独立条件記述に基づくコンポーネント条件テスト手法

4J−2

坂井 悠樹　鷲崎 弘宜　深澤 良彰
早稲田大学理工学部

1　はじめに

近年、ソフトウェア部品の組合せ開発を中心としたコンポーネント技術が、開発コスト削減とソフトウェアの信頼性向上を目的に注目されている。本稿では、コンポーネントとはオブジェクト指向言語により実装した、独立して交換・再利用可能なソフトウェア構成単位を指す。

再利用を目的として開発されたコンポーネントは、コンポーネント開発者が想定しないような使われ方をする可能性があるため、徹底的な単体テストをコンポーネント開発者が行う必要がある [1]。一方、利用者は、利用対象コンポーネントの動作を単体テストにより確認したいという要求を持つ。単体テストの手段の一つとして、条件テストがある。しかし、従来の条件テストを実現する手法では、コンポーネント本体のコード部分と条件テストスクリプト（条件テスト用コード）の明確な分離、および柔軟な条件記述の変更を同時に実現できなかった。

本稿では、比較的細粒度なコンポーネントに対して、可観測性の高さを利用して、条件テストスクリプトを自動生成し、さらに条件記述を別ファイルとして分離することで、条件テストにおける問題を解決する手法を提案する。本手法が提案するテストスイートは、テスト対象コンポーネントのソースコードを必要としない。

2　条件テスト

クラスに対する条件テストは、クラスが実行時に満たすべき論理的正しさをテストすることを目的とする [1]。満たすべき条件として、クラス不変条件（クラスが有効であるとき真でなければならない条件）・事前条件（メソッドを実行するときに真でなければならない条件）・事後条件（メソッドの実行を終了したときに真でなければならない条件）・ループ不変条件（ループを実行するとき常に真でなければならない条件）が挙げられる。

要求される仕様とソフトウェア本体の働きが異なることを避けるために、プログラムそのものに仕様として埋め込まれる記述が表明である。例えば、assert(money > 0) は、状態 money が 0 より大きいという条件を満たしていなければならないことを意味する。表明を条件テストスクリプトの実現方法として用いるとき、ハイゼンバグ [2] に注意する必要がある。ハイゼンバグとは、テスト用のコードがテスト対象ソフトウェアの本来の挙動を変更してしまうバグである。

3　従来の実現手法

条件テストの実現手法として、テスト対象に条件テストスクリプトを埋め込む手法がある（埋込み手法）[1]。埋込み手法は、可読性・条件記述変更時の柔軟性を欠く。また、テストの実行にはソースコードが必要である。

一方、条件テストスクリプトをソースコード中のコメントとして記述し、本体コードと条件テストスクリプトの分離を図る試みとして、iContract[3] などのコメント手法がある。コメント手法では、条件テストスクリプトと条件記述の分離は実現していない。また、テストの実行にソースコードを必要とする。

また、テスト対象クラスに対して表明定義クラスを作成し、バイトコード編集により条件テストの実行を行うクラスを生成する手法 [4] がある。この手法は、本体コードと条件記述の分離を果たしているが、条件テストスクリプトが条件記述に依存し、頻繁な条件記述の変更に向かない。また、事前条件のみを対象としている。

以上 3 つの手法は全てハイゼンバグの危険性を持つ。

4　コンポーネント条件テスト手法の提案

我々は、細粒度コンポーネントの状態観測性の高さを利用した条件テスト手法を提案する。本手法において、条件記述の値の比較対象を、コンポーネントの実行時に外部から観測可能な状態およびメソッドの引数・戻り値に限定し、記述対象をメソッドの事前・事後条件に絞る。本手法の概要を、テスト対象コンポーネント例として、Account とした場合について、図 1に示す。テスト対象コンポーネントは状態 money を持ち、観測操作 getMoney によって外部に公開されている。テスト用コンポーネント生成ツールにより、テスト対象コンポーネントのテスト対象メソッドをオーバーライドし、テスト対象コンポーネントを継承したサブクラスとなるテスト用コンポーネント内に条件テストスクリプトを埋め込む。テスト担当者は、テストドライバを起動する際に、テストの対象とするクラス・メソッドを指定したテストケース記述を外部から与える。テスト対象メソッドが満たすべき事前・事後条件の記述は、条件記述ファイルとして外部から与える。以上の設計より、条件テストスクリプトと本体コー

Software Components Conditional Testing Technique Based on Separated Conditional Descriptions
Yuhki Sakai,Hironori Washizaki and Yoshiaki Fukazawa.
School of Science & Engineering,Waseda University.

ドの分離、および条件テストスクリプトと条件記述の分離を実現する。なお、テスト結果は最後にレポートとして報告される。

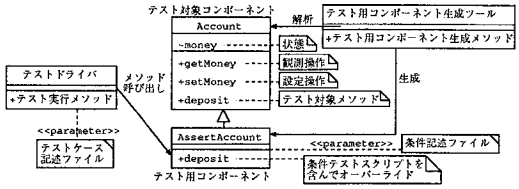

図 1: 本手法の概要

条件記述の記述形式を RTN を用いて図 2に示す。

状態の型を数値型・文字列型・オブジェクト型の 3 種に分類して扱う。数値型の検証は、等しい、大きい、小さいの各記述により行う。文字列型は、文字列が一致もしくは完全に含む・含まないという条件を検証する。なお、Boolean 型も文字列型として処理する。オブジェクト型は、インスタンスが存在するか、しない (null) かについて検証する。図 3に例を示す。

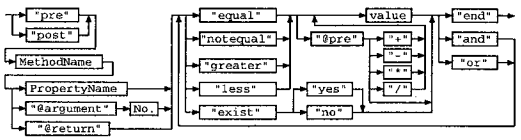

図 2: 条件記述の記述形式

```
<数値型>
pre, deposit(int), money, greater, 0, end
post, deposit(int), money, greater, @pre+100, end
post, setId(int), id, less, 100, or, more, 50, end
<文字列型>
pre, setLabel(java.lang.String), @argument1, greater, foo, end
post, getLabel(), @return, less, bar, end
<オブジェクト型>
pre, addPanel(java.awt.Panel), container, exist, yes, end
```

図 3: 条件記述の具体例

5 評価

評価サンプルとして、埋込み手法およびコメント手法によって条件テストを実現している 5 つのサンプル A ～ E を対象とする。サンプル中の条件テストスクリプトは合計で、事前条件数 16, 事後条件数 18, クラス不変条件数 10 である。

埋込み手法、コメント手法、および本手法とで、通常ソースコード内に埋め込む必要がある、メソッドあたりのコード量を比較した結果を、表 1 (追加行数) に示す。また、表 1(追加合計) に、全サンプルに対する追加コード行数の合計を示す。本手法の必要行数は従来手法よりも多くなるが、本手法が必要とする条件テストスクリプトは全て自動生成されるため、コーディングによってテスト実行者にかかる負担はほぼゼロに近いと言える。

また、埋込み手法・コメント手法と本手法とで、条件記述を変更する際に必要なテストコードの修正行数の比較結果を表 1中の修正行数欄に示す。従来手法では条件記述変更の度にテスト対象コンポーネントのソースコードを修正し、再コンパイルする必要がある。対して本手法では、条件記述に変更があった場合、条件記述ファイルに手を加える必要はあるが、、一度生成されたテスト対象コンポーネント (テストコード) は、一切変更する必要がない。よって、本手法は従来手法に対し、条件テスト仕様変更時における保守コストが低い。

表 1: 必要追加コード行数および必要修正コード行数

手法	追加行数	追加合計	修正行数
埋込み手法	条件記述数	21	1
コメント手法	条件記述数 +2	31	1
本手法 (自動)	13 または 15	219	0
本手法 (手書)	0	0	0

埋込み手法・コメント手法による記述に対する本手法の記述の網羅性を、全評価サンプルについて比較した。比較結果のうち、事前条件を図 4(a) に、事後条件を図 4(b) に示す。図 4において、事前条件はサンプル記述の平均して約 9 割 (88%) が網羅されていることから、本手法における事前条件に関する網羅性は高いことが分かる。一方、事後条件の網羅率は低い (37%)。これは c.contains(o) などの記述ができないことが原因だが、このように自由な条件記述内でのメソッド呼び出しはハイゼンバグとなる可能性がある。

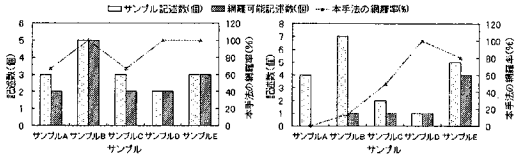

図 4: (a) 事前条件網羅率 (b) 事後条件網羅率

6 おわりに

本手法では、テスト対象コンポーネントの本体コードと条件テストスクリプトの明確な分離を実現した。機械的に同一の条件テストスクリプトを自動生成し、条件記述対象を限定した結果、ハイゼンバグの可能性を排除した。また、本体コードから条件記述部分を分離し、別ファイルとして与えることで、条件記述の作成や変更・削除に関する作業に対して条件記述以外の一切の変更を必要とせず、従来手法に対して、条件テスト仕様変更時における柔軟な対応を実現した。今後、事後条件記述の網羅性改善のために条件記述形式を見直していく。

参考文献

[1] S.Siegel, 古宮誠一, 廣田豊彦監訳, “オブジェクト指向ソフトウェアテスト技法”, 共立出版, 1999

[2] A.Hunt, D.Thomas, “The Pragmatic Programmer,” Addison Wesley, 2000

[3] R.Kramer, “iContract - The Java Design by Contract Tool,” TOOLS, 1998

[4] 田中 哲, 一杉裕志, “バイトコード編集による Java 言語の表明検査の制御”, コンピュータソフトウェア, Vol.18, No.3, 2001