

18. テストプログラム記述言語†

迫 田 行 介††

1. はじめに

近年、プログラムの開発量は年々増大していく傾向にあり、その生産性、信頼性の向上が叫ばれ、ソフトウェアエンジニアリングの研究がさかになってきていることは周知のとおりである¹⁾。

しかし、大規模なプログラムを組織的に開発する場合には、相変らず設計工程、製作工程の後のテスト工程で膨大な量のテストケースを実施し、設計誤り、製作誤りを見つけ、手直しをしているのが現実の姿であることも事実である (図-1)。

このような手順は、現在のプログラム生産技術では避けられないものであり、プログラムが正しく動くらしいことを確認するためにテストをすることは必須であり、信頼性を高めるためには、十分に確認できるまでテストケースをつくり、実施しなければならない²⁾。

現在、テストケースジェネレータ、テストドライバ、テスト網羅性チェックなどのテストツールやテスト技術の実用化の研究・開発が行われてきている。

テストプログラム言語もその1つであり、テストシステムの入力言語として大規模なプログラムの開発時テストで生じる種々の問題を解決させることを目的とした問題向き言語である (図-2)。

2. テストの実施上の問題点

テスト作業は確認作業とデバッグ作業とからなる。確認のためのテストランの結果によってデバッグ作業が発生する。デバッグ作業がプログラムの誤りがどこにあるかを探索し、見出すことであるのに対して、テスト作業は、大量のテストケースを実施し、その結果を確認していくことである。したがって、いかに能率よく実施していくかに重きがある (図-3)。

テストの実施に関しては、次のような問題がある。

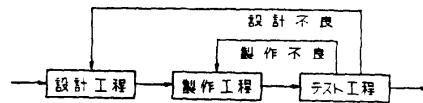


図-1 プログラムの開発工程

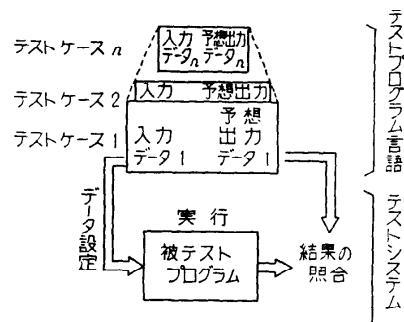


図-2 プログラムのテスト

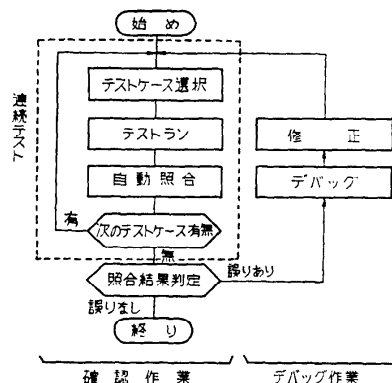


図-3 連続テストでの作業の流れ

(1) テスト方法の標準化

テスト方法が、担当者の経験、能力によりまちまちになりがちである。個人差をなくし、誤りを防ぎ、均一レベルのテストができるように、テスト方法を標準化する必要がある。

† Test Program Language by Kousuke SAKODA (Systems Development Laboratory, Hitachi Ltd.).

†† (株)日立製作所システム開発研究所

(2) 分離記述

被テストプログラム（テストされるプログラム）の中にテストに必要な文を埋め込む方法では、埋め込み時やテスト終了後の復元時に操作誤りを起こしやすく、また、ドキュメント性も悪い。さらにテスト内容の変更や再テストにも不都合である。

テストプログラムを被テストプログラムと混在させないことが必要となる。

(3) テストの自動化

大量のテストケースを短時間に実施するために、テストケースを自動的に連続実行できる必要がある。実行時エラーでデータを破壊したり、暴走して命令例外割り込みを生じたり、無限ループに飛び込んだりした場合にも、次のテストケース以後が支障なく、人手の介入なしに実施できる必要がある。

(4) ドキュメント性

どのようなテストをどのように実施し、どのような結果となったかというような記録としてのドキュメントは、作業を分担しながらプログラムを開発していく場合には非常に重要なものであるが、その作成は大変な労力を必要とするものである。

テストプログラム記述言語自身およびその実行時の出力結果がそのままドキュメントとして利用できるものである必要がある。

3. テストプログラム記述言語の機能

テストプログラム記述言語に必要な主な機能には次のものがある。

(1) データエリア宣言機能

パラメータエリアや外部データエリアを確保する。

(2) データ設定機能

パラメータやデータエリアに値を設定する。大量のデータエリアをファイルに退避し、回復する機能があると、再設定が容易になる。

(3) 自動照合機能

実行結果と予想値の照合を自動的に行う。実行結果の予想値はあらかじめ与えておくべきであり^{1),2)}、連続実施するためには必須の機能である。

(4) ブレイクポイント機能

実行途中で中断点を設定する機能。途中結果の確認やテスト手順の変更が必要となる。

(5) シミュレーション機能

サブルーチンをテストするときに、呼出しプログラムの代りをする driver 機能と、未接続の下位のサブ

ルーチンの代りをする stub 機能のモジュールテスト用のものや入出力命令やスーパーバイザコール命令などの模擬実行機能。

(6) テストケース記述機能

テストケース単位にテスト手続きを記述する機能。

(7) 高水準記述機能

データの指定やブレイクポイント位置の指定にソースプログラムに現われる記号名称を用いる機能。データの型やデータ構造などの情報を用いて、データ設定やデータ表示を行うことにより、ソースレベルテストが可能となる。

(8) 手続き定義機能

テスト手続きをマクロとか内部手続きとして定義しておくことで、記述量を減らすことができる。

4. 実行方式

分離記述型のテストプログラム記述言語の実行方式は、次の3つのものに大別される。

(1) 展開/挿入方式

テスト手続きを被テストプログラムの中に埋込んでしまう方式。テストの実行制御は被テストプログラム自身が行う。記号名称とアドレスの対応表などのプログラム情報も内部に組込まれる。文献3)は、ソースプログラムレベルで組込む例である。

この方式は、実行時オーバーヘッドは少ないが、処理プログラムは対象言語ごとに必要となる。

(2) インタプリト方式

被テストプログラムをトランスレータで中間語に変換し、それをインタプリタが解釈実行する。テストプログラム自身も同じ中間語に変換され、解釈実行されることが多い⁴⁾。

実行スピードは低いが機能的柔軟性は高い。インタプリタは共通にできるがトランスレータは対象言語ごとに必要となる。ホストマシンでテストする場合、中間語の代りにターゲットマシンの機械命令を用いることもある。

(3) モニタ方式

テストモニタが、被テストプログラムの実行を制御し、テストプログラムを実行する。

モニタへ制御を戻すためのトラップ命令やプログラム情報を、オブジェクトコード内に持つもの⁵⁾と、独立した表として持つもの⁶⁾に分けられる。後者では、テスト用のオブジェクトコードを別に作成する必要はない。

モニタ自身は、システムに1つ開発すればよく、言語プロセッサも表の出力部を追加するだけでよいから、多種の言語のサポートも容易である。実行時オーバヘッドが少なく、高度な機能が実現できる。

5. テストプログラム記述言語の例

これまでに発表されているテストプログラム記述言語のうち、いくつかを紹介する。

5.1 AUT (Automated Unit Test)^{11,12}

MIL-S というデータ宣言のような言語でパラメータを宣言し、かつ各テストケースの入力データと予想出力データとを記述する(図-4)。AUT はこのテストプログラムをコンパイルし、対象となるモジュールを呼出し、実行後実際の出力値と予想値とを比較し、その結果を表示する。stub 機能、driver 機能も持っている。

```
(1) IN02      PARMLIST ESTAB, UNRNAME,
               MCODE;
(2) ESTAB     PTR      ESTABLE;
(3) UNRNAME   DATA    (8) (' ');
(4) MCODE     DATA    (2) IGNORE;
(5) ESTABLE   DATA    (4) ('ESTB');
(6) SIZE      DATA    (2) ('1D');
(7)           DATA    (8) ('A');
(8)           DATA    (2) ('EP');
(9)           PTR      ESTABLE;
(10) *
(11) OUT02    COPY;
(12) MCODE    DATA    (2) ('0D');
```

説明 (1)は入力パラメータリストの始まりを示し、ESTAB, UNRNAME, MCODE をパラメータとして被テストプログラムをコールするを示す。
(2)~(9)は引数のデータ型、サイズ、値を宣言している文である。
(2), (9)はデータ ESTABLE のアドレスを値とするポインタ型データである。
(3), (5), (7), (8)は文字型のデータでそれぞれ8, 4, 8, 2 バイトの長さを持つ。たとえば(5)は値 'ESTB' を、(7)は 'A' が8つ並んだ値を持つ。
(4)はデータ長は2バイトであるが、その値は任意であることを示している。
(11), (12)は実行後の結果の確認用のデータを宣言している文である。
(11)は入力用のデータの宣言部をそっくりコピーすることを示している。
(12)はコピーしたデータと異なる部分についての確認用データを宣言している。
つまりこのテストケースでは入力データの値は実行後でも MCODE 以外は不変で、MCODE は 10 進で0の値になることをテストすることを示している。

図-4 AUT によるテストプログラムの例

5.2 MTS (Module Test System)^{13,14}

parameter 文、external 文で必要なパラメータエリアや外部エリアを宣言し、そのエリアにデータを設定したり、register 文でレジスタに値を設定し、実行さ

```
(1) PARAMETER CODA 1 CORE.
(2) PARAMETER POUNDS 9 CORE.
(3) PARAMETER PENCE 4 CORE.
(4) PARAMETER RESULT 4 CORE.
(5) CODA      C'A'.
(6) POUNDS    C'000000150'.
(7) RESULT    D(4)+150.
(8) TEST      MOD 3 CODA POUNDS PENCE.
(9) PRINT     COMPARE PENCE RESULT.
```

説明 (1)~(4)はパラメータエリアの宣言をしている文である。

(1)は CODA というパラメータ変数に1バイトをメモリ上に確保することを示している。

(5)は CODA に文字型のデータ 'A' を設定する文である。(7)は RESULT に10進バック形式で+150を設定する文である。

(8)は被テストプログラム MOD 3 を、CODA, POUNDS, PENCE をパラメータにしてコールすることを示す文である。

(9)は実行後の PENCE と RESULT を比較し、不一致であれば印字する文である。

図-5 MTS によるテストプログラムの例

せる。実行結果は print 文で印字できる。このとき、ほかのエリアの内容と比較し、一致しない場合のみ印字したり、変更されたデータのみ印字することができる(図-5)。

at 文でブレイクポイントを設定したり、patch 文でプログラムやデータを修正できる。simulate 文ではじまる文は stub の定義を与える。

内部データの参照やブレイクポイントの指定は相対バイトアドレスでしか書けない。また、予想結果を直接書くことはできない。

5.3 TPL/F (Fortran Test Procedure Language)¹⁵

Fortran の構文を使ってテスト手続きを書き、被テストプログラム(群)とともにプリコンパイルし、リンケージエディタでテストライブラリのモジュールと結合して実行する。

テスト手続きには Fortran 文の代入文、if 文などが書けるほか、verify 文でデータを判定することができるが、大量のテスト結果の照合指定は記述しにくい。

I/O Simulate 文と record 文で入出力をシミュレートしたり、retrieve 文で出力データのテストができる。テスト手続きの記述量を減らすために手続きをマクロ定義することができる(図-6)。

5.4 TPL (Test Program Language)¹⁶

TPL は制御用プログラムのモジュールテストや組み合わせテストのためのテストプログラム記述言語である。テスト手続きは1回のテストランに対応した case unit をいくつも記述したものである。

case unit には、実行タイミングに対応したいくつ

```

(1)      MODULES SUB 6
(2)      C
(3)      C  DEFINE GENERAL TEST FOR SUB6
(4)      C
(5)      MACRO TST6 (IP, JP, AP)
(6)      : I=IP
(7)      : J=JP
(8)      VERIFY (: N. EQ. AP)
(9)      EXECUTE
(10)     MEND
(11)     C
(12)     C  TEST CASES
(13)     C
(14)     TST6 (1,5, 18)
(15)     TST6 (1000, 9999, 10000)
(16)     FIN

```

説明 (1)はモジュール SUB 6 のテスト手続きの始まりを宣言する文である。
 (5)～(10)まではテストの手続きの本体を、マクロとして定義している。マクロが参照されると(6)～(9)の文の IP, JP, AP が参照変数に置きかわる。
 (6), (7)は代入文で, (1)で指定したモジュール SUB 6 の内部変数 I と J に引数の値が代入されることを示している。
 (8)は実行後に SUB 6 の内部変数 N の値を確認するためのアサーションを与える文である。これは実行途中での変数の値を確認するアサーションを与える文もある。
 (9)は被テストプログラムの実行を指示する文である。
 (14), (15)は実際にマクロを参照している文である。
 (16)はテスト手続きの終りを示す。(2)～(4), (11)～(13)はコメント行である。

図-6 TPL/F によるテストプログラムの例

かの block が記述できる。実行直前直後に実行する entry, exit block とブレイクポイントに対応した at block がある。また、任意の文の集まりを common block として定義し、refer 文で参照できる。

block の内部では、set 文で変数にデータをセットしたり、check 文、verify 文で変数のデータ値を自動照合する。verify 文では自動照合後、データの予想値に修正するので、その後のテストランの実行の有効性が保証される。print 文のほか、setp 文、checkp 文、verifyp 文でもデータ値を印字する。if 文で判定し、テストケースの選択もできる(図-7)。データをファイルに退避しておき、別のテストケースあるいは別のジョブで使用するための retain 文、restore 文がある。

高級言語レベルの高水準記述ができるので、テストプログラムの設計が容易で、ドキュメント性も高い。

5.5 その他

以上のほかにも、コンパイラの開発時テスト用の FADEBUG-III¹¹⁾、PL/I 型制御用言語を対策とした SOLDA⁹⁾ などがある。また、デバッグを主目的とした会話チェック型 PL/I コンパイラ⁴⁾、Fortran 会話デバッグ¹²⁾、対象言語を限定しない TEST コマンド¹³⁾などがある。

```

(1)      TEST      SUBR (IX, P)
(2)      DUMP      PRINT  'COMMON DATA', A, N,
(3)              'PARAMETER', IX, P
(4)      CINIT     SETP   A=(0.1,4(0.3)),N=5
(5)      C
(6)      CASE 1    CASE
(7)      ENTRY REFER CINIT
(8)              SETP   IX=3, P=6.2
(9)      AT        SUBR.100 DOWN 2
(10)             VERIFYP B=(0.2,4(0.6))+ -0.01
(11)             IF     IY .GT. IX+10
(12)             REFER  DUMP
(13)             SKIP
(14)             IFEND
(15)      EXIT     CHECK  B=(0.3,4(0.9))+ -0.01
(16)             CEND
(17)
(18)
(19)
(20)      END

```

説明 (1)は SUBR のテストプログラムの始まりを示す。パラメータは IX と P であることを宣言している。
 (2)～(4)は DUMP と CINIT という2つの共通ブロックを宣言している。これらを参照している(7), (12)に、それぞれが埋込まれたのと同じ意味になる。
 (6)～(16)は1つのテストケースの範囲である。
 (7), (8)は ENTRY ブロックで、このテストケースでの、実行前の手続きを与える。(8)は IX に3を、P に6.2を代入し、かつこれらの変数の値を変数名とともに印字することを示す文である。
 (9)～(14)はこのテストケースで、実行途中のブレイクポイントでの手続きを与える。(9)はブレイクポイントを SUBR 内の文番号が100である文の2行下の文の実行直前であることを指定する文である。
 (10)は配列 B の5つの要素の値を誤差 0.01 の範囲で確認し、かつ正しくないときには指定してある値に置きかえる文である。
 (11)～(14)は IF 条件ブロックで、(11)に示した条件が成立したときのみ実行される。(13)はこのテストケースの実行を中断し、すぐ次のテストケースを実行することを指示する文である。
 (15)はこのテストケースでの、実行後の手続きを与える。
 (16)はほぼ(10)と同様であるが、値を変更しない。

図-7 TPL によるテストプログラムの例

6. おわりに

ソフトウェア生産のためには多くの生産技術が必要である。テストプログラム言語はその1つであり、実際のソフトウェアを開発している生産場における重要な治工具である。

テストプログラム言語については、それを実行するテストシステム(モニタ)との一体化を通して実用化を図っていくことが大きな課題となる。

TPL¹⁰⁾は、その後対象言語に依存しないようにし、プロセスシミュレーション機能やオンラインテスト環境設定機能が追加され、計算制御用ソフトウェアの一貫テストシステム HITEST¹⁰⁾のテストプログラム言語として適用されており、テスト実行時の操作ミスや確認ミスの減少、不良の発見・対策の集中化・効率向

上などの効果が得られている¹⁴⁾。

テストプログラム言語システムを製品として入手できるものもある⁹⁾が、多くはソフトウェア製造会社が独自に開発しているのが実状である。

現在、特に言語の標準化の動きはない。しかし、受注条件として細かい受入れテストが義務付けられるようになってくると、その内容を正確に記述するために標準的なテストプログラム言語が必要になってくるかもしれない。

参 考 文 献

- 1) Myers, G. J.: Software Reliability Principles and Practice, Wiley Interscience (有沢 誠訳, ソフトウェアの信頼性, 近代科学社) (1976).
- 2) Hetzel, W. C., Ed.: Program Test Method, Prentice-Hall (鳥居宏次訳, プログラム・テスト法, 近代科学社) (1973).
- 3) Panzl, D. J.: Test Procedures: A New Approach to Software Verification, Proc. 2nd International Conference of Software Engineering, pp. 477-485 (1976).
- 4) IBM System/360 Operating System: TSO PL/I Checkout Compiler, SC 33-0033-0.
- 5) 春原, 大井, 関本, 中村: 高位言語デバッキングシステム SOLDA, 情処学会論文誌, Vol. 20, No. 5, pp. 405-411 (1979).
- 6) 迫田, 霜田, 小島, 桑原, 小崎: プログラムテストシステム TPL (1), (2), 昭 49 情処学会第 15 回大会, pp. 661-664 (1974).
- 7) Heuermann, C. A., Myers, G. J. and Winterton, J. H.: Automated Test and Verification, IBM Technical Disclosure Bulletin, 17(7), pp. 2030-2039 (1974).
- 8) MTS 概説書, JMA Systems Corporation, System Marketing Division.
- 9) プログラムテスト支援 (HMTS) 文法, HITAC 解説/文法書. 8080-3-311-20.
- 10) 海永, 薄井, 大島, 林: HITEST/F-TPL の開発, 昭 53 情処学会第 19 回大会, pp. 327-328 (1978).
- 11) Itoh, D. and Izutani, T.: FADEBUG-I, A New Tool for Program Debugging, IEEE Symposium on Computer Software Reliability (1973).
- 12) Fortran 端末使用の手引, HITAC 手引書, 8090-3-215.
- 13) IBM System 360 Operating System: TSO Command Language Reference, GC 28-6732-1.
- 14) 大島, 林, 海永, 薄井: 制御用ソフトウェア機能一貫テストシステム "HITEST/F", 日立評論, Vol. 62, pp. 47-52 (1980).

(昭和 56 年 2 月 23 日受付)