社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－SE－149（13）
2005／7／29

統計的回帰テストのための期待出力の導出方法

高木 智彦 †　　古川 善吾 †　　山崎 敏範 †

自動回帰テストにおいて，系統的なテスト法とは別のアプローチとして統計的テスト法を導入することを提案する．統計的テスト法の主な目的は，実際の運用環境におけるソフトウェアの信頼性を予測することである．そのために，ユーザの利用の特徴を利用モデルと呼ばれるマルコフ連鎖として定義し，利用モデル上の確率に比例してランダムにテストケースを生成する．本稿の手法は，被テストプログラムの前バージョンに対してランダム生成した入力条件を適用することにより，厳密な期待出力を自動的に生成する．本手法が機能することを確認するために，小規模のクライアントサーバプログラムおよび自動テスト環境を構築した．

Generating Expected Output for Statistical Regression Testing

Tomohiko Takagi†　　Zengo Furukawa†　　Toshinori Yamasaki†

This paper proposes that the statistical testing method, which is a different approach from systematic ones, is applied for automated regression testing. The main purpose is to calculate software reliability on the actual usage environments. Therefore, it needs to define the characteristics of the users' operations as a Markov chain called usage model, and randomly generate testcases in proportion to the usage model. Under the method of this paper, random-generated test data is inputted into the previous version program of the tested one to generate expected output automatically. To confirm this method works, a small client-server program and its automated test environment had been developed.

1. はじめに

近年，テスト作業の効率化やプログラムの信頼性向上に対する要求が高まっており，xUnit をはじめとしたテスティングフレームワークや記録再生方式の自動テストツールなどが開発現場に浸透しつつある．これらの多くは回帰テストを自動化するものであり，テスト担当者はあらかじめテストケースを設計しておくことによって，プログラムの更新を繰り返す場合のテスト工数を短縮することができる．テストケースは，コードや機能を特定のテスト基準で網羅するための系統的方法に基づいて，一つ一つ手作業で設計されるのが一般的である．

本稿では，回帰テストにおける異なったアプローチとして，ブラックボックステストでありランダムテストの一種である統計的テスト法 [1, 2, 3] の導入を提案する．統計的テストの主な目的は，実際の運用環境におけるソフトウェアの信頼性を予測することである．そのために，ユーザの利用の分布を利用モデルと呼ばれるマルコフ連鎖 [4] として定義し，利用モデル上の確率に比例してランダムにテストケースを生成する．この方法は結果的に，実際の運用環境下で顕在化しやすい（すなわち信頼性に与える影響が相対的に大きい）バグを優先的に洗い出すことを可能にする．

テストケースをランダム生成する場合，入力条件は極めて容易に生成することができる反面，それに対応する厳密な期待出力を生成することが困難であることが従来から指摘されている [5]．しかし，統計的テストにおいて必要とされる大量のテストケースをすべて手作業で設計することは現実的ではない．そこで本研究では，被テストプログラムの前バージョン（以降，基準プログラムと呼ぶ）に対してランダム生成した入力条件を適用し，そこから得られる出力を期待出力とする．これを統計的回帰テスト法と呼ぶ．統計的回帰テスト法には，基準プログラムとの相違しか検出できないという制限があるものの，現実的なコストの範囲で大量のテストケース設計を行うことが期待できる．

本稿では，まず 2 章で統計的回帰テスト法の手順を示す．次に 3 章で適用実験の過程と結果を述べる．最後に 4 章で一般論を含めた考察を行う．

2. 方法

本章では，統計的回帰テスト法の手順について述べる．本手法は，(1) 利用モデルの作成，(2) 入力条件の生成，(3) 期待出力とテスト出力の生成，(4) バグの検出，(5) 信頼性の評価，の 5 つのステップから構成される．概念を図 1 に示す．

2.1 利用モデルの作成

まず，被テストプログラムに対する操作列をステートマシン図 [6] によって定式化する．ユーザの誤操作もモデル化の対象に含まれる．終了擬似状態から開始擬似状態への暗黙的な遷移の存在を仮定すると，特殊なソフトウェアを除き，ステートマシン図は吸収的である（すなわち再帰可能な状態のみから構成される）．利用モデルを完成させるためには，以下に例示する方法を用いて，ステートマシン図上のすべての遷移確率を決定する必要がある．

- 利用現場のデータに基づく方法 [3, 7]．プログラム

†香川大学大学院工学研究科
Graduate School of Engineering, Kagawa University

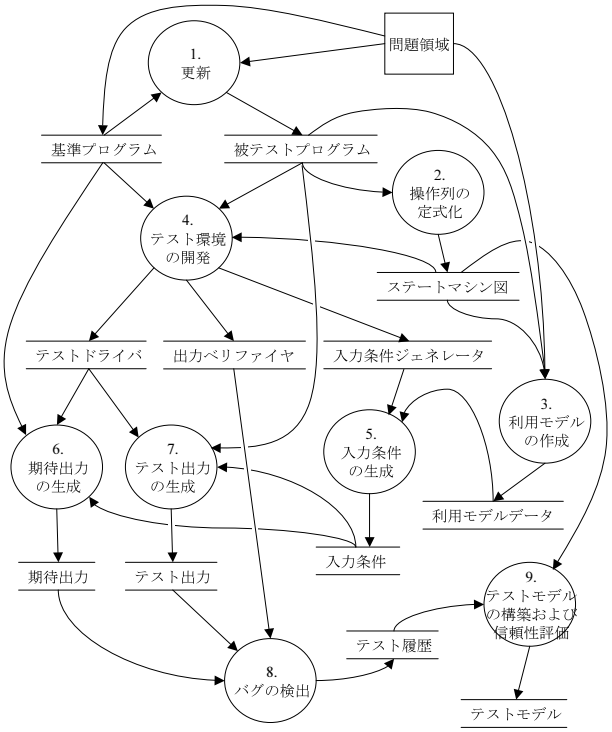


図 1: DFD で表現された統計的回帰テスト法の概念

のログや業務の生データ，作業手順などから遷移確率を算出する．ソフトウェアの信頼性を正確に推定するには，この方法を選択するべきである．

- 開発者の推定（期待）に基づく方法 [3]．悪意がなく操作に習熟したユーザならどう使うかを考える．
- 各状態の出力遷移に一様な遷移確率を割り当てる方法 [3]．利用モデルのエントロピーが最大になる．
- 各遷移の誤り確率の推定に基づいてステートマシン図をパーティションに分割し，コストや予算などから最適な遷移確率を決定する方法 [8]．

利用モデルの再利用を促進するためには，すべての遷移確率を数理計画法における制約条件に簡約する [9]．例えば，「遷移 a の遷移確率 P_a は遷移 b の遷移確率 P_b の 2 倍である」ことを制約条件として表現すると，$P_a - 2P_b = 0$ となる．ただし，再利用の際には，ソフトウェア全体としての仕様の差異が制約条件に与える影響について十分検討しなければならない．

2.2 入力条件の生成

利用モデルに基づいた入力条件を生成する．そのためには，利用モデルデータと入力条件ジェネレータを用意する必要がある．

利用モデルデータは，2.1 節で構築した利用モデルを，入力条件ジェネレータが読める形式にコード化したデータファイルである．利用モデルの変更に柔軟に対応するために，入力条件ジェネレータのプログラム本体からは分離することが望ましい．

一方，入力条件ジェネレータとは，利用モデルデータを解釈し，これに基づく確率的な手法によって入力条件を生成するためのプログラムのことである．被テストプログラムのステートマシン図に基づいて開発される．入力条件には，基準プログラムや被テストプログラムを駆動するために用いられる操作列の情報が，テストドライバ（2.3 節で述べる）によって解釈可能な形式で記述される．ただし，入力条件が基準プログラムと整合しない場合には特別な処理が必要である．ここで，被テストプログラムおよび基準プログラムにおける操作の集合をそれぞれ O_{test}，O_{stnd} のように表現した場合，問題となるのは $O_{stnd} - O_{test}$ である．これについては，操作をスキップしたり，あるいは直近の $O_{stnd} \cap O_{test}$ に接続するための最低限の操作に置き換えるなどの対応が考えられる．

2.3 期待出力とテスト出力の生成

期待出力を生成するために，2.2 節で生成した入力条件を基準プログラムに適用する．同様に，テスト出力を得るために被テストプログラムにも適用する．ここで，基準プログラムおよび被テストプログラムに入力条件を適用するために，テストドライバが用いられる．

テストドライバは，プログラムのインタフェース仕様および 2.1 節で構築したステートマシン図に基づいて開発されるプログラムである．入力条件ジェネレータから供給される入力条件を解釈実行することによって，基準プログラムや被テストプログラムを駆動し，期待出力とテスト出力を生成する．なお，入力条件はテストドライバ本体からは分離される必要がある．なぜなら，入力条件は固定的なものではなく，テストの進捗状況や利用モデルの変化に応じて動的に生成し，適用できなければならないからである．

基準プログラムと被テストプログラムは，テストに役立つ内部的なデータを同一の基準で出力するように制御できることが望ましい．必要に応じて，入力条件を生成する際に，そのようなデータを出力するための操作を挿入してもよいが，この操作が副作用を持っていないことを事前に確認する必要がある．

2.4 バグの検出

期待出力とテスト出力を比較し，バグを検出する．各出力はデータ量が多いため，この作業はプログラム（出力ベリファイヤ）により自動化される必要がある．

期待出力とテスト出力の差分は必ずしもプログラムのバグを意味しない．バグ以外のあらかじめ想定できる差分については，その条件を明示することによって無視することも可能になる．もし差分が検出されたなら，必要に応じて人手で原因を究明する．差分が生じる原因としては以下のものが存在する．

- 新しいバグの発生．基準プログラムには存在しなかったバグが，被テストプログラムに作り込まれた．
- 以前のバグの修正．基準プログラムのバグが被テストプログラムにおいて修正された．

- 実行環境の動的条件の差異．例えば，テストの実行時刻，並行動作するプロセスやスレッド，通信トラフィックなど．一方，マシンのハードウェア構成や OS などの静的条件については，2.3 節で揃えることができれば問題にならない．

- 仕様の差異．ここで，被テストプログラムおよび基準プログラムの機能集合をそれぞれ F_{test}，F_{stnd} とする．$F_{test} - F_{stnd}$ に関わるテスト出力については，基準プログラムから期待出力を得ることができないので，通常のテストと同様にテスト出力の内容を判定する必要がある．一方，$F_{stnd} - F_{test}$ はテスト対象に含まれないので，この機能集合に由来する出力は無視する．

2.5 信頼性の評価

最後に，テスト結果をステートマシン図に記入してテストモデルを作成し，信頼性を評価する．テストモデルは，利用モデルと同様でマルコフ連鎖として表される．もしバグが検出されれば，バグが検出された状態から誤り状態への遷移が追加される．このとき，バグが軽微であれば元の状態への遷移が，また，バグが致命的であれば終了擬似状態への遷移が追加される．したがって，元のステートマシン図が吸収的であれば，テストモデルも吸収的である．

信頼性の尺度には，MTTF（Mean Time To Failure）を用いることができる．MTTF は誤り状態に対する平均再帰時間とも解釈される．本テストは，目標とする信頼性の閾値を達成した時点で終了する．閾値に関しては，システムの性格や要求，基準などを勘案して決定する．

3. 適用実験

前章で述べた統計的回帰テスト法の適用実験を行った．題材として用いたのは酒類卸売業者の受付係問題（図 2）で，これは文献 [10] において提案された在庫管理システム問題を一部改変したものである．

本研究では，受付係システムをソースコード行数 4k 程度の小規模なクライアントサーバプログラムとして実装した．以降特に断りのない限り，受付係システムは，クライアントプログラムとサーバプログラムの両方を含意するものとする．使用した開発環境は J2SE（Java 2 SDK Standard Edition version 1.4.1）[11] である．この受付係システムはあくまで実験用であり，実際に運用する予定はない．本適用実験では，2 つのバージョンの受付係システムを用意した．1 つは，問題文を満たす必要最低限の機能を備えた初期バージョンで，基準プログラムとして用いる．他方は，初期バージョンに対して以下の変更を加えた最新バージョンで，被テストプログラムとして用いる．

- 空になる予定のコンテナを倉庫係に通知するために，出庫指示書に「空コンテナ搬出マーク」を追加する．倉庫係は，その場の判断で空コンテナを搬出するのではなく，出庫指示書の内容にのみ従うものとする．

ある酒類卸売業者の倉庫では，ビン詰めの酒を積載したコンテナが毎日数個搬入されてくる．1 つのコンテナには，その容量の許す限り複数の銘柄を混載できる．扱い銘柄は約 100 種類ある．倉庫係は，コンテナを受け取ってそのまま倉庫に保管し，積荷票を受付係へ手渡す．また受付係から出庫指示を受けると，在庫品を古いものから出庫することになっている．コンテナの内蔵品は別のコンテナに詰め替えられたり，別の場所で保管されることはない．空になったコンテナはすぐに搬出される．移送や倉庫保管中に酒類の損失は生じない．

さて，受付係は契約先の小売店から毎日数十件の出庫依頼を受け，その都度倉庫係へ出庫指示書を出すことになっている．出庫依頼は出庫依頼票または電話によるものとし，1 件の依頼では 1 銘柄のみに限られている．在庫量が不足の場合には，その旨依頼者に電話連絡する．同時に在庫補充依頼票に記入し，酒造業者に不足品の注文を行う．そして当該品の在庫が必要量あった時点で，倉庫係に不足品の出庫指示をする．

受付係が扱う帳票の内容は次の通りである．

- 積荷票：コンテナ ID，搬入日時，{ 注文番号，品名，数量 } の繰り返し
- 出庫依頼：注文番号，注文日時，品名，数量，送り先名
- 出庫指示書：注文番号，品名，送り先名，{ コンテナ ID，数量 } の繰り返し
- 在庫補充依頼：注文番号，注文日時，品名，数量

受付係の仕事を自動化する計算機プログラム（受付係システム）を作成せよ．なお，この課題は現実的でない部分もあるので，入力データのエラー処理などは簡略に扱ってよい．

図 2: 酒類卸売業者の受付係問題

基準プログラムと被テストプログラムの間のソースコード差分について，標準的な Unix ツールである diff（GNU diffutils version 2.8.4）で調べた結果，修正 5 行（内，コメント行は 3），追加 30 行（内，コメント行は 5），削除 0 行であった．ただし，空白文字や空白行に関する差異は無視している．このソースコード差分はすべて上記の機能追加に由来するものである．本適用実験では，このソースコード差分が従来の機能を損ねていないことを統計的回帰テスト法によって確認する．

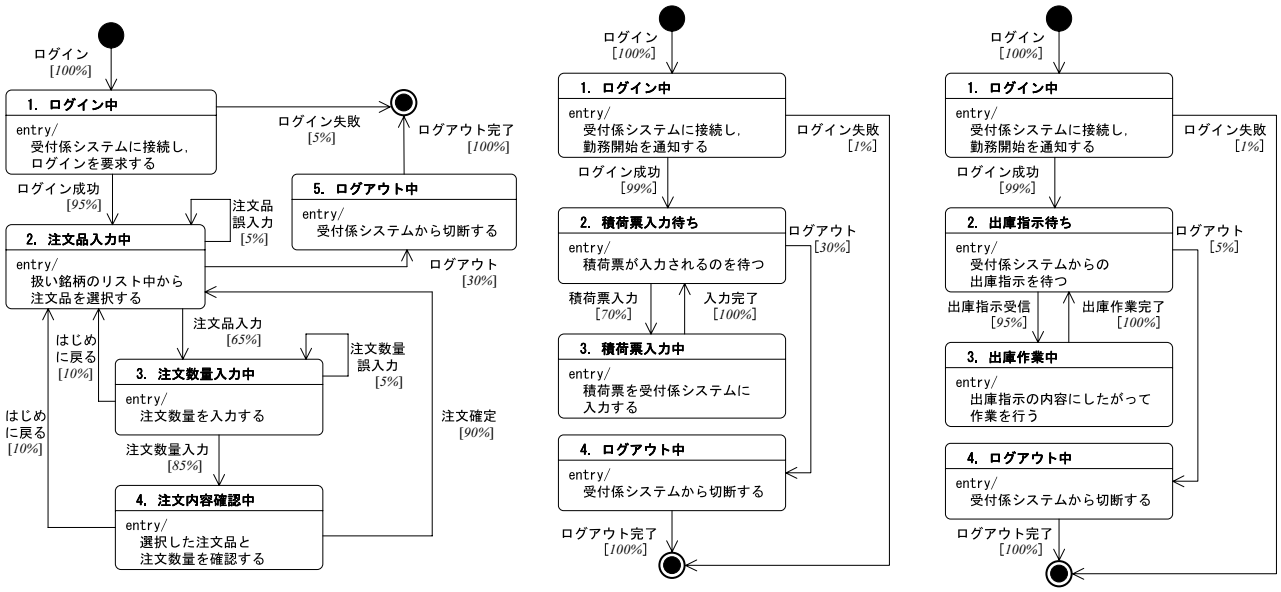
以下に，作業の過程と結果を示す．

Step 1．利用モデルの作成

受付係システムに関わる外部実体は，小売店と倉庫係である．倉庫係は搬入係と出庫係に分けられる．これらの外部実体の振る舞いは以下のようなものであると仮定する．

- 小売店

  小売店は，当卸売業者に出庫を依頼するためにインターネットに接続された PC（パーソナルコンピュータ）を用いる．PC にはサーバプログラムに接続するためのクライアントプログラムがあらかじめインストールされている．小売店は受付係システムを利用するために，まず，小売店毎に割り当てられた ID を入力してログインする必要がある．ログインは同一の ID で複数行うことが許される．ログインに成功すると，サーバプログラムから扱い銘柄の一覧表が送信されるので，小



(a) 小売店の利用モデル　　(b) 搬入係の利用モデル　　(c) 出庫係の利用モデル

図 3: 受付係システムに対する各外部実体の利用モデル

売店は希望の銘柄および数量を入力する．入力後，注文内容と納品時期に関する情報がフィードバックされる．出庫依頼は繰り返し行うことができる．出庫依頼が完了すれば，小売店は受付係システムからログアウトする．

- 搬入係

  搬入係には携帯端末が支給される．携帯端末には，サーバプログラムに接続するためのクライアントプログラムがあらかじめインストールされている．搬入係は業務開始時，倉庫係毎に割り当てられる ID を用いて携帯端末から受付係システムにログインする．同一の ID で複数ログインすることは許されていない．酒造業者から発送されたコンテナが到着すると，倉庫に搬入する．その際，積荷票を必ず受け取って携帯端末に入力することになっている．積荷票のデータはネットワークを経由してサーバプログラムに送信される．サーバプログラムはデータを受信すると，送信元の携帯端末に対してフィードバックのメッセージを送る．搬入係は業務終了時に受付係システムからログアウトする．

- 出庫係

  出庫係には携帯端末が支給される．携帯端末には，サーバプログラムに接続するためのクライアントプログラムがあらかじめインストールされている．出庫係は業務開始時，倉庫係毎に割り当てられる ID を用いて携帯端末から受付係システムにログインする．同一の ID で複数ログインすることは許されていない．出庫係の仕事は，サーバプログラムから携帯端末に随時送信される出庫指示にしたがって，在庫品を梱包し，小売店に発送する準備をすることである．発送準備が完了すると，出庫係は携帯端末を用いて受付係システムにその旨通知し，次の出庫指示を待つ．出庫係は業務終了時に受付係システムからログアウトする．

以上に基づいて，各外部実体が受付係システムのクライアントプログラムに対して行う操作を，ステートマシン図により定式化した．

利用モデルを完成させるためには，さらに，ステートマシン図上の遷移確率を決定しなければならない．遷移確率は実際の運用状況を反映することが望ましいが，今回の実験では実際の運用を想定できないので，推測に基づいて決定することとした．完成した利用モデルを図 3 に示す．この中で特に重要なものは，(a) の小売店の利用モデルである．なぜなら，小売店の振る舞いは能動的であるのに対して，倉庫係の振る舞いは主に受動的であるからである．すなわち，倉庫係は比較的限定的な時間および人数で勤務を開始（ログイン）し，終了（ログアウト）するのが普通である．また，1 日あたりの出庫指示の件数や積荷票の入力頻度は，小売店の振る舞いに大きく影響される．以上の理由から，小売店の利用モデルは他よりも詳細にモデル化される必要があると判断できる．

Step 2. 入力条件の生成

Java 言語により，入力条件ジェネレータを備えたテストドライバを開発した．テストドライバは，クライアントプログラムに対して，入力条件ジェネレータが供給する入力条件に従った操作を自動的に行う．同じ乱数の種と利用モデルデータを与えることによって，入

力条件ジェネレータは，確率的でありながら全く同じ入力条件を生成できる．

Step 3. 期待出力とテスト出力の生成

受付係システムは，標準出力を用いてその重要な動作過程を表示できる仕組みになっている．そこで，期待出力およびテスト出力を得るために，受付係システムのすべての標準出力および標準エラー出力をテキストファイルに記録する機能をテストドライバに実装した．以上により，乱数の種と利用モデルデータ，および生成処理の終了条件を指定してテストドライバを実行すれば，期待出力とテスト出力は自動的に得られる．

なお，これらの生成過程においてバグを 1 件検出した．このバグはスレッド間共有データへのアクセスの同期に関するもので，`synchronized` 修飾子の欠落が原因であった．このことは本手法の本質的な有効性を示すものではないが，非決定的に振る舞うバグの検出に役立つことを確認できた．

Step 4. バグの検出

標準 Unix ツール diff，および diff の出力結果を処理するための Java プログラムによって，出力ベリファイヤを実現した．まず，期待出力とテスト出力を diff で処理し，差分を抽出する．サーバプログラムに対する処理結果の一部を図 4 に示す．行頭に”>”のある行は，テスト出力に含まれるが期待出力には含まれないことを意味している．逆に，”<”の行は，期待出力に含まれるがテスト出力には含まれない．diff による差分には，バグの疑いのあるものの他に，テストの実行時刻の相違に由来するもの（図中の出庫日時の行を参照）や，追加機能に由来するもの（図中のコンテナ搬出の行を参照）などが多く含まれる．さらに，サーバプログラムはマルチスレッドであるため，スレッドの実行タイミングの相違に起因する差分も含まれる（図中のコンテナ内蔵品割り当ての行を参照）．そこで，バグの有無やその原因を人手によって効率的に確認できるようにするために，Java プログラムを用いて diff の結果からバグに由来しない差分を取り除く．こうして得られた最終的な差分は，元の期待出力とテスト出力の合計サイズの 1%未満であった．なお，Java プログラムの代わりに，標準的な Unix ツールや簡便なスクリプト言語を利用することも可能である．

以上の方法によりバグの検出を試みたが，本実験でバグは検出されなかった．

ミューテーション解析

上記実験では，バグ以外のあらかじめ想定できる差分を無視できることは確認できたが，本手法のバグ発見能力を検証することができなかった．そこで，テストケースの品質判定のための既知の手法であるミューテーション解析 [5] を，本適用実験の被テストプログラムに対して行うことにした．ミューテーション解析では，まず，元のプログラム P にバグを混入した変異プログラム M_i $(i = 0, 1, \cdots, m)$ を生成する．M_i はミュータントと呼ばれる．そしてテストケース集合 T を P と M_i で実行し，結果を比較する．最終的にテストケースの品質指標として，以下に示すミューテーションスコア $S(P, T)$ が算出される．

```
---
> コンテナ内蔵品 00008 を出庫依頼 00008 に割り当て中...
> 割り当て前の内蔵量：256 -> 割り当て後の内蔵量：241
820,822d822
< コンテナ内蔵品 00008 を出庫依頼 00008 に割り当て中...
< 割り当て前の内蔵量：256 -> 割り当て後の内蔵量：241
<
839c839
< 出庫日時：Mon Jun 27 05:07:05 JST 2005
---
> 出庫日時：Mon Jun 27 05:19:05 JST 2005
845a846
> コンテナ 00005 を搬出
850a852,853
> コンテナ 00005 の搬出を完了しました.
>
852c855
< 出庫日時：Mon Jun 27 05:07:10 JST 2005
---
> 出庫日時：Mon Jun 27 05:19:10 JST 2005
```

図 4: diff による期待出力とテスト出力の差分の一部

$$S(P, T) = \frac{k}{m - e - c} \times 100 \tag{1}$$

ただし，k はテストケース集合 T によってバグを検出されたミュータント数，m はミュータントの総数，e は等価ミュータント（P と等価の M_i）の数，c はコンパイルエラーとなった M_i の数である．

実験では，ランダムに選択した `if` ステートメントの条件式に含まれる算術演算子を置き換えることによってミュータントを生成した．テストドライバの処理終了条件を「入力条件の適用開始から 3 分経過後」として統計的回帰テストを実行した結果，ミューテーションスコアは 90（$k = 9, m - e - c = 10$）であった．本研究は，この結果に関して妥当な比較の対象を持っていないため，客観的な有効性は明らかでないが，少なくとも，統計的回帰テスト法がバグを検出できることを確認した．なお，検出できなかった唯一のミュータントに含まれるバグは，稀な状況でしかその影響が顕在化しないため，短時間でのテストでは検出が困難であったと考えられる．

4. 考察

4.1 信頼性の推定精度

本手法の第 1 の目的はソフトウェアの信頼性を推定することである．統計的テストにおいて，ソフトウェアの信頼性はバグの分布と利用の分布の関係によって決まると考える．これは，ソフトウェア信頼性工学の分野でのデータ領域モデルに類するものである [12]．信頼性を正確に推定するためには，2 つの事項に留意しなければならない．

まず 1 つ目は，プログラムの出力に潜在するバグの結果や兆候を，可能な限り見逃さずに発見できる必要がある．もしバグを見逃すと，現実より高く信頼性を

推定することとなる．このようなことを避けるためには，期待出力の確度や出力ベリファイヤの検証機能を確かなものとしなければならない．

期待出力の確度 本手法は，基準プログラムから期待出力を生成するため，期待出力の確度に限界がある．例えば，基準プログラムのバグをそのまま被テストプログラムが継承している場合，そのバグは隠れる可能性が高い．ただし，基準プログラムに多数のユーザによる長期間の運用実績がある場合は，基準プログラムの信頼性（すなわち期待出力の確度）に対して確証を得ることができる．

出力ベリファイヤの検証機能 期待出力とテスト出力の差分が，バグによるものか，仕様の変更によるものか，あるいはテストの実行条件の差異によるものかを正確に判別できる必要がある．

2 つ目は，利用モデルが，実運用におけるユーザの利用の分布を正確に反映している必要があるということである．これには，利用現場のデータを用いる方法（2.1 節参照）が有効である．ただし，利用モデルの精度と信頼性の評価精度には相関があるが，利用モデルの精度が高いほど多くのバグが発見できるわけではない．

4.2 関連研究

回帰テストの観点で本研究に最も関連するのは，松下らの提案するプログラム差分を用いたデバッグ手法である [13]．この手法は，基準プログラムと最新バージョンのプログラムの間でテスト出力に差異が認められた場合，基準プログラム以降のどのバージョンでバグが作り込まれたかを，新しいバージョンから線形探索することによって特定する．そして，バグが作り込まれたバージョンとその直前のバグを含まないバージョンとのプログラム差分においてバグが存在すると考える．入力条件の作成方法は特に説明されていないが，期待出力は本研究と同様，基準プログラムから導出するので，本研究との親和性は高い．

5. おわりに

回帰テストにおいて，系統的テストとは別のアプローチとして統計的テスト法を導入することを提案した．本手法は，安定した基準プログラムに対してランダム生成した入力条件を適用することにより，厳密な期待出力を自動的に生成する．本手法の限界は，被テストプログラムが基準プログラムよりも高い信頼性を達成することが困難な点である．これは，期待出力が基準プログラムから生成されるためである．しかし，基準プログラムに多数のユーザによる長期間の運用実績がある場合は，基準プログラムの信頼性（すなわち期待出力の確度）に対して確証を得ることができる．さらに，統計的テストには，テストケースを確率的に際限なく生成できるという，従来の系統的なテストにはない特徴がある．それゆえ，従来方法だけでは見落とす可能性のあるバグが，本手法によって効果的に検出されることが期待できる．

今後の研究においては，出力ベリファイヤの検証機能を強化する方法の調査や適用実験を行う予定である．

参考文献

[1] Whittaker, J. A. and Poore, J. H.: Markov Analysis of Software Specifications, *ACM Transactions on Software Engineering and Methodology*, Vol. 2, No. 1, pp. 93–106 (1993).

[2] Whittaker, J. A. and Thomason, M. G.: A Markov Chain Model for Statistical Software Testing, *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, pp. 812–824 (1994).

[3] Walton, G. H., Poore, J. H. and Trammell, C. J.: Statistical Testing of Software Based on a Usage Model, *Software Practice and Experience*, Vol. 25, No. 1, pp. 97–108 (1995).

[4] 渡部隆一: マルコフ・チェーン, 共立出版 (1979).

[5] Beizer, B.: *Software Testing Techniques*, Van Nostrand Reinhold, 2nd edition (1990).

[6] Object Management Group: *OMG Unified Modeling Language Specification*, http://www.uml.org/.

[7] Takagi, T. and Furukawa, Z.: Constructing a Usage Model for Statistical Testing with Source Code Generation Methods, *Proceedings of 11th Asia-Pacific Software Engineering Conference*, pp. 448–454 (2004).

[8] Sayre, K. and Poore, J. H.: Partition testing with usage models, *Information and Software Technology*, Vol. 42, No. 12, pp. 845–850 (2000).

[9] Poore, J. H., Walton, G. H. and Whittaker, J. A.: A constraint-based approach to the representation of software usage models, *Information and Software Technology*, Vol. 42, No. 12, pp. 825–833 (2000).

[10] 二村良彦, 雨宮真人, 山崎利治, 淵一博: 新しいプログラミング・パラダイムによる共通問題の設計, 情報処理, Vol. 26, No. 5, pp. 458–459 (1985).

[11] Sun Microsystems Corporation: *Java 2 SDK Standard Edition*, http://java.sun.com/.

[12] 山田茂: ソフトウェア信頼性モデル, 日科技連出版社 (1994).

[13] 松下誠, 寺口正義, 井上克郎: プログラム差分を用いたデバッグ支援手法 DMET, 電子情報通信学会論文誌 D-I, No. 8, pp. 815–823 (2004).