テストケースの可視化を目的とした テスト用ダイアグラムの提案

浦田　聖也 [a)]・片山　徹郎 [b)]

Proposal of Testing Diagrams for Visualizing Test Cases in Software Testing

Seiya URATA, Tetsuro KATAYAMA

Abstract

A software system becomes a large scale in recent years. As a result, test cases used in software testing have become a large scale. It is difficult to understand where the software system is tested by a large quantity of test cases. For this reason, testing diagrams to visualize test cases are proposed. To generate the testing diagrams, the test case and UML (Unified Modeling Language) diagram are compared and their common information is added to the UML diagram. This paper uses communication diagram and state machine diagram. Generating the testing diagrams can overlook the whole test cases. As a result, we can easily understand where the software system is tested by test cases. Moreover, the testing diagrams support that you find faults in the test cases and/or UML diagrams.

Keywords: Software development, Software testing, Test case, Visualization, UML (Unified Modeling Language).

1. はじめに

近年、ICT(Information and Communication Technology)の進歩とともに、社会における情報システムの担う役割はますます大きくなっている。そのため、システム障害やソフトウェアの不具合がもたらす経済的、社会的影響は、計り知れないものとなっている [1)]。また、システムの需要が増大する中、システムの大規模化、多機能化、複雑化は格段に進んでいる [2)]。

このような背景から、システムの品質確保がより重要視されるようになった。ソフトウェアの品質を確保する上で欠かせない工程に、ソフトウェアテストがある。ソフトウェアテストを実施し、評価するために、ソフトウェアテストの準備の段階で用意するものの１つにテストケースがある。テストケースは、特定の目的またはテスト条件のために記述する [3)]。

一方、システムの大規模化、多機能化、複雑化に伴って、システムを設計することが困難になっている。複雑なシステムを開発する場合、システムをモデリングすることが重要になっている。システムを効果的にモデリングするために、必要なモデルを記述することができる言語の 1 つが UML(Unied Modeling Language： 統一モデリング言語) である [4,5)]。

テストケースの設計や作成は、主に人手によって行われる。そのため、テストケースに抜けや不備が含まれる可能性がある。テストケースに抜けや不備が含まれていた場合、そのテストケースを用いて、ソフトウェアテストを行っても、プログラムに含まれるエラーを発見できない可能性がある。プログラムに含まれるエラーが発見されないまま、ソフトウェアの開発が終了すると、稼働後のシステム障害につながる。すなわち、テストケースに抜けや不備が含まれていた場合、システムの信頼性が下がることになる。

そこで本論文では、ソフトウェアの信頼性を向上させるために、テストケースの可視化を目的としたテスト用ダイアグラムの作成手法を提案する。具体的には、テストケースと UML のダイアグラムの遷移を示す部分をそれぞれ比較し、その結果を UML のダイアグラムに書き込むことによって、UML のダイアグラムにテストケースの情報を追記した図を作成する。この時に作成した図を、テスト用ダイアグラムと呼ぶ。テスト用ダイアグラムを作成することによって、UML のダイアグラムが表す遷移のどこを、どのテストケースがテストしているかを俯瞰できるようになる。その結果、比較対象のテストケースの抜けや不備の発見を支援できる。

本研究では、テストケースとの比較にコミュニケーション図を用いるテスト用コミュニケーション図と、テスト

a)情報システム工学専攻大学院生
b)情報システム工学科准教授

ケースとの比較に状態マシン図を用いるテスト用状態マシン図の２つのテスト用ダイアグラムを提案する。

なお、本研究でのテストケースの抜けとは、必要なテストケースが存在しないことを意味する。テストケースの不備とは、テストケースに誤りが含まれていることを意味する。

2. テスト用ダイアグラム

テスト用ダイアグラムとは、テストケースと UML のダイアグラムを比較し、それらの共通する情報を比較対象のダイアグラムに追記したダイアグラムのことである。本研究で提案するテスト用ダイアグラムは、テスト用コミュニケーション図とテスト用状態マシン図である。本研究で対象となるテストケースは、システムの遷移をテストするためのテストケースである。このテストケースは、ダイアグラムと比較する箇所を明確にするため、本研究で定めるテストケースの記述事項に従って作成する必要がある。

本研究では、テストケースとダイアグラムを比較し、それぞれが一致した場合、ダイアグラムの遷移を表す部分を実線で囲み、実線の内側にテストケースの ID を追記することによって、テスト用ダイアグラムを作成する。このテスト用ダイアグラムを作成することによって、ダイアグラム上の遷移が、実線で囲まれ、実線の内側にテストケースの ID が記述されている場合、ダイアグラム上の遷移が、記述されているテストケース ID が示すテストケースによって、テストされることが分かる。

2.1 テスト用コミュニケーション図

テスト用コミュニケーション図とは、テストケースとコミュニケーション図を比較し、それらが一致した場合、コミュニケーション図のメッセージの矢印を実線で囲み、実線の内側にテストケース ID を追記したダイアグラムのことである。テスト用コミュニケーション図を作成するための、テストケースの記述事項を、表 1 に示す。

テスト用コミュニケーション図作成手法を、以下に示す。

(1) コミュニケーション図と比較していないテストケースを 1 つ選択
(2) 事前条件と送信側の参加要素の名前、または、クラスを比較
(3) 操作とメッセージを比較
(4) 事後条件と受信側の参加要素の名前、または、クラスを比較
(5) 比較した部分がすべて一致した場合、メッセージの矢印記号を実線で囲み、囲んだ実線の内側に、選択したテストケースのテストケース ID を追記
(6) 比較していないテストケースがある場合、(1) に戻る。ない場合テスト用コミュニケーション図が完成

表 1. テストケースとコミュニケーション図の対応表

テストケース ID	テストケース ID 名
事前条件	「送信側の参加要素の名前」がメッセージを送信する
操作	「ガード条件」を満たす「メッセージの名前」を送信する
事後条件	「受信側の参加要素の名前」がメッセージを受信する

2.2 テスト用状態マシン図

テスト用状態マシン図とは、テストケースと状態マシン図を比較し、それらが一致した場合、状態マシン図の状態遷移記述を実線で囲み、実線の内側にテストケース ID を追記したダイアグラムのことである。テスト用状態マシン図を作成するための、テストケースの記述事項を、表 2 に示す。表 2 に示す、テストケースの記述事項の操作(1)と操作(2)、また、期待出力(1)と期待出力(2)は、それらの両方、もしくは、どちらか一方を記述する必要がある。操作の(2) で、「ガード条件」が必要ない場合、「ガード条件」は記述しなくても良い。

テスト用状態マシン図作成手法を、以下に示す。

(1) 状態マシン図と比較していないテストケースを 1 つ選択
(2) 事前条件とソース状態の名前を比較
(3) 事後条件とターゲット状態の名前を比較
(4) 操作の(1) とソース状態の内部の振る舞いの完了を比較
(5) 操作の(2) と状態遷移記述(「ガード条件」と「トリガー」) を比較
(6) 期待出力の(1) と状態遷移記述(「振る舞い」) を比較
(7) 期待出力の(2) とターゲット状態の内部振る舞いを比較
(8) 比較した部分が、すべて一致した場合、状態遷移記述を実線で囲み、囲んだ実線の内側に、選択したテストケースのテストケース ID を追記
(9) 比較していないテストケースがある場合、(1) に戻る。ない場合テスト用状態マシン図が完成

手順(4)～(7) で、操作、または、期待出力と状態遷移図の比較対象が両方存在しない場合は、その手順をスキップする。

3. 適用例

本研究で提案するテスト用ダイアグラムを、プログラム例「整数を入力すると偶数か奇数か判定し出力するプログラム」に適用する。プログラム例を対象

表 2. テストケースと状態マシン図の対応表

テストケース ID	テストケース ID 名
事前条件	「ソース状態の名前」である
操作(1)	「ソース状態の内部の振る舞い」が完了する
操作(2)	「遷移記述のガード条件」を満たす「遷移記述のトリガー」
期待出力(1)	「遷移記述の振る舞い」
期待出力(2)	「ターゲット状態の内部振る舞い」
事後条件	「ターゲット状態の名前」である

にした、複数のテストケースの例と UML ダイアグラムの例を用いてテスト用ダイアグラムを作成する。複数のテストケースの集合をテストスイートと呼ぶ。このプログラム例の要求仕様の例を、表 3 に示す。

表 3. 整数を入力すると偶数か奇数か判定し出力するプログラムの要求仕様の例

要求仕様
ユーザは、1 つの整数をユーザインターフェースに入力する。 入力された数字を偶数か奇数か判定し、結果をユーザインターフェースに出力する。 ユーザインターフェースに出力した結果をユーザが確認することで、入力した数字が偶数か奇数かを把握する。
プログラムの実行例： 数字を入力してください。 数字：2 偶数です。 数字を入力してください。 数字：

3.1 テスト用コミュニケーション図の適用例

本研究で提案するテスト用コミュニケーション図を、プログラム例「整数を入力すると偶数か奇数か判定し出力するプログラム」を対象にした、複数のテストケースの例とコミュニケーション図の例を用いて作成する。

まず、コミュニケーション図の例と複数のテストケースの例に、本研究で提案するテスト用コミュニケーション図を作成する。この複数のテストケースの例とは、コミュニケーション図が表す、すべてのオブジェクト間の通信を、1 度以上テストすることができるテストケースの集合のことである。コミュニケーション図の例を、図 1 に示す。また、複数のテストケースの例をテストスイート A として記述した。このテストケース A を、表 4 に示す。

これらを用いて作成したテスト用コミュニケーション図を、図 2 に示す。このダイアグラムに記述されているテストケース ID から、テストケースがコミュニケーション図上のオブジェクト間の通信の、どこをテストしているのかが分かる。また、図 2 において、メッセージの矢印が、すべて実線で囲まれていることから、図 1 が表すオブジェクト間の通信が、表 4 のテストスイートによって、すべてテストされることが分かる。

次に、図 1 のコミュニケーション図の例と、抜けがある複数のテストケースを用いてテスト用コミュニケーション図を作成する。抜けがある複数のテストケースの例を、テストスイート B として記述した。このテストケース B を、表 5 に示す。表 5 に示すテストスイート B は、表 4 に示すテストスイート A と比較すると「テストケース ID:A2」にあたるテストケースがないため、テストスイート B には、抜けがあることが分かる。

これらを用いて作成したテスト用コミュニケーション図を、図 3 に示す。作成したテスト用コミュニケーション図には、実線で囲まれていないオブジェクト間の通信があるこが分かった。このことから、テストケースの抜けの発見を支援することができた。

続いて、図 1 のコミュニケーション図の例と、不備がある複数のテストケースを用いてテスト用コミュニケーション図を作成する。不備がある複数のテストケースの例を、テストスイート C として記述した。このテストスイート C を、表 6 に示す。表 6 に示すテストスイート C は、「テストケース ID：C3」が示すテストケースの「事前条件」を、『「ユーザ」がメッセージを送信する』とし、「事後条件」を『「偶数奇数判定部」がメッセージを受信する』とする。これにより、それらの条件を同時に満たすオブジェクト間の通信が図 1 のコミュニケーション図に示されていないため、テストスイート C の「テストケース ID:C3」が示すテストケースは、不備があることが分かる。

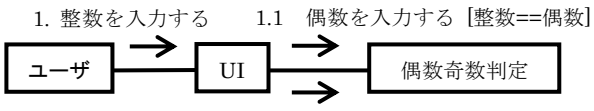

図 1. コミュニケーション図の例

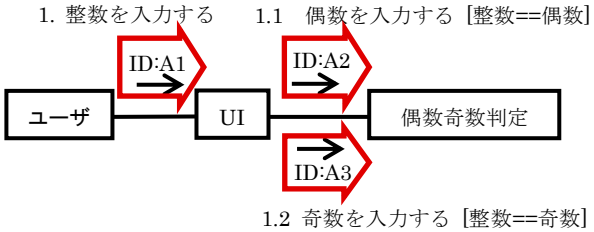

図 2. コミュニケーション図の例とテストスイート A を用いたテスト用コミュニケーション図の例

表4. テストスイートA

テストケース ID	A1
事前条件	「ユーザ」がメッセージを送信する
操作	「整数を入力する」を送信する
事後条件	「UI」がメッセージを受信する
テストケース ID	A2
事前条件	「UI」がメッセージを送信する
操作	「整数==偶数」を満たす「偶数を入力する」を送信する
事後条件	「偶数奇数判定部」がメッセージを受信する
テストケース ID	A3
事前条件	「UI」がメッセージを送信する
操作	「整数==奇数」を満たす「奇数を入力する」を送信する
事後条件	「偶数奇数判定部」がメッセージを受信する

表5. テストスイートB

テストケース ID	B1
事前条件	「ユーザ」がメッセージを送信する
操作	「整数を入力する」を送信する
事後条件	「UI」がメッセージを受信する
テストケース ID	B2
事前条件	「UI」がメッセージを送信する
操作	「整数==偶数」を満たす「偶数を入力する」を送信する
事後条件	「偶数奇数判定部」がメッセージを受信する

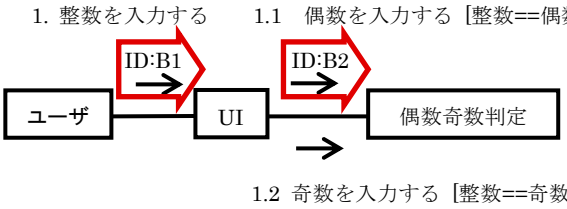

図3. コミュニケーション図の例とテストスイートBを用いたテスト用コミュニケーション図の例

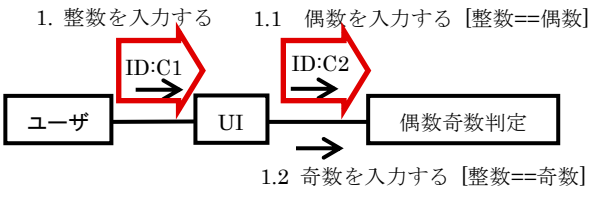

図4. コミュニケーション図の例とテストスイートCを用いたテスト用コミュニケーション図の例

表6. テストスイートC

テストケース ID	C1
事前条件	「ユーザ」がメッセージを送信する
操作	「整数を入力する」を送信する
事後条件	「UI」がメッセージを受信する
テストケース ID	C2
事前条件	「UI」がメッセージを送信する
操作	「整数==偶数」を満たす「偶数を入力する」を送信する
事後条件	「偶数奇数判定部」がメッセージを受信する
テストケース ID	C3
事前条件	「ユーザ」がメッセージを送信する
操作	「整数==奇数」を満たす「奇数を入力する」を送信する
事後条件	「偶数奇数判定部」がメッセージを受信する

3.2 テスト用状態マシン図の適用例

本研究で提案するテスト用状態マシン図を、プログラム例「整数を入力すると偶数か奇数か判定し出力するプログラム」を対象にした、複数のテストケースの例と状態マシン図の例を用いて作成する。

まず、状態マシン図の例と、状態マシン図の例で表す、すべての状態間の遷移を、1 度以上テストすることのできる複数のテストケースの例に、本研究で提案するテスト用状態マシン図を適用する場合を考える。この、状態マシン図の例を、図5に示す。また、複数のテストケースの例をテストスイートDとして記述した。このテストスイートDを、表7に示す。

これらを用いて作成したテスト用状態マシン図を、図6に示す。このダイアグラムに記述されているテストケースID から、テストケースが状態マシン図上の状態間の遷移の、どこをテストしているのかが分かる。また、図6において、状態遷移記述がすべて実線で囲まれていることから、図5が表す状態遷移記述が、表7のテストスイートによって、すべてテストされることが分かる。

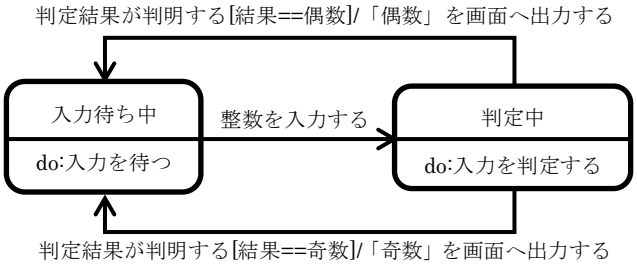

図5. 状態マシン図の例

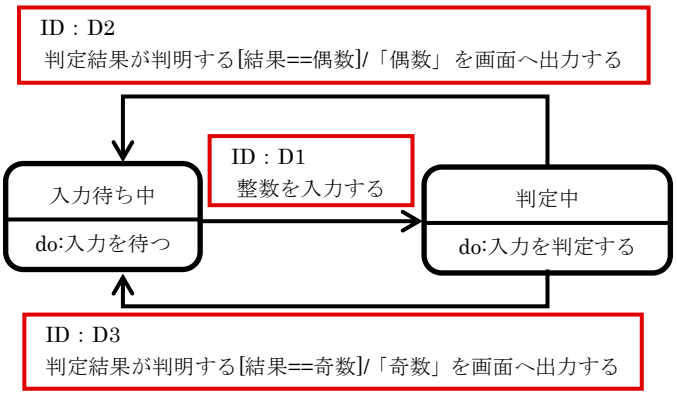

図 6. 状態マシン図の例とテストスイート D を用いたテスト用状態マシン図の例

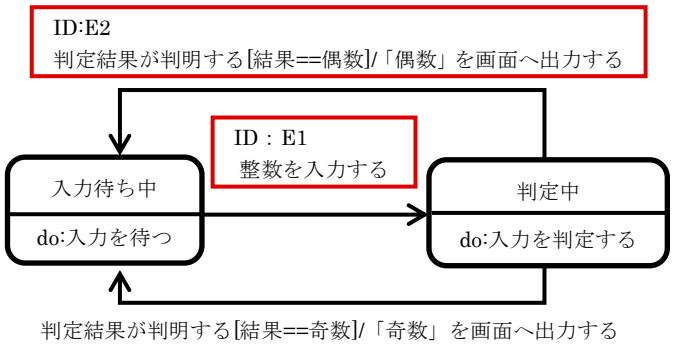

図 7. 状態マシン図の例とテストスイート E を用いたテスト用状態マシン図の例

次に、図 5 の状態マシン図の例と、抜けがある複数のテストケースを用いてテスト用状態マシン図を作成する。抜けがある複数のテストケースの例を、テストスイート E として記述した。このテストスイート E を、表 8 に示す。表 8 に示すテストスイート E は、表 7 に示すテストスイート D の「テストケース ID:D3」にあたるテストケースがないため、テストスイート E は、抜けがあることが分かる。

これらを用いて作成したテスト用状態マシン図を、図 7 に示す。作成したテスト用状態マシン図には、実線で囲まれていない状態間の遷移があることが分かった。このことから、テストケースの抜けの発見を支援することができた。

続いて、図 5 の状態マシン図の例と、不備があるテストケースを 1 つ含むテストスイートの例を用いてテスト用状態マシン図を作成する。不備があるテストケースを含むテストスイートの例を、テストスイート F として記述した。このテストケース F を、表 9 に示す。表 9 に示すテストスイート F は、「テストケース ID：F3」が示すテストケースの「事前条件」を、『「入力待ち中」である』とし、「事後条件」を『「入力待ち中」である』とすることで、それらの条件を同時に満たす状態間の遷移が、図 5 のコミュニケーション図に示されていないため、テストスイート F の「テストケース ID:F3」が示すテストケースは、不備があることが分かる。

表 7. テストスイート D

テストケース ID	D1
事前条件	「入力待ち中」である
操作(1)	「入力を待つ」が完了する
期待出力(1)	「整数を入力する」
期待出力(2)	「入力を判定する」
事後条件	「判定中」である
テストケース ID	D2
事前条件	「判定中」である
操作(1)	「入力を判定する」が完了する
操作(2)	「結果==偶数」を満たす「判定結果が判明する」
期待出力(1)	「“偶数”を画面へ表示する」
期待出力(2)	「入力を待つ」
事後条件	「入力待ち中」である
テストケース ID	D3
事前条件	「判定中」である
操作(1)	「入力を判定する」が完了する
操作(2)	「結果==奇数」を満たす「判定結果が判明する」
期待出力(1)	「“奇数”を画面へ表示する」
期待出力(2)	「入力を待つ」
事後条件	「入力待ち中」である

表 8. テストスイート E

テストケース ID	E1
事前条件	「入力待ち中」である
操作(1)	「入力を待つ」が完了する
期待出力(1)	「整数を入力する」
期待出力(2)	「入力を判定する」
事後条件	「判定中」である
テストケース ID	E2
事前条件	「判定中」である
操作(1)	「入力を判定する」が完了する
操作(2)	「結果==偶数」を満たす「判定結果が判明する」
期待出力(1)	「“偶数”を画面へ表示する」
期待出力(2)	「入力を待つ」
事後条件	「入力待ち中」である

これらを用いて作成したテスト用状態マシン図を、図 8 に示す。作成したダイアグラムには、実線で囲まれていない状態間の遷移があることが分かった。このことから、テストケースの不備の発見を支援することができた。

表 9. テストスイート F

テストケース ID	F1
事前条件	「入力待ち中」である
操作(1)	「入力を待つ」が完了する
期待出力(1)	「整数を入力する」
期待出力(2)	「入力を判定する」
事後条件	「判定中」である
テストケース ID	F2
事前条件	「判定中」である
操作(1)	「入力を判定する」が完了する
操作(2)	「結果==偶数」を満たす「判定結果が判明する」
期待出力(1)	「“偶数”を画面へ表示する」
期待出力(2)	「入力を待つ」
事後条件	「入力待ち中」である
テストケース ID	F3
事前条件	「入力待ち中」である
操作(1)	「入力を判定する」が完了する
操作(2)	「結果==奇数」を満たす「判定結果が判明する」
期待出力(1)	「“奇数”を画面へ表示する」
期待出力(2)	「入力を待つ」
事後条件	「入力待ち中」である

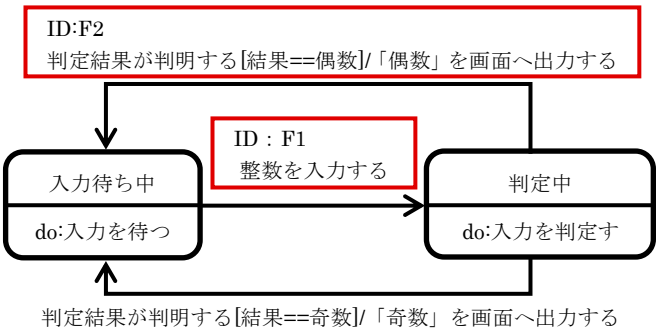

図 8. 状態マシン図の例とテストスイート F を用いたテスト用状態マシン図の例

4. 考察

本論文では、ソフトウェアの信頼性を向上させるために、テストケースの可視化を目的としたテスト用ダイアグラムの作成手法を提案する。具体的には、テストケースと UML のダイアグラムの遷移を示す部分をそれぞれ比較し、その結果を UML のダイアグラムに書き込むことによって、UML のダイアグラムにテストケースの情報を追記した図を作成する。この時に作成した図を、テスト用ダイアグラムと呼ぶ。テスト用ダイアグラムを作成することによって、UML のダイアグラムが表す遷移を、どのテストケースが、どこをテストしているかを俯瞰できるようになる。その結果、比較対象のテストケースの抜けや不備の発見を支援できる。

今回提案したテスト用ダイアグラムのテスト用コミュニケーション図とテスト用状態マシン図を、プログラム例「数字を入力すると偶数か奇数かのどちらかを出力するプログラム」に適用した。作成したテスト用コミュニケーション図から、コミュニケーション図で表すオブジェクト間の通信が、テストケースによって、1 度以上テストされることが確認できた。また、テストケースに抜けや不備がある場合、その抜けや不備を発見できることも確認できた。また、作成したテスト用状態マシン図から、状態マシン図で表す状態間の遷移が、テストケースによって、1 度以上テストされることが確認できた。また、テストケースに抜けや不備がある場合、その抜けや不備を発見できることも確認できた。このことから、今回提案するテスト用ダイアグラムを用いて、テストケースを可視化することによって、ソフトウェアの信頼性向上につながると考えられる。

今回提案したテスト用ダイアグラムで用いるコミュニケーション図と状態マシン図では、システムのワークフローや、時間などをモデリングすることができない。よって、それらをテストするためのテストケースが存在する場合は、コミュニケーション図や状態マシン図と比較しても、テストケースの抜けや不備を発見することができない。そこで、UML で表される他のダイアグラムとテストケースを比較することにより、比較の対象となったダイアグラムの抜けや不備を発見すると同時に、比較対象となったダイアグラムが表すシステムのワークフロー、または、時間などをテストするためのテストケースの抜けや不備を発見できると考える。

UML ダイアグラムやその他のモデルからテストケースを生成する手法は、MBT(Model Based Testing : モデルベースドテスト) と呼ばれ、現在研究が盛んに行われている[6, 7]。しかし、MBT は、モデルからテストケースを作成する手法であるため、モデルに抜けや不備が含まれていた場合、作成されるテストケースにも不備が含まれることになる。このことから、MBT を用いて生成されたテストケースに比べて、今回提案する手法は、テストケースの抜けや不備を事前に発見できるという利点がある。

UML を用いてシステムをモデリングする作業は、主に人手によるものである。そのため、UML を用いて記述されたダイアグラムに、抜けや不備が含まれる可能性がある。抜けや不備を含む UML のダイアグラムを用いて、本研究で提案したテスト用ダイアグラムを作成した場合、テストケースの抜けや不備の発見を妨げる可能性がある。よって、今後の課題として、テストケースの抜けや不備なのか、あるいは、UML のダイアグラムの抜けや不備なのかを、明確にできる必要がある。

5. おわりに

本論文では、ソフトウェアの信頼性を向上させるために、テストケースの可視化を目的としたテスト用ダイアグラムの作成手法を提案した。具体的には、テストケースとUML のダイアグラムの遷移を示す部分をそれぞれ比較し、その結果を UML のダイアグラムに書き込むことによって、UML のダイアグラムにテストケースの情報を追記した図を作成する。この時に作成した図を、テスト用ダイアグラムと呼ぶ。テスト用ダイアグラムは、テストケースとの比較にコミュニケーション図を用いるテスト用コミュニケーション図と、テストケースとの比較に状態マシン図を用いるテスト用状態マシン図の２つを提案する。これらのテスト用ダイアグラムを作成することによって、UML のダイアグラムが表す遷移を、どのテストケースが、どこをテストしているかを俯瞰できるようになる。その結果、比較対象のテストケースの抜けや不備の発見を支援できる。

今回提案したテスト用ダイアグラムを、プログラム例「数字を入力すると偶数か奇数かのどちらかを出力するプログラム」に適用した。作成したテスト用ダイアグラムから、作成に用いた UML のダイアグラムで表す遷移が、テストケースによって、1 度以上テストされることが確認できた。また、テストケースに抜けや不備がある場合、その抜けや不備を発見できることも確認できた。このことから、今回提案するテスト用ダイアグラムを用いて、テストケースを可視化することによって、ソフトウェアの信頼性向上につながると考えられる。

今後の課題として、以下を挙げる。

- 提案手法の充分性の検証
- UML ダイアグラムに抜けや不備が含まれていた際の対応
- 未対応箇所の検討
- テスト用ダイアグラムの可読性の向上
- 他の UML ダイアグラムを本研究に適用
- テスト用ダイアグラム作成の自動化

参考文献

1) 大西健二: ソフトウェアテスト入門　第 1 章 テストはなぜ必要なのか, ソフトウェア・テスト PRESS, Vol.1, pp.2-5, 2005.
2) 玉井 哲雄: ソフトウェア工学の基礎, 岩波書店, 2004.
3) E. van Veenedaal (JSTQB 技術委員会 翻訳):ソフトウェアテスト標準用語集(日本語版), http://jstqb.jp/dl/JSTQB-glossary.V2.0.J02.pdf
4) R. Miles, K. Hamilton (原 隆文 訳): 入門 UML2.0, オライリー・ジャパン, 2007.
5) D. Pilone, D. Pitman (原 隆文 訳): UML2.0 クイックリファレンス, オライリー・ジャパン, 2007.
6) 榊原 彰: モデル駆動テストへの挑戦, ソフトウェア・テスト PRESS, Vol.2, pp.108-115, 2005.
7) P. Samuel, R. Mall, P. Kanth : Automatic test case generationfrom UML communication diagrams, ScienceDirect, Informationand Software Technology, Vol.49, pp.158-171, 2007.