ソフトウェア工学 102-16
（1995．1．26）

オブジェクト指向プログラムのテストのためのモデルについて

古川善吾 * 梅田修一 ** 片山徹郎 ** 伊東栄典 ** 牛島和夫 **

* 九州大学 情報処理教育センター
** 九州大学 工学部 情報工学科

オブジェクト指向言語で記述されたプログラム (オブジェクト指向プログラム) の実用化が進んでいる。オブジェクト指向開発方法は、現実世界を反映したオブジェクト指向モデルに基づいてプログラムを設計、プログラミングすることによって、ソフトウェアの生産性や信頼性を向上する技法である。このオブジェクト指向開発方法の実用化が進展するに従い、大規模なソフトウェアが開発されるようになり、その信頼性向上手法としてのテストが必要になってきている。本報告では、オブジェクト指向プログラムのテストを実施する上でテストの品質 (効率、洩れや重複の防止) を左右するプログラムのモデル化について議論する。特にオブジェクト指向プログラムの振舞いに基づく動的モデルについて詳しく議論し、実体階層および導出操作類関数呼出モデルの詳細化および導出操作類関数呼出列への拡張の可能性を検討する。次にオブジェクト指向プログラムの静的なモデルである導出操作類関数モデルに基づく被覆率測定系を紹介する。

Program Models of Object Oriented Programs for Testing.

Zengo FURUKAWA*, Shuichi UMEDA**, Tetsuro KATAYAMA**,
Eisuke ITOH**, and Kazuo USHIJIMA**

* Educational Center for Information Processing, Kyushu University.
** Department of Computer Science and Communication Engineering,
Kyushu University.

6-10-1 Hakozaki, Fukuoka 812, Japan.

This paper discusses about program models for testing of programs written in object-oriented languages (obejct-oriented programs: OO programs). OO methodologies are widely used at present. Because objects are corresponding to concept and materials in the real world, OO methodologies are effectiveness for improvement of productivity and reliability. However, large OO programs require testing methods for more improvement of reliability. There are two types of program models: static models and dynamic models (or behavior models). The model "instance hierarchy and messages passing" is analyzed as a dynamic model of OO programs. The coverage measuring system is introduced which is based on the "inherited method model."

1. はじめに

オブジェクト指向技法で開発したソフトウェア (「オブジェクト指向プログラム」と呼ぶ) のテスト法に関する関心が高まっている [1], [4], [7]。これは、オブジェクト指向によるシステム解析法や設計法、プログラミング方法が普及するにしたがってソフトウェアの開発規模が拡大し、その信頼性の確保が困難になってきているためと考えることができる。

ソフトウェアテストにおいては、テスト十分性を評価したり、テストケースを作成するために、プログラムのモデルが重要な役割を果たす。例えば、プログラムを文の集まりとしてモデル化することによって、文被覆率 (C_0 被覆率) を用いたテスト十分性の評価が可能になる。

オブジェクト指向プログラムでは、テストのための標準的なモデルが確立されているとは言い難い。プログラムは、記述されたソースコードを計算機で実行することによって機能を果たしている。オブジェクト指向プログラムもソースコードを解析して得られる静的な側面と実行時の動的な側面を備えており、テストのために、それぞれの側面でのモデル化が必要である。特に、手続き型言語で記述されたプログラム (「手続き型プログラム」と呼ぶ) に比較して、静的側面と動的側面との違いが大きいことが、プログラムの理解を困難にし、信頼性の低下を招いている可能性が高い。

先に、C++ で記述されたプログラムを例としていくつかのモデルおよびテスト十分性評価方法を提案した [1]。その提案に基づいて静的モデル上での被覆率測定系を試作した [3]。本報告では、動的モデルについての議論と、静的モデルである「導出操作類関数テスト基準」に基づく被覆率測定系の紹介とを行なう。

2. ソフトウェアテストについて

ソフトウェアテストによってプログラムの正当性を証明することは実用時間で行なえないので、テストの終了を判定するための指標が必要である。テストにおいてプログラムを実施する (テスト実施) 前に抽出した測定対象の数とテスト実施によって実現された測定対象の数との比である被覆率をテスト終了の判定の指標として用いるのがテスト十分性評価技法である。

被覆率に基づくテスト十分性評価技法を実現するためには、以下のことを行なう必要がある [1]。

(1) 解釈モデルの構築
(2) 測定対象の定義：測定対象は、以下の条件を満足する必要がある。
   (a) 有限性 ： 抽出された測定対象が無限であると、テストの実施において実現された測定対象を計測しても被覆率が 0 % になるので被覆率を用いてテスト十分性を評価できない。
   (b) 実行前の決定 ： 測定対象は、テストの実施前に確定していなければならない。プログラムの実行状況に応じて測定対象が変動する場合には、被覆率の計算ができない。
   (c) 実現可能性 ： 測定対象は、テスト実施時に実現されたことが計測できなければならない。そのためには、測定対象が実現されなければならない。実現できない測定対象が含まれていると被覆率の値が低くなり、テスト十分性を正当に評価できなくなる。しかしながら、テストは、プログラムが実行できないことを含めた誤りの発見を目的としているので、実行できない測定対象の削除については慎重でなければならない。
(3) テスト基準の定性的信頼性
(4) テスト基準の有効性の実証

このテスト十分性評価技法を適用するソフトウェアテストとして以下のものがある。

(1) 単体テスト ： プログラム単位 (手続き、関数、等々) ごとに行なうテストである。プログラム単位内部のアルゴリズム、変数の宣言等に主に着目したテストである。
(2) 統合テスト：プログラム単位を統合するためのテストである。プログラムのインターフェイスに着目したテストである。
(3) システムテスト：プログラム全体としてのテストである。システムの要求を満足しているか否かを主にテストする。

3. オブジェクト指向プログラムのモデル

オブジェクト指向開発技法におけるモデルとしては、数多くのものが提案されており、使用目的に応じた使い分けが行なわれている。オブジェクト指向プログラムのテストに使用するモデルとしては、オブジェクトの構造を表現するために用いる静的モデルとオブジェクトの振舞いを表現するために用いる動的モデルとがある [5]。オブジェクト指向プログラムのテストのためのモデルとしても静的側面と動的側面をそれぞれモデル化したモデルが考えられる [1], [2], [6], [7]。これらの静的モデルと動的モデルについて本章では議論する。本論文ではオブジェクト指向の設計や解析、プログラミングの用語を断りなしに用いる (文献 [1] 参照)。

3.1 静的モデル

オブジェクトの構造を表現する静的モデルの中で、テストに利用されているモデルとしては、以下のものがある。

(1) 逐次処理モデル [1], [2] : クラスに定義されている操作類関数は、手続き型プログラムであるので、従来用いられている C_0 (文被覆) や C_1 (分岐被覆) テスト基準によってテスト十分性を評価することができる。C++ 言語を対象として C_1 被覆率を測定し、表示するツールが開発されている [2]。オブジェクト指向プログラムは、クラスの操作関数を必要に応じて追加していく漸増方式での開発が一般に行なわれるので、プログラム全体の C_1 被覆率を測定すると操作関数の追加を行なう度に被覆率の計測が必要になる。そのために、このモデルは単体テストにおいて利用することが推奨される。

(2) クラス階層、導出操作類関数モデル [1], [3] : クラスの上位、下位関係をクラス階層として表し、各クラスで宣言されている操作類関数 (生成関数や削除関数、操作関数、親密関数) だけでなく、上位のクラスから継承された操作類関数を測定対象とする。このモデルに基づく C++ プログラムの被覆率測定系を試作した [3]。すべての測定対象は、特定のプログラムの中では呼び出されていない可能性があるので、被覆率を 100 % にするためには、駆動系 (driver) が必要になる。オブジェクト指向言語では、漸増方式のプログラム開発を体系的に行なうために、ライブラリとしてプログラムを登録し再利用を支援するシステムを備えていることが標準的である。ライブラリに登録されたオブジェクト指向プログラムは、提供している全ての機能が利用される可能性があるので、導出操作類関数テスト基準によるテストが必要になる。

3.2 動的モデル

オブジェクト指向プログラムの振舞いを表現する動的モデルとしては、状態遷移あるいはペトリネット、決定表 [5] などが知られている。これらのモデルは、基本的にデータに依存したものであるので、そのままソフトウェアテストのための動的モデルとして用いることはできない。状態遷移に基づくモデルをソフトウェアテストに利用するものとして以下のものがある。

(1) ASF (Atomic System Function) モデル [6] : オブジェクト指向プログラムの統合テストのためのモデルである。ASF の定義は以下の通りである。

(a) オブジェクト : 操作関数の集合

(b) オブジェクト網 (object-network) : 節点が操作関数の集合 (オブジェクトとして表現)、枝がメッセージ (操作類関数の呼出) である有向グラフ

(c) MM 路 (Method/Message Path) : メッセージで結合された操作関数の列

(d) ASF : 入力点事象 (input port event) で始まり、出力点事象 (output port event) で終了する MM 路である。

図 1. に ASF の例を示す。3 つのオブジェクトがあり (Object 1, 2, 3)、それぞれ 2 つあるいは 3 つの操作関数 (meth1, meth2, meth3) を持っている。操作関数の呼出関係がメッセージであり、メッセージの列である MM 路が 3 つある。

この ASF を測定対象としてテスト十分性評価を行なうことが考えられる。

(2) テストグラフモデル [7] : テスト実施の自動化およびプログラムを実行した結果が正しいかを判定する (テスト結果判定) ためのモデルである。テストグラフは、クラスの単体テストに用いるためのモデルであり、クラスが保持するデータに応じた節点と節点間の遷移を表す枝 (一般には操作関数の呼出に対応) から構成されている。Hoffman 等は、このテストグラフの上の路 (開始節点から始まる節点の列) に従って、プログラムの駆動系 (driver) と結果を判定する神託 (oracle) を作成して自動的にテスト実施およびテスト結果判定を行なうためのシステムを構築している。テストグラフの例を図 2. に示す。このテストグラフは、テストのために人手でソースコードとは別に作成されているので、実際のテストではテストグラフそのものの正当性を何らかの手段で保証する必要がある。すなわち、正当性が保証されないと、テスト結果判定において誤りが発見されても本当にプログラム誤りがあることにはならない。

テスト十分性については、テストグラフの節点あるいは枝を測定対象とするテスト基準を考えることができる。

(3) 実体階層と操作類関数呼出モデル [1] : クラスの実体の生成関係に基づいて実体階層を構成する。ただし、実体階層が無限あるいは実行前の決定が不能になることを避けるために、同一のクラスからの実体生成の回数を制限する。制限を 1 回だけに限定すると、階層は、レベルを持った実体ネットワークに

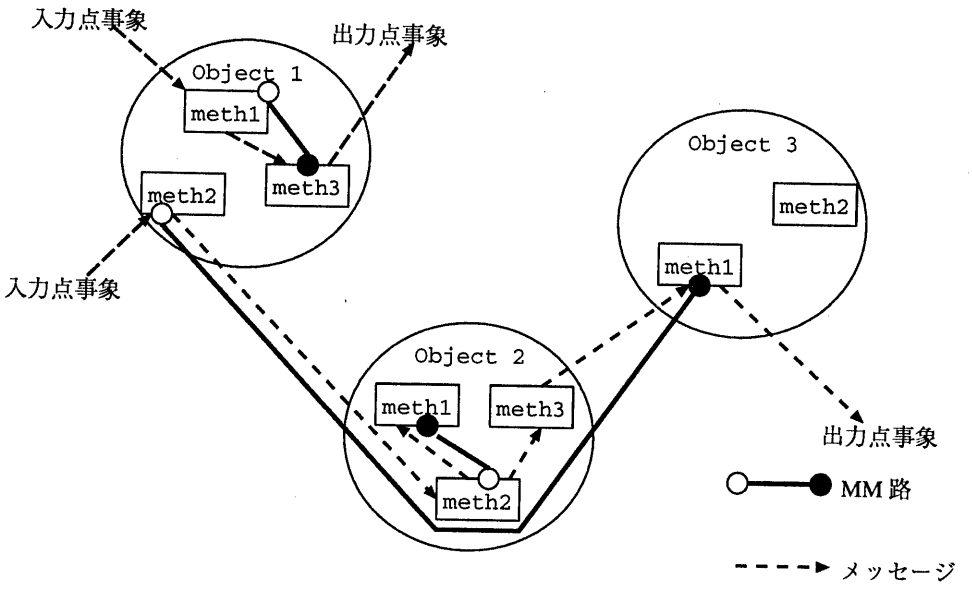

図 1: ASF (Atomic System Function) の例

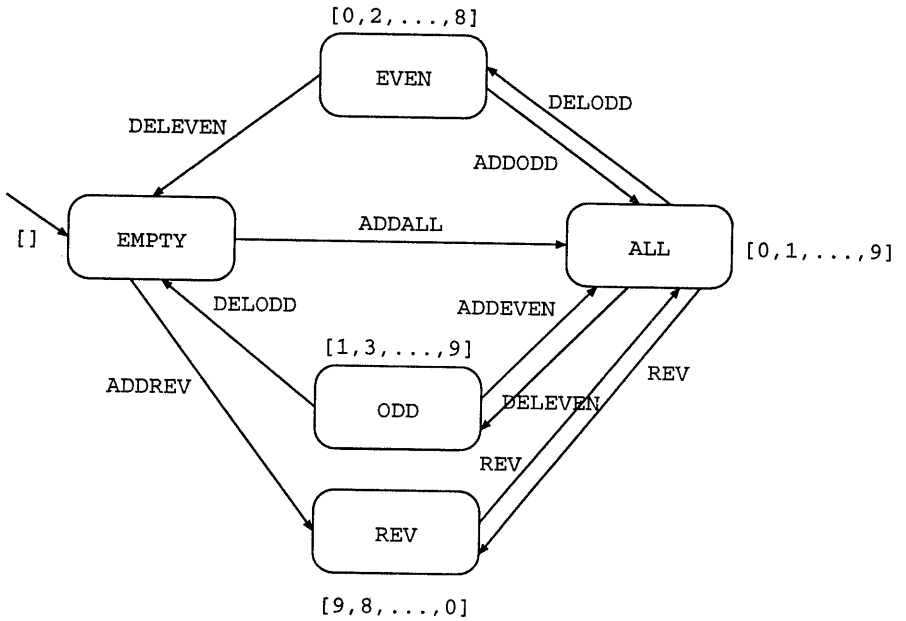

図 2: テストグラフの例

なる。一方、根から葉に至る路の上での出現回数を1回のみに限定すると実体木になる。実体階層の上で操作類関数呼出を測定対象としてテスト十分性を評価することができる。このモデルに基づくテスト

十分性評価は、オブジェクト指向プログラムの統合テストに適用される。

図 3. に C++ で概要を示すオブジェクト指向プログラムの実体階層を図 4. に示す。

```
/*Jorgensen Program outline*/
#include <stdio.h>
class One{
  static One *one;
 public:
  One();
  ~One();
  meth1(){scanf(...);meth3();}
  meth2(){scanf(...);two->meth2();}
  meth3(){printf(...);}
  };
class Two{
  static Two *two;
 public:
  Two();
  ~Two();
  meth1(){...}
  meth2(){meth1();meth3();}
  meth3(){three->meth1();}
  };
class Three{
  static Three *three;
 public:
  Three();
  ~Three();
  meth1(){printf(...);}
  meth2(){...}
  };
main(){
  One *one = new One;
  Two *two = new Two;
  Three *three = new Three;
}
```

図 3: 例題プログラムの概要

図 3. のオブジェクト指向プログラムの概要は、図 1. に示した ASF の例に対応している。ただし、C++ においては生成関数および削除関数を明示し、主プログラムを明示した。ASF をテスト十分性評価の測定対象に用いることについては以下のような点を明確にする必要がある。

(a) オブジェクトの生成・削除など必ずしも明示されない操作関数の呼出を明示的に抽出し、テストの対象とする必要がある。

(b) オブジェクトは操作関数の集合として定義されているが、実体との関係が不明である。もし、クラスの宣言のみに依存しているのであれば、MM 路の中で実行不能なものが存在する可能性がある。一方、実体と 1 対 1 に対応しているのであれば無限にならないことの保証が必要であるし、実行前に測定対象を決定できなければならない。

(c) MM 路の実行可能性が保証されていない。操作関数の呼出の実行が保証されないので MM 路の実行可能性も保証できない。

(d) MM 路は操作関数呼出の順序だけを考慮しているので、オブジェクト指向プログラムが並行に実行されると、MM 路を生成する順序だけでは不十分になる可能性がある。

実体階層と操作類関数呼出モデルの特徴は以下の通りである。

(a) 導出操作類関数呼出の 100 % 実行を要求するテスト基準は、C_0 に包含される。しかしながら、単体テストではなく統合テストで使用されるので、テストの手間の削減については効果がある。

(b) ASF モデルについて先に述べた点については解決が図られている。特に、実体の構造を明示することによって、オブジェクト指向プログラムの振舞いを忠実に反映することができる。

(c) 導出操作類関数呼出を列とすることによって、MM 路と同様のテスト基準を構成することができる。しかしながら、その場合には、導出操作類関数呼出列の実行可能性を明らかにする必要がある。

4. 導出操作類関数の被覆率測定系

オブジェクト指向プログラムのクラス階層モデル上で導出操作類関数の被覆率測定系を試作した [3]。これは、3.1 節で述べたように、ライブラリとして利用される可能性のあるオブジェクト指向プログラムのテスト十分性評価に有効なシステムである。

導出操作類関数の被覆率測定系の特徴は、以下の通りである。

(1) C++ のソースコードを入力し、導出操作類関数の被覆率を計算する。導出操作類関数の実現を記録す

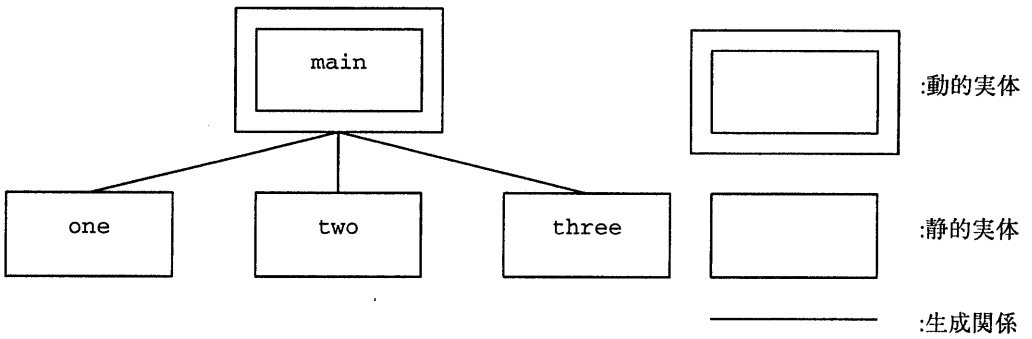

図 4: 実体階層の例

るために、ソースコード変換方式を用いる。ソースコードの変換手順は以下の通りである。

(a) クラスの継承関係を認識する。多重継承には対応していない。

(b) 継承関係に応じて下位のクラスに継承した操作関数を定義し直す。定義し直した操作関数は、継承した操作関数を呼び出す。

(c) 操作関数の実行を記録するために、操作関数名、引数の型、クラス名を保持した探針を挿入する。

(2) 100 % の被覆率を達成するためにほとんどの場合、駆動系を別途記述する必要がある。例えば、「抽象木操作プログラム」 [1] では、表 1. に示す 18 の操作類関数がある。主プログラムそのままで実行すると 18 の中の 16 の操作類関数が実行される (被覆率 89 %)。これを 100 % にするために駆動系が必要である。駆動系および神託の生成ではテストグラフ [7] が参考になる。

表 1: 抽象木操作プログラムの導出操作類関数

クラス	生成関数	削除関数	操作関数
Node	Node()	~Node()	eval()
Binop	Binop()	~Binop()	Node::eval()
Plus	Plus()	Binop::~Binop()	eval()
Times	Times()	Binop::~Binop()	eval()
Uminus	Uminus()	~Uminus()	eval()
Int	Int()	Node::~Node()	eval()

5. おわりに

本報告では、オブジェクト指向プログラムのテスト十分性評価のためにモデル化について議論し、静的モデルの 1 つである「導出操作類関数モデル」に基づく被覆率測定系について紹介した。今後は、動的モデルである「実体階層と導出操作類関数呼出モデル」について形式的な定義と定性的な有効性の検証を行なう。さらに、このモデルに基づく支援システムの開発が必要である。さらに、体系的なオブジェクト指向プログラムのテスト支援システムの構築について今後検討する予定である。

参考文献

[1] 古川善吾, 梅田修一, 片山徹郎, 伊東栄典, 牛島和夫: オブジェクト指向プログラムのテスト法に関する一考察, 情報処理学会情報研報, Vol.94, No.6, pp.123-130, 1994.

[2] 中本幸一, 市瀬規善, 臼井和敏, 小柳敏, 平松健司 et al. : C/C++ プログラミング環境 MKSPACE, NEC 技法, Vol.47, No.6, pp.30-35, 1994.

[3] 梅田修一, 古川善吾, 牛島和夫: オブジェクト指向プログラムの被覆率測定系の試作, 電気関係学会九州支部連合大会論文集, 1994.

[4] Special Section : Object Oriented Software Testing, CACM, Vol.37, No.9, 1994.

[5] MARTIN, J. and ODELL, J.J. : Object-Oriented Methods: A Foundation, Prentice Hall, 1995.

[6] JORGENSEN, P.C. and ERICKSON, C. : Object-Oriented Integration Testing, CACM, Vol.37, No.9, pp.30-38, 1994.

[7] HOFFMAN, D., SMILLIE, J. and STROOPER, P. : Automated Class Testing: Methods and Experience, Proc. of APSEC'94, pp.163-171, 1994.