海外研究動向

UDC 681.32.06.001.4 : 519.686

ソフトウェアのテスト技法

(紹介) 中所武司

中所武司：正員　日立製作所システム開発研究所
Testing Techniques for Software Programs. By Takeshi CHUSHO, Regular Member (Systems Development Laboratory, Hitachi Ltd., Kawasaki-shi).
資料番号：昭 56-89[海外研究動向-4]

1. まえがき

ソフトウェアプログラムのテストは，その開発費用の 50% を占め，生産性向上に重要であるばかりでなく，ソフトウェアの信頼性向上のための重要課題である．特に最近のマイクロコンピュータシステムをはじめとする計算機の応用分野の広がりに伴い，ソフトウェアの品質保証に関する要求が高まっている．

そこで，本文では IEEE の Transactions on Software Engineering (May 1980)[1]~[6] 及び Computer (Aug. 1979)[7]~[18] のソフトウェアのテストに関する特集号を中心に，最近のこの分野の研究動向について述べる．

2. テストケース／テストデータ作成法

ほとんどのプログラムでは考えうる入力データの数が膨大になるため，そのすべてを試すことは不可能である．そこで実際にはそのサブセットを用いてテストを行わざるをえない．従って，限られた時間と費用の中で高い品質保証の与えられるテストデータのセットを作成する必要があり，その元となるテストケースの選択はテストの最も重要な要素である．その代表的な方法として，プログラムの構造に基づいて選ぶ構造テストと機能仕様だけから選ぶ機能テストがある．

2.1 構造テスト

これは，プログラムテスト，ホワイトボックステスト，グラステストともよばれ，プログラムの制御構造を有向グラフ化して，そのパス解析に基づいてテストケースを選ぶ．最も一般的なのは，入口点から出口点に至る経路（パス）の構成要素である分岐点から分岐点までの経路(decision-to-decision path, dd パス）をすべて網らするようなパスのセットを選ぶ方法で，次のような手順[16]で行う．

（1） すべての分岐がいずれかのパスに含まれるように，パスの最小セットを選ぶ．
（2） 各パスを通過するためのパス条件を求める．
（3） 各パス条件を満たす入力データ値を求める．

しかしながら，この方式には次に述べるような幾つかの問題がある．P1 と P2 は運用上の問題[11]，他は方式上の問題である．

(P1) 実行不可能なパスの生成

パスを機械的に選んだ場合，パス条件が常に偽となる実行不可能なものが混じる．このうち，その原因がプログラムエラーに基づくものやモジュール単位の防衛的コードの存在による場合は問題ないが，他の場合は可能なパスと換える．

(P2) 大規模システムにおける完全網らの困難さ

システム全体のテストでは，網ら性の 100% 達成はコスト的に難しい．この場合，図 1 の例のように全体を幾つかのサブシステムに分割し，各々のテストで 100% を達成する．

(P3) 全パス実行との格差

dd パスの網らと全パス実行との間の大きな隔たりを埋めるため，次のような尺度 TER (Test Effectiveness Ratio) が提案されている[5]．

$$\mathrm{TER}_{n+2}=\frac{S_e(n)+P_e(n)}{S(n)+P(n)}$$

ここで dd パスに似たプログラム要素 LCSAJ(Linear Code Sequence and Jump) を用いると，$S(n)$ は n 個の LCSAJ からなる不完全パスの数，$S_e(n)$ はそのうちの実行された数，$P(n)$ は n 個以下の完全パスの数，$P_e(n)$ はそのうちの実行された数である．この尺度の特性と効果は実験的に確かめられている．

(P4) 分岐条件自身のテストの不十分性

手順 (1) のような分岐テストでは各分岐条件の真と偽の場合が 1 度ずつ実行されれば良いため，分岐条件自身の誤りが見逃されやすい．そこで，分岐条件が比較演算式を含む場合は演算子に関係なく，>，=，< になるような 3 種類のデータを選ぶという規則[8]や，複数の条件式からなる場合は各々をテストするような方法[23]が推奨されている．

(P5) パスの欠除の検出不可

構造テストでは必要なパスがインプリメントされていないという誤りの検出は不可能であり，機能テストが必要である．

2.2 機能テスト

これはブラックボックステスト，仕様テストともよばれ，プログラムの機能仕様だけからテストケースを作成するものだが，具体的技法は少ない．Howden[14] はそのガイドライ

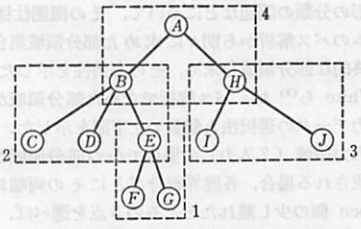

図 1　大規模ソフトウェアの統合テスト法

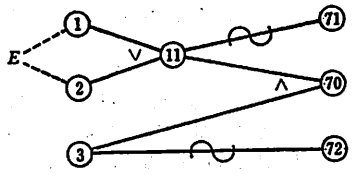

図 2 原因結果グラフの例

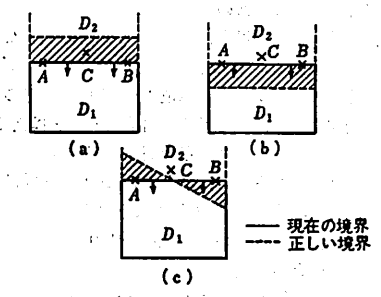

図 3 境界のずれによる3種類の誤り

ンとして，入力領域，極値（境界値），特殊値，それらの組合せ，分岐テストの併用の5項目（小項目では 17）を考え，その有効性を科学計算プログラム上で確かめた．

ツール化されている技法としては原因結果グラフ法がある[38]．これは機能仕様を組合せ論理で表すもので次の手順による．

（1） 仕様を適当に分割し，各々から原因と結果を識別する．

（2） 制約条件を付加した原因結果グラフを作成する（図2）．

（3） このグラフをある規則の下で決定テーブルに変換する．

（4） この決定テーブルの各列をテストケースに変換する．

2.3 領域法

これは入力領域を同じ結果を導く部分領域に分割して，各々からテストデータを選択するもので，具体的方法に依存して構造テスト，機能テスト，あるいはその併用と考えられる．

Weyuker ら[1]は，テスト選択基準が reliable でかつ valid ならば ideal であるという従来の Goodenough らの理論[37]は全入力のテストを必要とするなどの問題があることを指摘し，これを改良した領域法を提案した．すなわち，ある部分領域 S 内の一つの入力が誤りを検出するときは必ず基準 C を満たす任意のテストセットも誤りを検出する場合，C は S に対して revealing である，というプログラムに依存しない概念を導入し，これに基づいて部分領域を求める．具体例として3角形の分類の問題などについて，その機能仕様およびプログラムのパス解析から別々に求めた部分領域集合の重ね合せで最終的な部分領域を求め，その有効性を示した．

別に White ら[2]は，パス解析で求めた部分領域からの効果的な入力データの選択法と個数の上下限を示した．例えば，2次元の入力領域（2入力）の場合でかつ部分領域が線形不等式で形成される場合，各境界線分ごとにその両端および境界から open 側の少し離れたところの3点を選べば，図3に示すような3種類の境界のずれをすべて検出できる．

3. テストツール

テスト技法の中には自動化ツールを用いないものも多いが，プログラム解析，特定の種類のエラー検出，エラーの可能性の示唆，テストケース及びテストデータの選択のための情報提供，テスト実行結果のドキュメント化などに関するツールが開発されている．例えば，Fortran で記述されたプログラムを対象としたものでは，静的解析用の FACES, DAVE，動的解析用の PET，両機能を有する RXVP，記号実行のための DISSECT，ATTEST などがある．

これらはいずれも単品ツールであるが，これに対し，これらの機能を統合したツールとして SADAT[6],[8] がある．これは Fortran の単体モジュールテスト用で次の四つの機能からなる．

（1） 静的解析：プログラムを有向グラフ化すると共に，実行されない文，定義や参照の欠除などのエラーを検出する．

（2） テストケース生成：すべての dd パスを1度以上実行するようなパスのセットを選択する．

（3） パス条件の計算：記号実行によって各々のパス条件を計算する．結果の条件は入力，定数，演算子で構成される．

（4） 動的解析：DO, IF, 計算形／割当形 GOTO 文に CALL 文を埋込み，実行回数を収集，解析し，実行プロフィールや未実行の dd パス，データフロー，変数の値域などを出力する．

更に，総合ツールのため共通データベース利用の利点もある．一方，(2)では 2.1 の P1 の問題，(3) では計算時間の問題があり，基本的パスをユーザが指定する機能を設けている．

4. 品質評価

4.1 エラー分析

既に発生したエラーの分析は，エラーの防止や検出，あるいは残留エラー数の推定などに有用であるが実際にはあまり行われていない．そこでまずエラーの分類法を標準化しようという提案[18]がある．すなわち，次のように3種類の分類を行う．

（1） 作成フェイズ：要求定義，設計，コーディング，テスト，保守，修正的保守など．

（2） 原因：設計(5)，インタフェース(4)，データ定義(3)，論理(6)，データ処理(7)，計算(6)，その他(6) 但し，括弧内は小項目数．

（3） 重要度：critical（致命的），major（大），minor(小)．

幾つかの分析結果では，エラーの 60% 以上が設計フェイズで作られ，原因では設計エラーが 20～25% と多い．

4.2 品質評価法

幾つかの品質評価法の一つとして，2.1 で述べたパス網羅率を用いる方法がある．Sorkowitz[9] は品質管理スタッフが Cobol 用のプロフィール解析ツール CIP を用いて品質管理を行った経験を報告している．まずプログラム開発者にすべての分岐とすべての文を1度以上実行すること（最小テスト基準）を義務づけ，それを満たさないものは差しもどすようにした．その結果，初回の検査で合格するものはわずか 31

%，分岐の実行率は 56%，文の実行率は 82% であった．又，100% 達成が困難なものは，それを免除する制度も設けたが，それに該当するものは 7% であった．

又，Holthouse[11] らは，Fortran 用の dd パス網ら率測定ツールを機能テストの後で用い，網ら率 90% (システムレベル) 達成までは受入れテストを行わないようにした．その結果，機能テスト直後の網ら率は 70～85% であった．

このほか，網ら率によらない方法として，意図的に作り込んだエラーの検出率からテストケースの品質を推定する確率的手法[10]がある．又，品質に対する要求が特に厳しい分野では独自の基準が決められる．例えば，米軍の兵器システムソフトウェア用の基準 MIL-STD-1679 では，許容エラー率 (命令数比) がエラー重要度に応じて定められている[12]．IEEE コンピュータ部門ソフトウェア工学技術委員会の標準化小委員会でも，ソフトウェアの品質保証基準の設定作業が行われている[13]．

5. テスト技法の適用評価

Geiger[8] らは，テスト技法の有効性を調べるため，3人のプログラマに同一の形式的仕様を与えて，構造化 Fortran (Iftran), Pascal, 構造化マクロアセンブラを用いて開発させたプログラムを別の3人がテストする実験を行った．そして Iftran プログラムを 3. で述べた SADAT と RXVP を用いてテストした結果，静的解析はエラー検出に有効，動的解析はテストの完全性の測定に有効，又，現状では相補的な異なるテスト技法を組み合せる方式が効果的であることなどが確認された．

Gannon[10] はエラーが既知の 5,000 行の Fortran プログラムを静的および動的に解析し，それらのエラー検出能力を調べた．その結果，静的解析の検出率は 16% だが経済的，動的解析は多くの計算時間を要し，検出率は 45% 程度であった．

Glass[17] も 75 K, 150 K 命令の二つのプログラムのテストに用いた各技法のエラー検出能力を調べた結果，シミュレータで 90 件，データトレース，網ら解析，アサーション，ダンプで各々 40 件前後，論理トレースで 14 件のエラーが検出された．

6. そ　の　他

以上の章ではデータフロー解析，記号実行などの静的解析技法そのものについては詳しく述べなかったが，これらの研究は70年代半ばのものが多い．最近では，同期処理の表現形式を追加したフローグラフを用いて，従来のデータフロー解析技法を平行処理プログラムに拡張したものや[4]，Cobol 風言語で記述された事務用プログラムのすべての分岐を1度以上実行するようなパス集合を選び，その各パス条件を記号実行によって求める SMOTL システム[18]などがある．

又，プログラムに情報収集コードを埋め込むことにより，従来，静的に解析されていたデータフローを動的に解析すれば，より簡単かつ広汎にエラー検出できるという報告[19]がある．

テストと表裏一体のデバッグ技法についても特に章を設けて述べないが，既に多くのツールが実用になっている．最近では，高級言語プログラムにデバッグ用の文をそう入することなく，シンボリックなオペランド指定のできる会話形デバッグツール AIDS[20]や，アサーション機能を導入し，それが不成立なところでの中断機能を有するアセンブラ用デバッグツール ALADDIN[21]，あるいはトップダウン方式のデバックの推奨[22]などがある．

7. む　す　び

これまで述べてきたテスト技法は必ずしも大規模ソフトウェアにすぐ適用できるものばかりではない．計算機時間の問題はハードウェアの進歩で解決するとしても，利用者の準備作業，複雑な会話形処理，結果確認作業などに多くの手数を必要とするものは普及が難しいので，より使い勝手の良いツールにしていく必要がある．

本文では，テスト技法として，計算機言語で記述されたプログラムの検証を目的とするものに限ったが，本来，テストはソフトウェア開発工程全体の各段階で行うべきものである[16]．

特に，

(1) エラーの多くが設計フェイズで作り出される[15]，

(2) エラーの検出が要求定義，単体テスト，システムテストの各段階で行われたときのコスト比は，1：5：36 である[9]，

などの事実からも，要求定義技法，設計法，プログラミング法も含めた，一貫したテスト思想が必要であろう．

なお，今回は最近2年間の論文を中心に紹介してきたが，それ以前の研究については，文献に掲げた解説書[23]や論文集[24],[25]が参考になろう．

文　献

(1) Weyuker, E.J. and Ostrand, T.J. : "Theories of program testing and the application of revealing subdomains", IEEE Trans. Software Eng., SE-6, 3, pp. 236-246 (May 1980).

(2) White, L.J. and Cohen, E.I. : "A domain strategy for computer program testing", ibid., pp. 247-257.

(3) Foster, K.A. : "Error sensitive test cases analysis (ESTCA)", ibid., pp. 258-264.

(4) Taylor, R.N. and Osterweil, L.J. : "Anomaly detection in concurrent software by static data flow analysis", ibid., pp. 265-278.

(5) Woodward, M.R., Hedley, D. and Hennell, M.A. : "Experience with path analysis and testing of programs", ibid., pp. 278-286.

(6) Voges, U., et al. : "SADAT—an automated testing tool", ibid., pp. 286-290.

(7) Miller, E. : "Software quality assurance", Computer, 12, 8, pp. 7-9 (Aug. 1979).

(8) Geiger, W., et al. : "Program testing techniques for nuclear reactor protection systems", ibid., pp. 10-18.

(9) Sorkowitz, A.R. : "Certification testing : a procedure to improve the quality of software testing", ibid., pp. 20-24.

(10) Gannon, C. : "Error detection using path testing and static analysis", ibid., pp. 26-31.

(11) Holthouse, M.A. and Hatch, M.J. : "Experience with automated testing analysis", ibid., pp. 33-36.

(12) Bowen, J.B. : "A survey of standards and proposed metrics for software quality testing", ibid., pp. 37-42.
(13) Buckley, F. : "A standard for software quality assurance plans", ibid., pp. 43-49.
(14) Howden, W.E. : "Applicability of software validation techniques to scientific programs", ACM Trans. Program. Lang. & Syst., 2, 3, pp. 307-320 (July 1980).
(15) Bowen, J.B. : "Standard error classification to support software reliability assessment", Proc. NCC, pp. 697-705 (May 1980).
(16) Goodenough, J.B. and McGowan, C.L. : "Software quality assurance : testing and validation", Proc. IEEE, 68, 9, pp. 1093-1098 (Sept. 1980).
(17) Glass, R.L. : "A benefit analysis of some software reliability methodologies", ACM SIGSOFT Software Eng. Notes, 5, 2, pp. 26-33 (April 1980).
(18) Bicevskis, J., et al. : "SMOTL—a system to construct samples for data processing program debugging", IEEE Trans. Software Eng., SE-5, 1, pp. 60-66 (Jan. 1979).
(19) Huang, J.C. : "Detection of data flow anomality through program instrumentation", IEEE Trans. Software Eng., SE-5, 3, pp. 226-236 (May 1979).
(20) Hart, J.J. : "The advanced interactive debugging system (AIDS)", ACM SIGPLAN Notices, 14, 12, pp. 110-121 (Dec. 1979).
(21) Fairley, R.E. : "ALADDIN : assembly language assertion driven debugging interpreter", IEEE Trans. Software Eng., SE-5, 4, pp. 426-428 (July 1979).
(22) Lauesen, S. : "Debugging techniques", Software-Pract. & Exper., 9, 1, pp. 51-63 (Jan. 1979).
(23) Myers, G.J. : "The art of software testing", A Wiley-Interscience (1979).
(24) Miller, E.F. and Howden, W.E. (ed.) : "Software testing and validation techniques", IEEE Catalog No. EHO-138-8 (Sept. 1978).
(25) Yeh, R.T. (Ed.) : "Current trends in programming methodology, Vol. II : program validation", Prentice Hall (1977).
(26) Huang, J.C. : "Error detection through program testing", ibid., pp. 16-43.
(27) Goodenough, J.B. and Gerhalt, S.L. : "Toward a theory of testing : data selection criteria", ibid., pp. 44-79.