B-001

高カバレッジ単体テスト設計支援ツールの活用によるソフトウェア品質向上手法の提案
Unit Test Designing Tool of Wide Code Coverage for Developing High-Quality Software

斎藤 正己† 鎌田 典彦† 高岡 真則†
Masami Saito Norihiko Kamata Masanori Takaoka

1. はじめに

安全性や品質を重視するソフトウェアの領域では、開発ベンダは自らが開発したソフトウェアが安全性の基準や品質要件を満たすことの確証を示すように顧客やソフトウェアの利用者から求められる場合が増えている。その一つとして、ソフトウェアのテストを十分に実施したことを第三者に対して確証をもって主張できることが必要である。しかし漫然とテストを実施したのでは、十分なテストである確証を得ることはできない。

本研究では、ソフトウェア開発のプロセスにおいて最初に行うテストである単体テストに焦点を当て、単体テストを十分に実施するための方法を検討した。そして単体テストの構造カバレッジ基準である MC/DC(Modified Condition/Decision Coverage)に着目し、簡単かつ確実にこの基準を満たすテストの設計方法を検討して、単体テスト設計支援ツールを開発した。

このツールの活用により、単体テストにおけるテスト項目作成作業の負荷低減、テスト項目漏れ検出などの効果を期待できる。また、ツールがテスト項目作成を支援する情報を自動的に作成するので、テスト設計者のスキルを問わずに誰でも一定の基準に従ったテスト項目を作成可能であり、作業の効率化や品質の均一化を図ることができる。

更には、単体テストの前工程である製造工程においても本ツールを活用することにより、製造レビューにおけるバグ検出の効果も期待できる。

本稿では単体テスト設計支援ツールの機能、活用方法、および期待できる効果について報告する。

2. テスト構造カバレッジ MC/DC

MC/DC は、航空機ソフトウェアのテストで用いるカバレッジ基準として考案された。航空機ソフトウェアは、万が一バグによって正常に動作しないと、多くの人命にかかわる恐れがある。そのため、MC/DC は非常に厳しいカバレッジ基準となっている。航空機同様にソフトウェアの品質基準や安全性基準が厳しい領域として、近年では車載ソフトウェアでも MC/DC を適用している[1][2][3][4]。

MC/DC ほど厳しくないカバレッジ基準としては、命令網羅 (SC:Statement Coverage) 、判定網羅 (DC:Decision Coverage)、条件網羅(CC:Condition Coverage)、条件判定網羅(CDC:Condition Decision Coverage)などがある[5][6]。どのカバレッジ基準を適用するかは、そのソフトウェアに対する品質要件や安全性基準に依存するため、一概にはこれらの基準に対する優劣をつけることは出来ない。

本研究ではミッションクリティカルなソフトウェア領域全般を対象に、MD/DC を単体テスト基準として適用する。

†日本電気通信システム（株） NCOS ラボラトリ,
NCOS Laboratory, NEC Communication Systems, Ltd.

3. 単体テスト設計支援ツール

3.1 ツール機能概要

MC/DC は非常に厳しいカバレッジ基準であるため、それを満たすテスト項目を手作業で作成するのは容易でない。そこで我々は、ソースコードを入力すると MC/DC を満たすテスト項目を作成する作業を支援するための情報を自動的に生成するツールを開発した。本ツールが出力する情報は、「真理値表」、「ソースコード塗り分け」、「制御フロー図」の 3 種類の形式である。

例として図 1 に示すソースコードを入力した場合の出力結果を図 2、図 3、図 4 に示す。

```
1   int calcM(int *rawData, int size) {
2       double sum = 0;
3       int result = myFalse;
4
5       if (size != 0 && size < NUM) {
6           for ( i=0; i<size ; i++) {
7               sum += rawData[i];
8   }
                    ⋮
```

図 1　入力ソースコード

図 2 は本ツールが出力するテスト実行条件:1 に基づいてテストを実施する場合に、ソースコード内の分岐の各条件を true/false いずれとすべきかの真理値表を示す。例えば図 2 では、ソースコード 5 行目の分岐文の判定式「size != 0 && size < NUM」において、「size != 0」の値は true、「size < NUM」の値は true、判定式全体の値は true であることを示す。なお、本ツールでは真理値表を満たす具体的なテスト入力値を自動生成しないため、テスト設計者が設計書をもとに決定する。

テスト実行条件:1

判定式: size != 0 && size < NUM
（行番号:5）

size != 0	size < NUM	判定
true	true	TRUE

判定式: i<size
（行番号:6）

i<size	判定
true	TRUE

図 2　真理値表

図 3 は、テスト実行条件#1 のテストを実施する場合に、ソースコード上のどの部分が動作するかを赤い太字の斜体で示す。

```
1   int calcM(int *rawData, int size) {
2     double sum = 0;
3     int result = myFalse;
4
5     if (size != 0 && size < NUM) {
6       for ( i=0; i<size ; i++) {
7         sum += rawData[i];
8   }
                ⋮
```

図 3　ソースコード塗り分け

図 4 は、図 1 のソースコードをフロー図に変換し、更にテスト実行条件#1 によって動作する部分を赤い実線で示す。

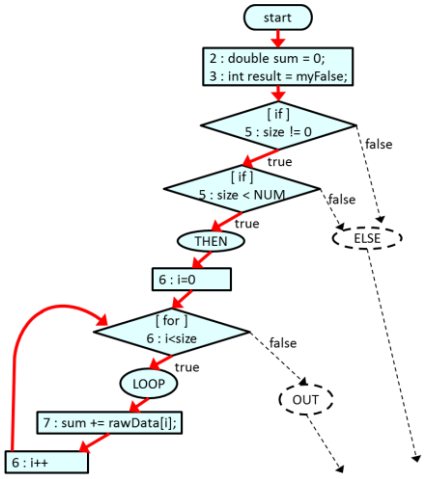

図 4　制御フロー図

このように、本ツールは入力したソースコードから求まる全てのテスト実行条件に対して、図 2、図 3、図 4 で示す情報を出力する。

3.2 カバレッジ低下を防ぐ工夫

現実のソースコードでは、同じ変数を複数の分岐で条件として用い、かつ分岐と分岐の間で変数の値が変化しないことがある。もし、各分岐でこの変数の値が互いに異なる条件値を選ぶと、実施不可能なテスト実行条件となる。

例えば図 5 のソースコードで分岐 1 と分岐 2 の間で変数 a の値を書き換えない場合は、分岐 1 の条件「a<10」と分岐 2 の条件「a<10」の結果は必ず同じである。そのため、分岐 1 の「a<10」を TRUE、分岐 2 の「a<10」を FALSE とするテスト実行条件を作成すると、テスト実施は不可能である。もし実施不可能なテスト実行条件を数多く生成すると、テスト実施可能な項目数が減り、十分なテストを行えず、MC/DC カバレッジを満たせない問題が発生する。

```
if (a < 10) { //分岐1
   ・・・・
} else {
   ・・・・
}
 ・・・・
if (a < 10) { //分岐2
   ・・・・
} else {
   ・・・・
```

図 5　複数分岐のソースコード例

そこで本ツールでは、複数の分岐で同じ条件式が登場し、かつ分岐と分岐の間の処理でその条件式で用いる変数を書き換えない場合は、各分岐の条件式の値は必ず同じとなるようにテスト実行条件を作成する。一方、分岐と分岐の間の処理にて前述の変数を書きかえる可能性のある場合は、各分岐の条件式の値は必ずしも一致しないと見なしてテスト実行条件を作成する。

この工夫により、テスト実施不可能なテスト実行条件の作成を抑止し、MC/DC カバレッジを満たすことができる。

4. 効果の検証

試作版ツールを作成して社内のソフトウェア開発現場の製造工程および単体テスト工程で試行し、効果を検証した。

①製造工程

ソースコードを入力してツールの出力結果をレビューする。テスト実行条件の内容が不適切である場合は、ソースコードにバグがあると見なし、ツール出力結果のおかしな部分を参考にソースコードの問題箇所を特定する。例えば設計書に記載していない不要な動作を間違ってコーディングした場合、設計書に基づいたテスト項目によるテストではそのコード箇所を見逃す恐れがある。しかし本ツールではそのコード箇所を通るテスト実行条件を出力するので、レビューにて不要なコード箇所を発見できる。

②単体テスト工程

設計書をもとに人手で作成したテスト項目と、本ツールが出力したテスト実行条件を突き合わせて、テスト項目の漏れ発見や項目強化を行う。例えば前者に含まれないテスト実行条件が後者に含まれる場合は、項目作成漏れの可能性があるため、ツールの出力結果をもとに項目を補充する。なお①で述べたとおり、ソースコードにバグがあると本ツールの出力する内容も適切でないので、必ずテスト設計者がツール出力結果の正当性を確認してから正規のテスト項目として扱うか否かを決定する。

上記①製造工程では、あるプログラマがソースコードのセルフレビューに活用し、レビュー工数を削減しつつバグ検出率も向上した。②単体テスト工程では、あるテスト設計者がテスト項目漏れチェック作業の工数を従来よりも 50%削減できた。このように利用者からは概ね好評を得た。

5. まとめ

現時点では小規模な試行を行ったのみであるが、利用者から好評を得ることができた。よって今後は広範囲にツールを適用し、より多くのデータを収集し、ツール活用による定量的効果の測定を継続して行う。

参考文献

[1] Kelly J. Hayhurst, Dan S. Veerhusen, John J. Chilenski, and Leanna K. Rierson, "A Practical Tutorial on Modified Condition / Decision Coverage", NASA/TM-2001-210876 (2001).
[2] Thomas K.Ferrell, Uma D.Ferrell, "RTCA DO-178B/EUROCAE ED-12B", The Avionics Handbook, SECTION IV Software, 27 (2001)
[3] ISO, "ISO26262 Part 6: Product development at the software level" (2011)
[4] 松本充広, "MC/DC による現実的な網羅のススメ", キャッツ株式会社, http://www.zipc.com/cesl/info/03.pdf (2009)
[5] Glenford J.Myers, 長尾真, 松尾正信, "ソフトウェア・テストの技法　第 2 版" (2006)
[6] Boris Beizer, 小野間彰, 山浦恒央, "ソフトウェア・テスト技法" (1994)