のデータ抽象化を達成しているクラスは単一オブジェクトをもち，他のクラスに継承されるという構造を構成する．したがって，高い水準のデータ抽象化を達成しているクラスは高いモジュールの拡張性を持ち，それはADTの重要な長所の一つである．次に，オブジェクト指向環境下のADT指向ソフトウェアの構成に関する用語を，C++のクラスを想定して定義する．

［定義２］ドメイン(D)：Dはクラスで宣言された変数である．Structで定義されたドメインを複合ドメインという．複合ドメインの構成要素を成分という．□

［定義３］演算(P)：Pはクラスの Public 部分で宣言されたすべての関数と手続きの集合である．□

ADTクラスの実現のため，クラスの定義部には関数と手続きの詳細を記述するが，宣言部には，関数名，手続き名，関数が返す値の型，引数の名前と型だけを記述する．Adaでは，演算名（手続き名，関数名）と定義された型との関係を，転送される引数を(in, in out, out, return)に分けて記述するので，その演算が定義されたドメインの中でどれを入力として受け入れ，どのドメインを返すかを決定することができる．しかし，C++では単に名前と転送される引数の型だけを知ることができるので，定義された演算との関係を定義だけをみて決定することはできない．

Zimmer[8]が定義したオブジェクトモジュールは状態空間（State Space），演算，Probeで構成され，各演算の遂行によって状態空間にある要素が生成変更され，Probeの遂行は状態に対する情報を示す．すべてのオブジェクトモジュールは初期化演算を持っているのでこの演算が遂行されるまで状態が定義されない．演算は単に状態変更と削除と関連があり，新しい状態の定義と関連がないと考えるので，C++のクラスにおいて，初期化演算はADT(D,P)で定義されたPの集合に含まれていないとする．

［定義４］参照：Dを名前がNであるドメインとし，Pを演算とする．Pの実現がNを含んでいるとき，演算Pが，名前がNであるドメインDを参照するという．□

Pの実現で定義される値のドメインは，Pで定義されるといい，返されるドメインという．また，使用される値のドメインを，入力として受け入れるドメインという．また，あるクラスで定義されたドメインや演算を下位クラスの演算の実現に含むとき，下位クラスの演算が上位クラスのドメインや演算を参照するという．

次に，オブジェクト指向環境下のADT指向ソフトウェアシステムでのモジュール強度を測定する基準をC++を想定して考察する．

3．4　モジュール強度測定基準の提案

クラスのモジュール強度を，無強度，中間強度，理想型強度に区分し，以下のように定義する．

［定義５］無強度：あるクラスAが無強度であるとは，クラスAのドメインの集合がふたつの部分に分けられて，クラスAのどの演算も高々一方のドメインしか必要としない場合をいう．すなわち，互いに関連性のないドメインと演算がひとつのクラスに含まれる場合の強度である．特殊な場合としては，次のような場合がある．

・Aのどのドメインをも使用しない演算がA内に存在する場合．

・Aにふたつ以上のドメインが定義されていて，その中でAで定義されたどの演算によっても使用されないドメインが存在する場合．□

無強度であるクラスは，個々のドメインとそれに関連した演算とでカプセル化することで，より高い強度をもつADTに分けられる．通常，無強度であるクラスは，図１で示すような参照グラフを持ち，ふたつに分離可能である．

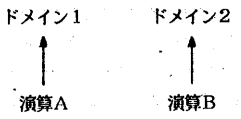

図１　無強度であるクラスの参照グラフの例（矢印は参照を表す）

［定義６］中間強度：クラスが無強度ではなく，２つ以上のドメインを持つか，複合ドメインを持つ場合，そのクラスは，中間強度であるという．□

中間強度であるADTは，分離することが望ましい場合がある．それぞれにドメインを分けて，分離された新しいADTをつくり，一方のドメインだけを参照する演算とその演算によって参照されるドメインを共にカプセル化すればよい．ただし，この過程において演算の変更が必要である．中間強度であるクラスの参照グラフの例を図２に示す．２つ以上のドメインが定義され，定義された演算が１つ以上のドメインを参照しているので，中間強度をもつ例である．

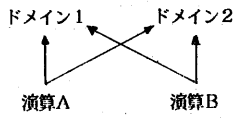

図２　中間強度であるクラスの参照グラフの例（矢印は参照を表す）

［定義７］理想型強度：クラスが，複合ドメインではないただ１つのドメインだけを含み，クラスで定義されたすべての演算がこのド

メインを参照するとき，理想型強度をもつという．□

理想型強度をもつクラスは最も高いモジュール強度をもつADTクラスであるという意味で最もよいデータ抽象化をなすクラスである．理想型強度であるクラスの参照グラフの例を図３に示す．

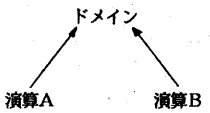

図３　理想型強度であるクラスの参照グラフの例
（矢印は参照を表す）

このようなモジュール強度基準を用いることで，クラスのADTとしての抽象化の程度を評価することができる．

３．５　結合測定基準の提案

ADT品質評価基準のもうひとつの尺度である結合度をみると，従来の機能中心の言語では，手続き呼び出しによる結合度の程度を分類していたが，オブジェクト指向言語環境下のADT中心ソフトウェアシステムは機能中心のソフトウェアとは異なり，部分型（subtype）と型の階層性の特性をもっている．もしAがBの部分型として定義されたとすれば，Aの値はBの値になり，Bのすべての演算はAの演算でもある．つまりBは基底型（Base type）であり，AはBの演算を継承したことになる．Pascalの場合，部分型として部分範囲型（Subrange type）があり，次のように用いられる．

```
type  COLOR  is  (BLUE,GREEN,YELLOW,ORANGE,RED);
type  WARM  is  YELLOW..RED;
type  HOT  is  ORANGE..RED;
```

上で定義した型HOTはWARMの部分型であり，WARMはCOLORの部分型である．Pascalでの部分型は型としては不十分である．その理由は，部分型に対するコンパイル時型チェックがないからである．Adaでは，部分型の概念を制約された型に拡張しているが，制約された型は単に取り得る値を制限するだけである．

型の概念をよく発展させた言語が，オブジェクト指向言語（C++，Smalltalk-80など）である．オブジェクト指向言語C++は，この部分型の概念をオブジェクト指向言語の特徴の１つである継承関係で実現している．C++のクラス中，下位クラスは上位クラスのすべての演算を継承し，固有の演算集合を追加することができる．つまり，オブジェクト指向言語では，継承機構を介してクラス間の関係が成り立ち，継承関係を介してコードを共有することになる[4,6]．継承関係によって，継承を受けたクラスに詳細な実現を見せてしまうならば，カプセル化ができず[4]，また，ソフトウェアモジュールを構成する際に，継承関係を使用することによって下位クラスに情報が漏れ，継承階層の変更が難しくなる．したがって，ADTを使うための理由の１つであるモジュール拡張の容易性を満たすためには，継承関係によってユーザに見せる範囲が狭いほどよい．よって，結合度の測定は，情報の露出程度で測定できる．結合度を提案するための表記を以下に定義する．

［定義８］任意のクラスAに対して，次の記号を定義する．

Dd(A)：Aで定義されているドメインの集合．

Md(A)：Aで定義されている演算の集合．

M(A)：Aで公開されている演算の集合．

Mi(A)：Aが上位のクラスから継承を受けている演算の集合．

Dr(A)：Aの定義部で参照されているドメインの集合．

Mr(f)：fの定義部で参照されている演算の集合．

C(f,A)：次の式を満たす最小解．

$$C(f,A) = \{g \mid g \in Mr(f), g \notin Md(A)\} \cup \bigcup_{g \in Mr(f)} C(g,A)$$

$$C(A) = \bigcup_{f \in Md(A) \cap M(A)} C(f,A)$$

C(A)は，Aで定義され，公開されている演算を実行することによって，実行される可能性のあるA以外で定義されている演算の集合を表している．また，M(A)∩Mi(A)は，Aで公開されているが，A以外で定義されている演算の集合を表している．□

以下に本稿で提案する結合度を定義する．

［定義９］無結合度：任意のクラスの組(A,B)に対して，条件

(Dr(B)-Dd(B))∩Dd(A)=∅，(C(B)∪M(B)∩Mi(B))∩Md(A)=∅

を満たすときAに対するBの結合度は無結合度をもつという．□

つまり，クラスの組(A,B)が無結合度を持つ場合は，BがAから継承を受けていない場合で，クラスAの実現に変更を加えてもクラスBには影響を与えない．

```
// this code define the class called counter
// file counter.h
#include <stdio.h>
class counter
 {
private:
   unsigned int value;
public:
   counter()  {value = 0;} ;
   void increment()  { if (value < 65535) value ++;} ;
   void decrement()  { if (value > 0) value--;} ;
   unsigned int access_value()  {return value;} ;
} ;
```

図４　隠れた結合度をもつADTペアの中の上位クラス

```
// derived class
// inherited from counter class
# include "counter.h"
class range_limited_counter:counter
 {
      private:
            int max_value;
      public;
            range_limited_counter (int max) :()
             {
               max_value = max;
             }
            void increment()
             {
               if (value() < max_value)
                   counter::increment();
             }
            int value()
             {
                counter::access_value();
             }
} ;
```

図５　隠れた結合度をもつADTペアの中の下位クラス

［定義１０］隠れた結合度：任意のクラスの組(A,B)において，(Dr(B)-Dd(B))∩Dd(A)=∅，C(B)∩Md(A)≠∅，M(B)∩Mi(B)∩Md(A)=∅のとき，Aに対するBの結合度は隠れた結合度をもつという．□

継承関係によって，Aで定義された演算がBで現われたもので，BはAで定義された演算でBで定義された演算を通じてのみ参照される．図４と図５の例を定義１０に適用すると，

Dd(A)＝{value}

Md(A)＝{counter,increment,decrement,access_value}

Dr(B)-Dd(B)＝∅，(Dr(B)-Dd(B))∩Dd(A)＝∅

C(B)＝{increment,access_value}，M(B)∩Mi(B)＝∅

C(B)∩Md(A)＝ {increment, access_value}
M(B)∩Mi(B)∩Md(A)＝∅

であるので，上の２つのADTペアは隠れた結合度をもつ例であり，下位クラスが定義した演算を通じて上位クラスの演算を参照する場合で，上位クラスの仕様の変更が発生すると，下位クラスの実現を変更しなくてはならない．

例えば図４と図５を図６のように表現しこれを使うプログラムが図７であるとした場合，図７の①部分でmy_great_counter.value()のメッセージ呼び出しは，図５からわかるように下位クラスrange_limited_counterでの演算value()を呼び出し（図６の②部分），このvalue()は上位クラスcounterのaccess_value()（図６の③部分）を呼び出し，ドメインの参照を実行することになる．

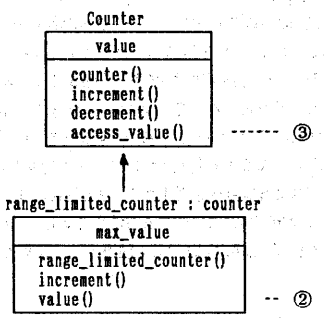

図６　図４と図５の図式化

```
#include "count1.h"
#include <stdio.h>
main()
 {
  range_limited_counter my_great_counter(10);
  for (int i = 0; i < 20; i++)
   {
     my_great_counter.inc();
     printf("¥nmy_great_counter = %d",
            my_great_counter.value());  ----  ①
   }
 }
```

図７　図４と図５を使うプログラム

つまり，下位クラスの演算が上位演算を参照する場合，上位演算の仕様変更はすぐ下位クラスに影響を与えることになる．

［定義１１］部分結合度：任意のクラスの組(A,B)において，(Dr(B)-Dd(B))∩Dd(A)＝∅，Md(A)∩M(B)∩Mi(B)≠∅のとき，Aに対するBの結合度は部分結合度をもつという．□

継承関係によって，Aで定義された演算がBで公開されており，隠れた結合度とは違ってAで定義された演算にBを介して直接参照を許している．図４を上位クラスに，図８を下位クラスにして定義１１を適用して分析すると，

Dd(A)＝ {value}
Md(A)＝ {increment, decrement, access_value}
Dr(B)-Dd(B)＝∅，(Dr(B)-Dd(B))∩Dd(A)＝∅
C(B)＝ {increment, access_value}
M(B)∩Mi(B)＝ {access_value, decrement}
C(B)∩Md(A)＝ {increment}
Md(A)∩M(B)∩Mi(B)＝ {access_value, decrement} ≠∅

となる．この例は部分結合度をもつクラスの例で，下位クラスで新しい演算を定義せずに直接上位クラスの演算を公開している．つまり，これは上位クラスの仕様を下位クラスが利用している．したがって，後で上位クラスの仕様を変更するとき，下位クラスを使うほかのすべての応用プログラムの実現コードの変更が必要になる．

```
//derived class
//inherited from counter class (public form)
// file derived_counter.h
# include "counter.h"
class range_limited_counter : public counter
 {
       private:
          int max_value;
       public:
          range_limited_counter (int max) : ();
          void increment();
 };
// this is implementation file
# include "derived_counter.h"
range_limited_counter::range_limited_counter
                       (int max) : ();
 {
       max_value = max;
 }
void range_limited_counter::increment();
 {
       if ( access_value() < max_value)
             counter::increment();
 };
```

図８　部分結合度をもつADTペアの中の下位クラス

例えば，図４と図８を図９のように表現し，これを使うプログラムが図１０であるとするとき，図１０の①部分でmy_great_counter.access_value()のメッセージ呼び出しは，図９の下位クラスで定義された演算ではなく上位クラスcounterの演算access_value()を参照することになる．したがって，上位クラスのaccess_value()の仕様の変更は図１０でのオブジェクトの実現に影響を与える．

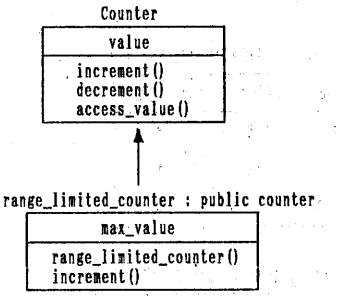

図９　図４と図８の図式化

```
#include "count1.h"
#include <stdio.h>

main()
 {
     range_limited_counter my_great_counter(10);
     for (int i = 0; i < 20; i++)
      {
         my_great_counter.inc();
         printf("¥nmy_great_counter = %d",
         my_great_counter.access_value());  ----- ①
      }
 }
```

図１０　図４と図８を使うプログラム

［定義１２］開いている結合度：任意のクラスの組(A,B)において，(Dr(B)-Dd(B))∩Dd(A)≠∅のとき，Aに対するBの結合度は開いている結合度をもつという．□

開いている結合度は，継承関係によって，Aで定義されたドメインをBで直接使う場合でBの実現がAの実現に依存していることを示す．つまり，Aではカプセル化が行われず，Aの実現を変更する場合，このAを継承するほかのクラスなどの実現の変更をもたらす．このような場合は，C++プログラム作成時には，friendの使用によって生じる．例えば，クラスAのオブジェクトが，クラスBのオブジェクトに特別なメッセージを頻繁に送るときには，クラスBのmethodがクラスAのPrivateドメインを直接アクセスできるようfriendを宣言できる．

次の例は，friendの使用例であり，クラスBすべてと，クラスCのstrange()がクラスAのPrivateドメインを直接アクセスすることを許す例である．

```
class A
 {
    friend class B;
    friend char *c::strange();
 };
```

このように，無結合度が隠れた結合度や部分結合度より結合性が低く，各クラス間の独立性が維持されており，開いている結合ドメインの場合各クラス間の情報隠蔽が破壊されていることがわかる．

4．クラスの品質評価基準の適用

ここで提案した品質評価基準をオブジェクト指向言語で作成したすべてのクラスに適用したとき，それぞれのクラスが唯一の品質評価値をもつことを示し，オブジェクト指向環境下の品質評価基準として妥当であるかどうかを調べるために，すでに作成されたC++のクラスに対してこの品質評価基準を適用した．評価のためのC++のクラスは，任意に作成されたクラス３０個（A）と文献（９）にあるクラス３０個（B）を選択した．その結果を表１に示す．表２でAvg.部分はAとBを平均した値である．

表1　提案モジュール強度特性に対する比率

区分	無強度	中間強度	理想型強度
A	10%	75%	15%
B	5%	70%	25%
AVG.	7.5%	72.5%	20%

表2　提案結合度特性に対する比率

区分	無結合度	隠れた結合度	部分結合度	開かれた結合度
A	3%	68%	15%	14%
B	4%	80%	8%	8%
AVG.	3.5%	79%	11.5%	11%

上の結果から，理想型強度であるクラスと，無結合度であるクラスが平均的にそれぞれ20%，3.5%であるが，これは作成した開発者が抽象化を十分には理解していないことによって，ADT中心での抽象化を構成していないことがわかる．また，キュー，複素数のように簡単でかつ，標準化されたデータ構造を抽象化する次の段階として，品質評価基準をクラス作成の基準にして複雑なADTを作成することによって，よりよい品質のADTソフトウェアを開発できよう．

5．まとめ

ADTを中心としてソフトウェアシステムを構成するのがソフトウェア生産性と維持保守のための有用な方法であるという考えの下に，これをオブジェクト指向言語で実現する場合に適用する方法に対して考察した．本稿ではオブジェクト指向言語環境下のADT指向ソフトウェアシステムでの各モジュールに対するモジュール強度とEmbleyの結合度で考慮していない継承機構に基づいたモジュール間の結合度測定基準を提案した．

提案した品質評価基準をC++クラスに適用した結果，多くのソフトウェア開発者がADTと継承機構を正確には理解せず，開発していることがわかった．このような品質評価基準はソフトウェア開発者に品質のよいソフトウェアを開発するための指針を提供することができる．しかし，提案した品質評価基準はオブジェクト指向言語C++のクラスを基準として研究したので，ほかのオブジェクト指向言語に適用しようとするとき，それらの言語が標準的な継承機構以外の特殊な機能を持つのであれば，それらの性質を考慮することが必要である．したがって，これからの研究方向としては，多重継承機構を支援するオブジェクト指向言語環境下で開発されたクラスの特性を分析し，より拡張された結合度評価基準を考案することがある．また，低い水準のADTをより高い水準のADTに変換する方法を考案し，その変換を自動化できるとすれば，品質のよいADTを誘導でき，信頼性も高まると考えられる．

参考文献

(1) J.Guttag："Notes on type abstraction(Version 2)", IEEE Trans. Software Eng., Vol.SE-6, No.1, pp.13-23 (Jan. 1980).
(2) J.Martin and C.McClure："Software maintenance: The problem and its solution", Prentice-Hall (1983).
(3) D.W. Embley and S.N. Woodfield："Asscessing the quality of abstract data types written in Ada", Proc. of Pheonix conf. on Computers & Comm. (Feb. 1987).
(4) A. Synder："Inheritance and development of encapsulated software components", in Research Directions in Object-Oriented Programming, ed. B. Shriever, D.Wegner, The MIT Press, Cambridge, Massachusetts, pp.165-188 (1989).
(5) D.W. Embley and S.N. Woodfield："Cohesion and Coupling for Abstract Data Types", Proc. of the 1987 Phoenix Conf. on Computers & Comm. (Feb. 1987).
(6) B.Hailpen and V.Nguyen："A Model for object-based inheritance", in Research Directions in Object-Oriented Programming, ed. B.Shriever, D.Wegner, The MIT Press, Cambridge, Massachusetts, pp.147-164 (1989).
(7) H.Robinson and J.Emms："Abstract Data Types: Their Spec., Representation and use", CareIndon press, Oxford (1988).
(8) J.A. Zimmer："Abstraction for Programmers", MaGraw-Hill Book Company (1985).
(9) R.S. Wiener and L.J. Pinson："An Introduction to Object-Oriented Programming and C++", Addison-Wesley Publishing Company (1988).
(10)梁　海述，辻野嘉宏，都倉信樹："データフロー情報を考慮したソフトウェアの複雑度について"，信学論(DI)，J72-D-I，11，pp.789-796 (1989).
(11)梁　海述，辻野嘉宏，都倉信樹："モジュール間依存度を考慮したプログラムの複雑度"，信学論(DI)，J73-D-I，11，pp.882-890 (1990).