VDM++仕様を対象とした テストケース自動生成ツール BWDM における 適用範囲の拡大のための機能拡張

武藤　崇史 a)・片山　徹郎 b)

Extension to Expand the Scope of Application of BWDM to Generate Test Cases from VDM++ Specifications

Takafumi MUTO, Tetsuro KATAYAMA

Abstract

One of the methods to eliminate the ambiguity of specifications in software design is to use formal methods. One of the formal specification description languages is VDM++. Software testing is necessary in using formal methods, but generating test cases manually is time-consuming and labor-intensive. Therefore, we developed BWDM, which is an automatic test case generation tool for VDM++ specifications, in our laboratory. However, the existing BWDM does not support type definition blocks and conditional expressions for invariant conditions, pre-conditions, and post-conditions. Moreover, it cannot generate test cases for operation definitions that manipulate a state of objects. Therefore, to improve the usefulness of BWDM, this research extends BWDM to solve the above three problems. Consequently, it is confirmed that the extended BWDM can save about 17 minutes in generating test cases compared to test cases generation by hand.

Keywords: software testing, formal methods, test cases, VDM++, automatic generation

1. はじめに

近年のソフトウェアの大規模化及び高機能化に伴い，ソフトウェアのバグが社会にもたらす影響は甚大なものになっている [1]．ソフトウェアにバグが混入する原因の 1 つに，上流工程のソフトウェア設計段階において，自然言語を一般に用いていることが挙げられる．自然言語は元来，曖昧さを含んでいるため，プログラマが仕様書の記述内容を，仕様書の作成者が本来意図していない意味で捉えてしまうことが起こる．それにより，ソフトウェアにバグが混入されてしまう．

この問題を解決するための 1 つの方法として，形式手法 (Formal Methods) [2]を用いた上流工程でのソフトウェア設計が挙げられる．また，形式仕様記述言語の 1 つに VDM++ (Vienna Development Method)がある [3]．

一方で，自然言語，あるいは形式手法を用いた設計のいずれにおいても，ソフトウェアテストが必要であるが，人手によりテストケースを作成することは，手間と時間がかかる．そこで，VDM++仕様を対象としたテストケース自動生成ツール BWDM(Boundary Value/Vienna Development Method)を当研究室で開発した [4,5]．BWDM は，VDM++仕様を対象に，境界値分析 [6]と記号実行 [7]，ドメイン分析 [8]を行い，テストケースを自動生成する．BWDM が生成したテストケースによって，境界値テストとドメイン分析テスト，更に，if-then-else 式の構造認識に基づいたテストを実施できる．

しかし，BWDM は，対象とする VDM++仕様には制限が多く，有用性が高いとは言えない．具体的には，不変条件と事前条件と事後条件における条件式に対応していない問題点や，型定義ブロックに対応していない問題点，オブジェクトの状態を操作する操作定義のテストケースを生成できない問題点の 3 つの問題点が存在する．

そこで本研究では，BWDM の有用性の向上を目的として，BWDM の拡張を行い，上記 3 つの問題点を解決する．具体的には，まず，不変条件と事前条件と事後条件に対応していない問題点を解決するために，条件式の解析及び評価処理を追加する．次に，型定義ブロックに対応していない問題点を解決するために，構文解析処理の修正を行う．加えて，オブジェクトの状態を操作する操作定義のテストケースを生成できない問題点を解決するために，オブジェクトの状態に対するテストケース生成手法を提案し，オブジェクトの状態に対するテストケース生成機能を追加す

a)工学専攻機械・情報系コース大学院生
b)情報システム工学科教授

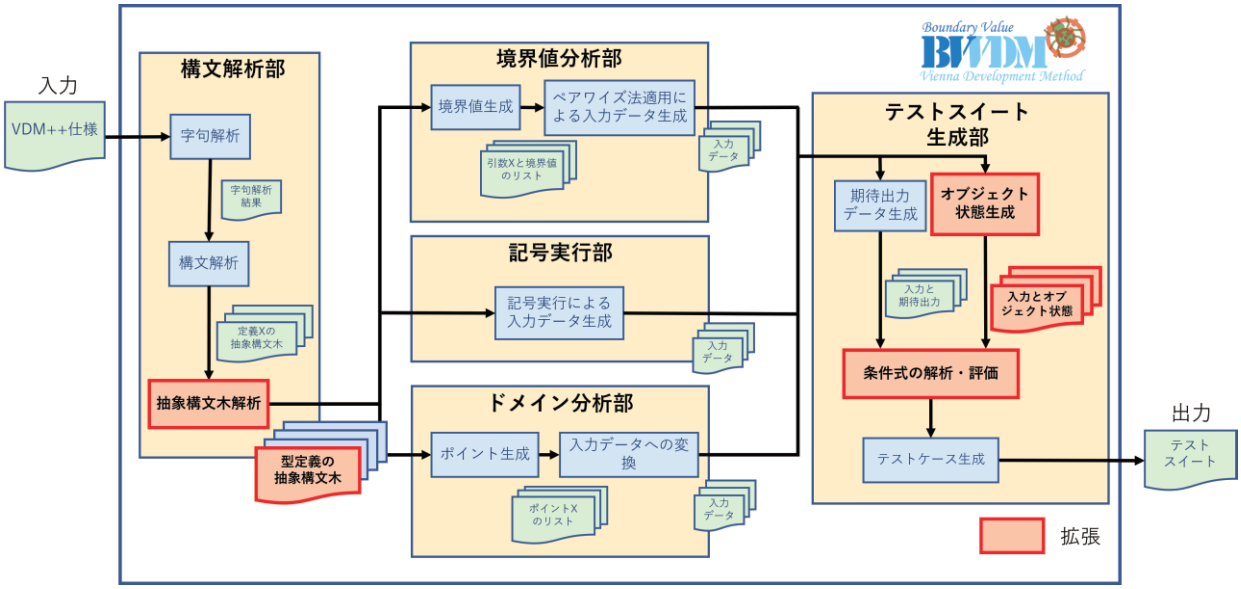

図 1　拡張後の BWDM の構造

る.

2.　BWDM の拡張

BWDM の拡張について説明する．拡張後の BWDM の構造を，図 1 に示す.

2.1　不変条件と事前条件と事後条件における条件式の解析及び評価

既存の BWDM の，定義内で設定される，不変条件と事前条件と事後条件における条件式に対応していないという問題点を解決するために，拡張後の BWDM では，式を解析及び評価する ConditionAnalyzer クラスを追加し，更に，構文解析部とテストスイート生成部を修正する.

構文解析部において, テストケース生成の対象となる定義を表す Definition クラスがあり，このクラスのフィールドに, 入力データを決定する際に必要となる条件式を格納する inputConditions と, 期待出力を決定する際に必要となる条件式を格納する outputConditions を追加する.

inputConditions には，事前条件と引数型に定義されている不変条件を, outputConditions には, 事後条件とインスタンス変数定義ブロック内の不変条件を格納する．なお，期待出力を決定する際に必要となる条件式には，inputConditions に格納する条件式も含まれる.

inputConditions は境界値分析部に渡し，事前条件と引数型の不変条件に対応した入力データを得る．なお，事後条件とインスタンス変数の型の不変条件においては, 操作及び関数の処理終了時に評価する条件式であるため，入力データを決定する際には使用しない．また，事後条件やインスタンス変数の型の不変条件で, 操作後の値を参照する場合はテストすべき項目は存在するが, 本研究で拡張する BWDM では対象としない．ここで，条件式を inputConditions と outputConditions の 2 つに分けて格納しているのは，2.3 節で後述するオブジェクトの状態に対するテストケース生成において, 操作前のエラーと操作後のエラーを区別するためである.

新たに追加する ConditionAnalyzer クラスでは，文字列として与えられた式の比較演算や算術演算, 論理演算処理を行う. テストスイート生成部の期待出力を生成する処理の際に，inputConditions と outputConditions の条件式を，ConditionAnalyzer クラスを用いて評価する. 条件式が false の場合は，処理開始時あるいは処理終了時に，定義内で設定されている条件式を満たしていないことになるため，"Undefined Action"を期待出力とする. 条件式が true の場合は，既存の BWDM と同様の期待出力とする.

2.2 型定義ブロックへの対応

既存の BWDM の，型定義ブロックに対応していないという問題点を解決するために，構文解析部を修正する.

拡張後の BWDM では，構文解析部において，抽象構文解析処理を行う InformationExtractor クラスに, 型定義を表す連想配列 types を新たに追加し，型定義の抽象構文木を types に格納する．各定義ブロック内で型定義を使用している場合，types を基に型定義を実際の型に変換する．さらに，この変換の際, 変換する型が操作定義及び関数定義の引数型であれば inputConditions に, インスタンス変数の型であれば outputConditions に，不変条件の条件式を追加する.

型定義に記述されている不変条件には, その型の任意の値を表す文字列が含まれる．不変条件の条件式を inputConditions もしくは outputConditions に追加する際，この文字列をその型を使用している仮引数もしくはインスタンス変数に置換する．この置換によって，不変条件に対する境界値分析を行うことができる.

表 1　期待する状態と該当する条件

期待する状態	該当する条件
Normal	すべての条件式が true
Failure	インスタンス変数定義の不変条件が false
	事前条件が false
	操作後のインスタンス変数が型の範囲外の値
Undefined Action	事前条件が false
	引数型の不変条件が false
	実引数が引数型の範囲外の値

2.3 オブジェクトの状態に対するテストケース生成機能の追加

既存の BWDM の，オブジェクトの状態を操作する操作定義のテストケースを生成できないという問題点を解決するためにオブジェクトの状態に対するテストケース生成機能を追加する．VDM++仕様は，クラスや型に対する不変条件や，操作定義及び関数定義に設定する，事前条件と事後条件がある. 本研究で新たに生成するオブジェクトの状態に対するテストケースは, これらの条件を用いることで, 入力に対して操作後のオブジェクトの状態が正常か異常か, もしくは入力のエラーがあるかどうかを期待出力とするテストケースである. 操作後のオブジェクトの状態が正常である場合は”Normal”とし，異常である場合は”Failure”とする．また，入力のエラーがある場合は，既存のテストケース生成と同様に，期待出力を"Undefined Action"とする. 期待する状態と，その状態に該当する条件を，表 1 に示す.

オブジェクトの状態に対するテストケース生成機能を追加するために，構文解析部を修正する．既存の BWDM では，操作定義の文の中に if-then-else 式がない場合，その定義のテストケースを生成せずに, 次の定義のテストケース生成処理に移る. そこで, 拡張後の BWDM では, if-then-else 式はないがオブジェクトの状態を操作する操作定義である場合は, テストケースを生成する処理に進むように構文解析部を修正する．また，オブジェクトの状態を操作する操作定義である場合, 操作後のインスタンス変数が満たすべき条件式を，outputConditions に追加する.

更に，テストケースを生成する際に，インスタンス変数に代入する演算式と, インスタンス変数の値が必要になるため，この 2 つをテストスイート生成部に渡して，期待するオブジェクトの状態を求める.

リスト 1　事前条件を含む VDM++仕様の例

```
class 成績評価
functions
  成績評価 : nat -> seq of char
  成績評価(テスト点) ==
    if(テスト点 >= 60) then
      if(テスト点 >= 70) then
        if(テスト点 >= 80) then
          if(テスト点 >= 90) then
            "秀"
          else
            "優"
        else
          "良"
      else
        "可"
    else
      "不可"
    pre テスト点 <= 100;
end 成績評価
```

リスト 2　拡張後の BWDM にリスト 1 を適用した際の出力の一部

```
各引数の境界値
テスト点 : 4294967295 4294967294 0 -1 100 101 60 59
70 69 80 79 90 89

境界値分析によるテストケース
No.1 : 4294967295 -> Undefined Action
No.2 : 4294967294 -> Undefined Action
No.3 : 0 -> "不可"
No.4 : -1 -> Undefined Action
No.5 : 100 -> "秀"
No.6 : 101 -> Undefined Action
No.7 : 60 -> "可"
No.8 : 59 -> "不可"
No.9 : 70 -> "良"
No.10 : 69 -> "可"
No.11 : 80 -> "優"
No.12 : 79 -> "良"
No.13 : 90 -> "秀"
No.14 : 89 -> "優"
```

3.　適用例

拡張した BWDM が正しく動作することを，適用例を用いて確認する.

3.1 不変条件と事前条件と事後条件の条件式が設定されている仕様

事前条件を含む VDM++仕様の例をリスト 1 に，これを拡張後の BWDM に適用した際の出力の一部をリスト 2 に，それぞれ示す.

リスト 2 から, 事前条件の条件式に対するテストケースを生成できることを確認した．また，不変条件と事後条件を含む VDM++仕様を拡張後の BWDM に入力として適用し, 不変条件か事後条件を満たしてない入力に対するテストケースでは, 期待出力が"Undefined Action"となっている

リスト3　型定義を用いたVDM++仕様の例

```
class 平成生まれと令和生まれ
types
   public 「年」 = nat;
   public 「月」 = nat1
   inv m == m <= 12;
functions
   生まれ判定 : 「年」 * 「月」 -> seq of char
      生まれ判定(年, 月) ==
         if(年 <= 2019) then
            if(月 < 4) then
               "平成の早生まれ"
            else
               if(月 > 4) then
                  "令和の遅生まれ"
               else
                  "平成の遅生まれ"
         else
            if(月 < 4) then
               "令和の早生まれ"
            else
               "令和の遅生まれ"
   pre 年 > 1990;
end 平成生まれと令和生まれ
```

リスト4　拡張後のBWDMにリスト3を適用した際の出力の一部

```
各引数の境界値
年 : 4294967295 4294967294 0 -1 1991 1990 2019 2020
月 : 4294967296 4294967295 1 0 12 13 3 4 5

境界値分析によるテストケース
No.1 : 4294967295 4294967296 -> Undefined Action
No.2 : 4294967294 4294967296 -> Undefined Action
No.3 : 0 4294967296 -> Undefined Action
No.4 : -1 4294967296 -> Undefined Action
No.5 : 1991 4294967296 -> Undefined Action
--- 以下省略 ---
```

ことを確認した.

3.2 型定義を用いた仕様

型定義を用いたVDM++仕様の例をリスト3に，これを拡張後のBWDMに適用した際の出力の一部をリスト4に，それぞれ示す.

リスト4から, 型定義ブロックで定義した型がどのような型であるかを特定できていることを確認した. また，型定義ブロックで記述している不変条件に対応した境界値分析に基づくテストケースを生成できることも確認した.

3.3 オブジェクトの状態を操作する操作定義を含む仕様

オブジェクトの状態を操作するVDM++仕様の例をリスト5に，これを拡張後のBWDMに入力として適用した際の出力をリスト6に，それぞれ示す.

リスト5の「カードと割引券で支払う」操作で，0と10を入力とした場合，操作後のインスタンス変数「割引券」の値は-2となる.「割引券」はnat型であり，outputConditionsに格納している条件式「0 <= 割引券」がfalseとなることから，期待するオブジェクトの状態は

リスト5　オブジェクトの状態を操作するVDM++仕様の例

```
class カード払い
types
   public 「円」 = nat;
values
   カード利用限度額: 「円」 = 100000;
instance variables
   割引券: nat := 8;
   カード利用額: 「円」 := 0;
   inv カード利用額 <= カード利用限度額;
operations
   カードと割引券で支払う:「円」 * nat ==> ()
   カードと割引券で支払う(金額, 枚数) ==
      (カード利用額 := カード利用額 + (金額 - 金額 *
         (枚数 * 0.1));
      割引券 := 割引券 - 枚数)
   pre 枚数 <= 10
   post 割引券~ = 割引券 + 枚数;
end カード払い
```

リスト6　拡張後のBWDMにリスト5を適用した際の出力

```
関数名 : カードと割引券で支払う
引数の型 : 金額:nat 枚数:nat
戻り値の型 : ()
生成テストケース数 : 24 件

各引数の境界値
金額 : 4294967295 4294967294 0 -1
枚数 : 4294967295 4294967294 0 -1 10 11

オブジェクトの状態に対するテストケース
No.1 : 4294967295 4294967295 -> Undefinde Action
No.2 : 4294967294 4294967295 -> Undefinde Action
No.3 : 0 4294967295 -> Undefinde Action
No.4 : -1 4294967295 -> Undefinde Action
No.5 : 4294967295 4294967294 -> Undefinde Action
No.6 : 4294967294 4294967294 -> Undefinde Action
No.7 : 0 4294967294 -> Undefinde Action
No.8 : -1 4294967294 -> Undefinde Action
No.9 : 4294967295 0 -> Undefinde Action
No.10 : 4294967294 0 -> Failure
No.11 : 0 0 -> Normal
No.12 : -1 0 -> Undefinde Action
No.13 : 4294967295 -1 -> Undefinde Action
No.14 : 4294967294 -1 -> Undefinde Action
No.15 : 0 -1 -> Undefinde Action
No.16 : -1 -1 -> Undefinde Action
No.17 : 4294967295 10 -> Undefinde Action
No.18 : 4294967294 10 -> Failure
No.19 : 0 10 -> Failure
No.20 : -1 10 -> Undefinde Action
No.21 : 4294967295 11 -> Undefinde Action
No.22 : 4294967294 11 -> Undefinde Action
No.23 : 0 11 -> Undefinde Action
No.24 : -1 11 -> Undefinde Action
```

"Failure"となる．リスト 6 の No.19 のテストケースでは，0 と 10 を入力として期待するオブジェクトの状態を"Failure"としているため，オブジェクトの状態に対するテストケースを適切に生成できることが確認できる．ことを確認した．また，不変条件と事後条件を含む VDM++仕様を拡張後の BWDM に入力として適用し，不変条件か事後条件を満たしてない入力に対するテストケースでは，期待出力が"Undefined Action"となっていることを確認した．

4. 考察

4.1 不変条件と事前条件と事後条件に対応したテストケース生成に関する評価

拡張後の BWDM では，不変条件と事前条件と事後条件における条件式に対応したテストケースを生成できることを確認できた．これにより，拡張後の BWDM は，VDM++仕様の不変条件と事前条件と事後条件における条件式に対応した境界値及び期待出力の生成が可能となった．

したがって，定義内で設定されている条件式に対応したテストケース生成が可能となり，BWDM の適用範囲が拡大した．このことから，BWDM の有用性が向上したと言える．

4.2 型定義ブロック対応に関する評価

拡張後の BWDM では，型定義を用いた VDM++仕様のテストケースを生成できることを確認できた．これにより，拡張後の BWDM は，型定義ブロックに記述される定義に対応したテストケース生成が可能となり，BWDM の適用範囲が拡大した．

このことから，BWDM の有用性が向上したと言える．

4.3 オブジェクトの状態に対するテストケース生成機能に関する評価

拡張後の BWDM では，オブジェクトの状態に対するテストケースを生成できることを確認できた．また，操作後のオブジェクトの状態を，入力データと記述された条件式から導出して，期待するオブジェクトの状態を適切に生成できていることを確認できた．これにより，拡張後の BWDM は，既存の BWDM が生成できないオブジェクトの状態を操作する操作定義のテストケース生成が可能となった．

したがって，BWDM にオブジェクト操作のためのテストケース生成機能が追加され，BWDM の有用性が向上したと言える．

4.4 人手によるテストケース作成との比較検証

オブジェクトの状態に対するテストケース生成において，拡張後の BWDM を使用した場合と，人手の場合で，生成及び作成に要した時間の比較検証を行った．対象とした VDM++仕様は，リスト 1 の仕様である．比較検証の結果を，表 2 に示す．

表 2　比較検証の結果

	時間
被験者 5 人の平均	17m19s
BWDM	1.4s

リスト 1 の仕様では，インスタンス変数「割引券」と「カード利用額」を操作する「カードと割引券で支払う」操作が記述されている．操作後に 2 つのインスタンス変数の値が，条件式で記述されている要求を満たしているかどうかをテストするテストケースを作成する時間を計測した．この時，入力エラーとなる場合のテストケースも作成する．作成するテストケースは，入力と期待するオブジェクトの状態(表 1 参照)を記述する．

人手による検証は，本研究室の大学院生 2 人と学部 4 年生 3 人の計 5 人で行い，テストケースを書き始めてから，必要なテストケースを全て記述し終えるのに要した時間を計測した．テストケースが不正確な場合は，間違いを指摘して，被験者が正しいテストケースを記述した時点で時間計測終了とした．

拡張後の BWDM の検証では，コマンドライン上で拡張後の BWDM を実行して，テストケース生成を行うのに要した時間を計測した．

拡張後の BWDM によるテストケース生成時間と人手によるテストケース作成の平均時間を比較した結果，拡張後の BWDM を使用する場合は 17 分程度の時間短縮が確認できた．また，人手によるテストケース作成では，ヒューマンエラーが見られた．仕様の規模が更に拡大すると，人手とコンピュータの処理速度の差や，ヒューマンエラーの有無などにより，人手によるテストケース作成と BWDM によるテストケース生成に要する時間の差は更に拡大すると考えられる．

本研究で追加した，オブジェクトの状態に対するテストケース生成機能においても，既存の BWDM の特徴であったテストケース生成に要する時間を短縮できることと，ヒューマンエラーを除去できることが確認できた．

したがって，BWDM の有用性が向上したと言える．

5. おわりに

本研究では，BWDM の有用性の向上を目的として，既存の BWDM が持つ 3 つの問題点に対応する，3 つの拡張を行った．

問題点が解決できたことを確認するために，拡張後の BWDM に対する適用例を示した．不変条件と事前条件と事後条件の条件式が設定されている仕様の適用例では，不変条件と事前条件と事後条件に対応していない問題点が

解決できたことを確認した. 型定義を用いた仕様の適用例では, 型定義ブロックに対応していない問題点が解決できたことを確認した. オブジェクトの状態を操作する操作定義を含む仕様の適用例では, オブジェクトの状態を操作する操作定義のテストケースが生成できない問題点が解決できたことを確認した．これらのことから，拡張後の BWDM は，本研究で行った拡張によって上記の 3 つの問題点を解決できた.

更に, オブジェクトの状態に対するテストケース生成において，拡張後の BWDM に適用した際の生成時間と，人手による作成時間の比較検証を行った．検証の結果，検証に用いた VDM++仕様においては，拡張後の BWDM を使用する場合は平均で 17 分程度の時間短縮が確認できた．これにより，本研究で追加した，オブジェクトの状態に対するテストケース生成機能においても, テストケース生成に要する時間を短縮できることが確認でた．加えて，人手によるテストケース作成ではヒューマンエラーも見られ, 拡張後の BWDM によるテストケース生成では，ヒューマンエラーを除去できることも確認できた.

以上のことから，本研究で拡張した BWDM は有用性が向上したといえる.

以下に，今後の課題を示す.

- 整数型以外の型への対応
  BWDM は，int 型と nat 型と nat1 型のみに対応可能であり, それ以外の real 型や rat 型などの型に対応していない. これにより, 整数型以外の型を用いた仕様を入力とした場合，適切なテストケースを生成できないため，対応する必要があると考える.
- 操作後の値を参照する条件式への対応
  BWDM は, 事後条件やインスタンス変数の型の不変条件において, 操作後の値を参照する場合に, その条件式に対する，境界値分析に基づいたテストケースを生成できない．事後条件やインスタンス変数の型の不変条件に対する境界値テストも必要であるため，対応する必要があると考える.
- 操作及び関数を呼び出す操作定義と関数定義のテストケース生成への対応
  BWDM は, 操作定義と関数定義内で操作及び関数を呼び出す処理に対して，テストケースを生成できない. これにより, 関数を呼び出して計算処理を行う操作などについてはテストケースを生成できないため，対応する必要があると考える.

参考文献

1) 失敗知識データベース: 失敗事例 - みずほファイナンシャルグループ大規模システム障害. http://www.shippai.org/fkd/cf/CA0000623.html. Accessed:2022/02/13.
2) 荒木啓二郎, 張漢明: プログラム仕様記述論. オーム社，2002.
3) Overture Project. https://www.overturetool.org/method/. Accessed:2022/02/13
4) Tetsuro Katayama, Hiroki Tachiyama, Yoshihiro Kita, Hisaaki Yamaba, Kentaro Aburada, and Naonobu Okazaki: “BWDM: Test Cases Automatic Generation Tool Based on Boundary Value Analysis with VDM++”. Journal of Robotics, Networking and Artificial Life, Vol.4, No.2, pp.110-113, 2017.
5) Tetsuro Katayama, Futa Hirakoba, Yoshihiro Kita, Hisaaki Yamaba, Kentaro Aburada, and Naonobu Okazaki: “Application of pairwise testing into BWDM which is a test case generation tool for the VDM++ specification”. Journal of Robotics, Networking and Artificial Life, Vol.6, No.3, pp.143-147, 2019.
6) ロジャー・S・プレスマン: 実践ソフトウェアエンジニアリング. 日科技連出版社，2005.
7) James C.King: Symbolic execution and program testing. Communications of the ACM, 1976.
8) Lee Copeland(訳:宗雅彦): はじめて学ぶソフトウェアのテスト技法. 日経 BP 社, 2005.