情報処理学会論文誌　Vol. 51　No. 9　1794–1804 (Sep. 2010)

多重マルコフ連鎖に基づく精密化利用モデルの構築とその適用例

高木智彦[†1]　古川善吾[†1]

統計的テスト法においてテストケースを生成するために用いる利用モデルを，多重マルコフ連鎖に基づき，現実のユーザの振舞いをより反映するように構築する手法を提案する．統計的テスト法は欠陥を網羅的に検出することよりもソフトウェア信頼性を評価することに主眼を置いたソフトウェアテスト技法であり，その評価の正確さはユーザの振舞いのモデルである利用モデルに依存している．従来手法では単純マルコフ連鎖や斉時マルコフ連鎖などが利用モデルとして用いられる．その場合，ユーザの次の振舞いがそれまでの振舞いの内容に影響されるような状況を表現することができず，利用モデルの正確さを損なう場合があった．そこで本稿では，多重マルコフ連鎖に基づく精密化利用モデルの構築手法を提案する．本手法の手順やアルゴリズム，適用例などについて述べる．精密化利用モデルが従来のものよりも正確であり，ソフトウェア信頼性を正確に評価するうえで役立つこと，そして実際のソフトウェア開発に適用可能であることが分かった．また，本手法をより有効化するための課題について明らかにすることができた．

Detailed Usage Model Construction Based on High-order Markov Chains and Its Examples

Tomohiko Takagi[†1] and Zengo Furukawa[†1]

This paper shows the detailed usage model construction method based on high-order Markov chains to generate test cases in statistical testing. The statistical testing is used rather for estimating software reliability than for discovering failures comprehensively, and the accuracy of the estimation depends on a usage model that represents the behavior of users. So far usage models have been constructed as simple or time-homogeneous Markov chains. However, such usage models can't represent situations where previous behavior of users influences next behavior, and therefore it can't always realize its accuracy. In order to solve this problem, we propose the construction method of detailed usage models based on high-order Markov chains. This paper includes the procedure, algorithm, and examples of this method. It was found that the detailed usage models are more accurate than usual ones and enable accurate estimation of software reliability. This method is of practical use in software development. In addition, challenges for making this method more effective emerged.

1. はじめに

近年，コンピュータシステムの品質はソフトウェアに大きく依存する傾向があり，それだけソフトウェア信頼性の重要性が増している．ソフトウェア信頼性とは，実際の利用環境下でユーザがソフトウェアの欠陥に遭遇することなく利用できる可能性のことである．これを改善するための直接的な活動として，ソフトウェアを実行することによって残存する欠陥を出荷前に発見するソフトウェアテストがある．ソフトウェアテストを効果的に行うための技法は従来から数多く提案されており[1)–3)]，特に，ソフトウェアの機能やソースコードを特定の基準に基づいて網羅する系統的テスト法[2)]は，開発現場で広く用いられている．系統的テスト法の主要な目的は欠陥を満遍なく検出することである．しかしながら，ソフトウェアを完全網羅することは現実的には不可能であり，ある基準で網羅できたとしても一定のソフトウェア信頼性を達成したことの根拠にはならない．また，ソフトウェア信頼性は，欠陥の潜在するソフトウェアの機能に対するユーザの利用頻度によって決まるものであるが，系統的テスト法はこのような利用頻度を考慮しないので，ソフトウェア信頼性を求めることができない．たとえば，全分岐網羅基準を満足したことに基づいて，ソフトウェア信頼性の尺度の 1 つである MTTF（Mean Time To Failure）が一定時間以上であることを示すことはできないし，また，全分岐網羅によって欠陥を発見したとして，その欠陥がユーザの利用環境においてどのような頻度で顕在化するのか明らかにできない．ソフトウェア信頼性による評価が困難であるという系統的テスト法の欠点を補完するには，統計的テスト法（statistical testing）[4)–6)]が有効であることが知られている．

統計的テスト法は，欠陥を網羅的に検出することよりもソフトウェア信頼性を評価することに主眼を置いた技法である．テストケースは，ユーザの振舞いを表すマルコフ連鎖に基づいて確率的に生成される．このマルコフ連鎖を利用モデル（usage model）あるいは運用

†1 香川大学工学部
Faculty of Engineering, Kagawa University

プロファイル（operational profile）[7),8)] という．利用モデルが正確であるほど（すなわち，利用モデルが現実のユーザの振舞いに近いほど），ソフトウェア信頼性を正確に評価することができる．したがって，利用モデルの正確さは統計的テスト法の有効性を左右する重要なポイントである．しかしながら，従来研究はこの問題に対して十分な解決方法を示しているとはいえない．たとえば，利用モデルとして単純マルコフ連鎖（simple Markov chain）や斉時マルコフ連鎖（time-homogeneous Markov chain）を用いることが一般的である．単純マルコフ連鎖は，イベントの生起確率が現在状態のみに依存するモデルであり，一方の斉時マルコフ連鎖は，状態を考慮せずにイベントの生起確率のみで構成するモデルである．このような利用モデルは容易に作成できる反面，現実のユーザの振舞いを正確に反映することが困難となる場合がある．これは，モデルの単純さゆえに，ユーザの次の振舞いがそれまでの振舞いの内容に影響されるような状況を表現することができないためである．

そこで本稿は，多重マルコフ連鎖（high-order Markov chain）に基づいて利用モデルを精密化する手法（以下，本手法）を提案する．多重マルコフ連鎖は，イベントの生起確率が現在状態だけでなくそれ以前の状態遷移にも依存するモデルである[9)–11)]．そのような精密化利用モデルが従来のものよりも正確であり，ソフトウェア信頼性を正確に評価するうえで役立つことを示すのが本稿の主目的である．そのうえで，本手法が開発現場で適用可能であることを示すことを副目的とする．まず 2 章で従来の統計的テスト法の概要と問題点を整理する．次に 3 章で本手法の手順とアルゴリズムを，4 章で適用例を通して従来手法の限界や本手法の有効性などを述べる．そして 5 章でコスト対効果や精密化の程度について考察を行い，最後に 6 章で関連研究について述べる．

2. 従来の統計的テスト法

2.1 概　　要

統計的テスト法はクリーンルーム開発手法[12)] を構成する一技術として考案され，その後，他の一般的な開発手法においても採用されるようになった．ソフトウェア信頼性の評価が統計的テスト法の主要な目的である．結果的に，ソフトウェア信頼性に深刻な影響を与える欠陥（すなわち，使用頻度の高い機能に関連する欠陥）を洗い出すことができるので，この点に期待して採用することも多い．適用可能なテストレベルは，単体テストから受け入れテストに至るすべてであり，特に受け入れテストとしての効果が期待できる．テストケース生成に用いるモデルの特性上，メニュー駆動型ソフトウェアや外部環境の変化に応じて振舞いを変えるソフトウェア，他のシステムやコンポーネントとの相互作用を行うソフトウェアなどのテストに適している．

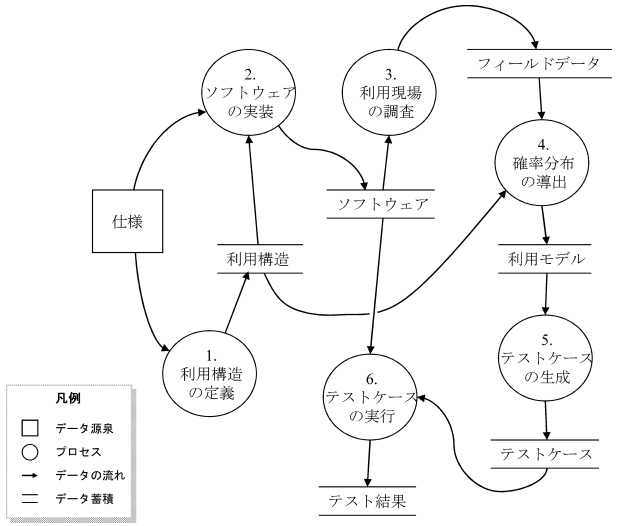


図 1　従来の統計的テスト法の概要（DFD による表現）
Fig. 1　Overview of traditional statistical testing (described in DFD).

従来の統計的テスト法の概要を**図 1** に示す．以下に各プロセスについて説明する．

プロセス 1.　利用構造（利用モデルの構造）として，ステートマシンあるいはイベントリストをソフトウェアの仕様に基づき定義する．

プロセス 2.　仕様に基づいてソフトウェアが実装される．利用構造も仕様の一部であり，ソフトウェアはこれを反映しなければならない．

プロセス 3.　ソフトウェアの利用現場を調査し，フィールドデータ（ソフトウェアの実行履歴や業務の生データなど，ソフトウェアの利用状況を把握できる情報）を収集する．

プロセス 4.　収集したフィールドデータを用いて，利用構造における確率分布を導出する．利用構造がステートマシンの場合は単純マルコフ連鎖としての，イベントリストの場合は斉時マルコフ連鎖としての利用モデルが完成する．

プロセス 5.　利用モデルをテストケース生成ツールとして任意の方法で実装する．そして，利用モデルに基づいて確率的に（すなわち，乱数発生器を用いて確率分布に従うように）テストケースを生成する．

プロセス 6.　生成したテストケースを実行する．必要に応じて，自動テストのためのテストドライバを用意する．最終的に，ソフトウェア信頼性をはじめとしたテスト結果を得る．

2.2 問　題　点

統計的テスト法を効果的に行うためには，利用モデルが正確でなければならない．ここで「正確である」とは，現実のユーザの振舞いを反映していることを意味する．正確な利用モデルを構築するには，(a) 信頼できるフィールドデータを収集すること，(b) 利用構造を精密化すること，の 2 点が必要である．

(a) については従来研究でいくつか提案があり，すでに解決済みである．ユーザビリティテストの分野ではビデオ撮影やアンケートなどによる評価方法が確立されており，これらを導入することができる．過去の統計的テスト法に関する研究においても，アンケート調査から航空機サブシステムの利用モデルを構築した事例がある[13]．また，テスト対象ソフトウェアのプロトタイプや前バージョン，類似する他のソフトウェアなどから自動的にデータ収集する方法は，環境が整えば効果的に実施できる．たとえば，Hartmann ら[14] は，テスト実行ツールの一種であるキャプチャ/プレイバックツールを用いて医療機器の操作履歴を収集し，利用モデルを作成した．また，Kallepalli ら[15] は，Web システムの利用モデルを構築するために，Web サーバのアクセスログを用いている．Shukla ら[16] は，Java API に探針（メソッドのコールシーケンスを記録するプログラム片）を挿入するという方法をとった．

一方，(b) は十分に検討されていない．まず，ステートマシンは状態が定義される分イベントリストより精密である．ステートマシンを精密化する方法については，再訪可能状態をその訪問回数で区別して再定義するというアイデアが示されている[6]．たとえば，ある再訪可能状態 s を，1 回目に訪問した状態 s_1 と 2 回目以降に訪問した状態 s_2 に区別する．これによって，s_1 と s_2 の確率分布が s の確率分布として平均化される，という問題を解決できるので，より正確な単純マルコフ連鎖としての利用モデルが得られる．しかしながら，この方法は手作業によって行うことを想定しており，体系的に行うことが困難である．作業を担当する技術者は，利用現場の調査結果や自身の経験，推測などに基づいて利用特性を認識できた箇所についてのみ s_1 と s_2 のように区別することになるので，得られる効果は属人的である．また，一定規模以上の利用構造を分析しようとする場合には，コストや人間の理解力による限界が生じる．

s_1 と s_2 は，自身を始点として生成できるイベントとアクションの列の集合がまったく同一という意味で等価であり，これらを等価状態（equivalent states）[3] という．ソフトウェアの仕様として s_1 と s_2 が同じものを表現していることは明らかである．ゆえに，ソフトウェアの仕様であるステートマシンにおいて等価状態は冗長であり，通常作り込んではならない．しかしながら，前述のように，ユーザの振舞いを利用モデルとして精密に定義するうえでは有効である．

ここまでの内容をまとめると，正確な利用モデルを構築するためには (b) についてさらに検討する必要があるといえる．(b) に関する従来研究では，イベントリストではなくステートマシンを定義し，単純マルコフ連鎖としての利用モデルを構築することが唯一の現実的な解である．しかしながらこれでは，ユーザの次の振舞いがそれまでの振舞いの内容に影響されるような状況を表現できないという問題がある．等価状態を手作業で作成するアイデアは，この問題を解決する方向を示してはいるものの，作業の品質やコストなどの問題により現実的な解とはなりえない．

3. 多重マルコフ連鎖に基づく精密化利用モデル

3.1 手　　順

2.2 節で述べた問題点を解決するために，多重マルコフ連鎖に基づく精密化利用モデルの構築手法を提案する．多重マルコフ連鎖は，イベントの生起確率が現在状態だけでなくそれ以前の状態遷移にも依存するモデルである．これによって，ユーザの次の振舞いがそれまでの振舞いの内容に影響されるような状況を表現できないという問題を解決できる．本手法はツールによる自動化が可能であるため，手作業に起因する問題は生じない．また，精密化（すなわち等価状態の作成）の対象を再訪可能状態に限定しないことや，マルコフ連鎖を前提に構築された従来の統計的テスト法の理論やツールとの親和性が高いことなどが特長としてあげられる．

本手法を組み込んだ統計的テスト法の概要を**図 2** に示す．図 2 では，図 1 から変更のある部分を強調表示した．プロセス 1，3，4 を改良してプロセス 1’，3’，4’ とし，新たにプロセス 1” を追加している．以下では，本手法を構成するそれらのプロセスについて，**図 3** に示す例題を用いながら説明する．

プロセス 1’.　従来の統計的テスト法と同様に，利用構造としてステートマシン[17] を定義する．本手法は利用構造としてイベントリストを用いない．本プロセスで作成する利用構造を，プロセス 1” で作成するものと区別するために，標準利用構造と呼ぶことにする．標準利用構造では，「ユーザが特定の操作を行う」というイベントを遷移のラベルとして，また，ユーザの振舞いの段階を状態として表す．標準利用構造は等価状態を含

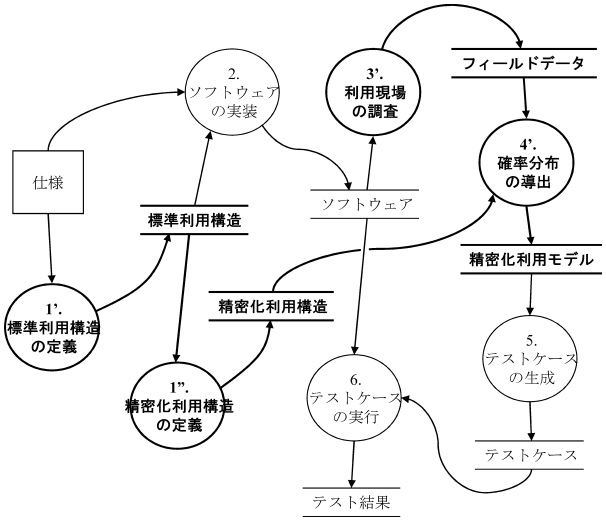


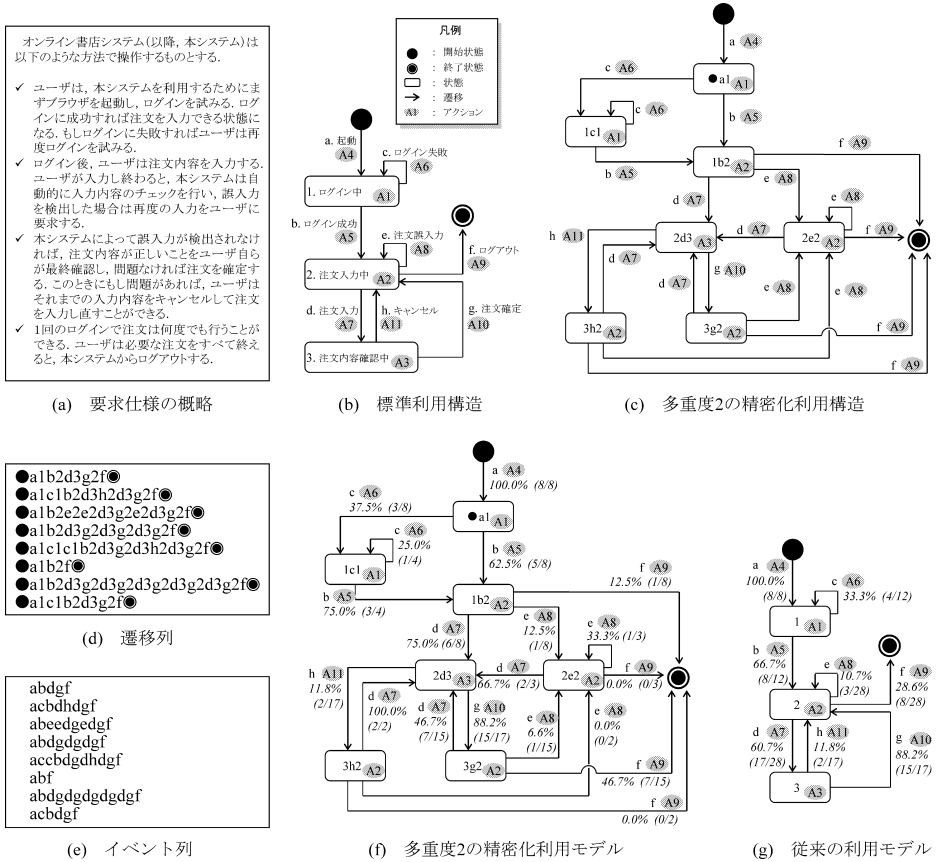
図 2　本手法を組み込んだ統計的テスト法の概要（DFD による表現）
Fig. 2　Overview of statistical testing this method is incorporated into (described in DFD).

んではならない．状態や遷移はアクションを持つことができるので，必要に応じて，テストデータを入力したりテスト出力を検証する方法をアクションとして定義する．

標準利用構造の例を図 3 (b) に示す．これは，筆者らが独自に想定した図 3 (a) の要求仕様に基づいている．状態や遷移はそれぞれ固有のアクションを持つが，各アクションの具体的内容は本稿では割愛する．また，以降では簡単のため，状態やイベントをそれぞれ番号やアルファベットによって参照することとする．

プロセス 1"． m 重マルコフ連鎖のためのステートマシンを標準利用構造に基づいて構築する．本研究ではこのステートマシンを精密化利用構造と呼ぶ．精密化利用構造の特徴は，各状態が長さ $(m-1)$ の状態遷移の履歴を持っている点である．より詳しくいえば，精密化利用構造の状態は，標準利用構造における長さ $(m-1)$ の遷移列，および開始状態を始点とする長さ $(m-2)$ 以下の遷移列によって特徴付けられる．m を多重度という．多重度 1 の精密化利用構造は標準利用構造と同じであるため，多重度は 2 以上でなければならない．多重度 m の決定方法は 5.2 節で，また，精密化利用構造を構築するための具体的なアルゴリズムは 3.2 節で示すことにする．

図 3 (c) は，図 3 (b) に基づいて多重度 2 で構築した精密化利用構造である．図 3 (c) の



(a)　要求仕様の概略　(b)　標準利用構造　(c)　多重度2の精密化利用構造

(d)　遷移列

(e)　イベント列　(f)　多重度2の精密化利用モデル　(g)　従来の利用モデル

図 3　オンライン書店システムの例題
Fig. 3　Example of an online book store system.

各状態は図 3 (b) における状態遷移を 1 つ記憶している．たとえば状態「1b2」は，状態「1」においてイベント「b」が発生した結果，現在は状態「2」であることを表している．アクション A2 を持つのは，この状態が状態「2」に対応しているからである．状態「2」に対応するものとしては状態「1b2」のほかに「2e2」「3g2」「3h2」があり，これら 4 つの状態は等価状態である．図 3 (c) の状態の表記法では，末尾の数字が同じも

のが等価状態となる．

プロセス 3'.　フィールドデータを遷移列またはイベント列の形式で収集する．これには 2.2 節で示した収集方法を用いる．遷移列とイベント列は，標準利用構造における開始状態から終了状態に至る経路を表すものである．どのような多重度の精密化利用構造に対しても直接対応付けできる汎用的な形式であるため，プロセス 4' の自動化が可能になる．遷移列とイベント列のどちらを使うべきかは状況により判断する．すなわち，利用構造とフィールドデータの整合性を確認する必要があれば前者を，整合性が確認済みであり，フィールドデータのサイズを小さくする必要があれば後者を選択する．

遷移列の例を図 3 (d) に，イベント列の例を図 3 (e) に示す．これらのフィールドデータとしての意味は同一である．

プロセス 4'.　収集したフィールドデータで精密化利用構造をトレースし，確率分布を導出する．遷移確率は，遷移元の状態の訪問回数に対する，当該遷移の実行回数の割合である．精密化利用構造の各状態が状態遷移の履歴を持つため，現在状態だけでなく状態遷移の履歴をも加味してイベントの生起確率を決定できる．以上により，m 重マルコフ連鎖としての精密化利用モデルが完成する．精密化利用モデルはプロセス 5 でテストケースを生成するために用いられる．テストケースは，開始状態から始まり終了状態で終わる，アクションをともなう状態遷移列である．

図 3 (f) に精密化利用モデルの例を示す．これは，図 3 (d) の遷移列（または図 3 (e) のイベント列）を用いて図 3 (c) の精密化利用構造をトレースした結果である．

3.2　精密化利用構造の構築アルゴリズム

本研究では，グラフ探索アルゴリズムとして一般的に知られている深さ優先探索を応用し，精密化利用構造を構築するアルゴリズムを検討した．本アルゴリズムは以下に示す 5 つのステップから構成される．実行に必要なものは，標準利用構造と多重度 m の 2 つである．

ステップ 1.　標準利用構造の開始状態から探索を開始する．なお，以降ではすべての状態遷移の実行を記録し，開始状態から現在状態に至るまでの経路をつねに把握できるようにする．

ステップ 2.　標準利用構造の現在状態において生起可能なイベントを選び，そのイベントの生起にともなう遷移を実行したと仮定する．

もし，（仮定上の）現在状態に至る直近の長さ $(m-1)$ の遷移列が未実行であれば，精密化利用構造の状態と遷移を新たに識別したことを意味する．この場合，まず標準利用構造において仮定した遷移を実際に実行する．そして，その長さ $(m-1)$ の遷移列によって特徴付けられる精密化利用構造の状態を追加し，さらに，先ほど実行した遷移に対応する精密化利用構造上の遷移を追加する．**図 4** (a) を前提として，以上の処理を行った例を図 4 (b) に示す．

一方，もし，（仮定上の）現在状態に至る直近の長さ $(m-1)$ の遷移列がすでに実行済みであれば，それは精密化利用構造の遷移を新たに識別したことを意味する．この場合，仮定した遷移に対応する精密化利用構造上の遷移を追加する．仮定した遷移は実行しない．図 4 (a) を前提として，これらの処理を行った例を図 4 (c) に示す．

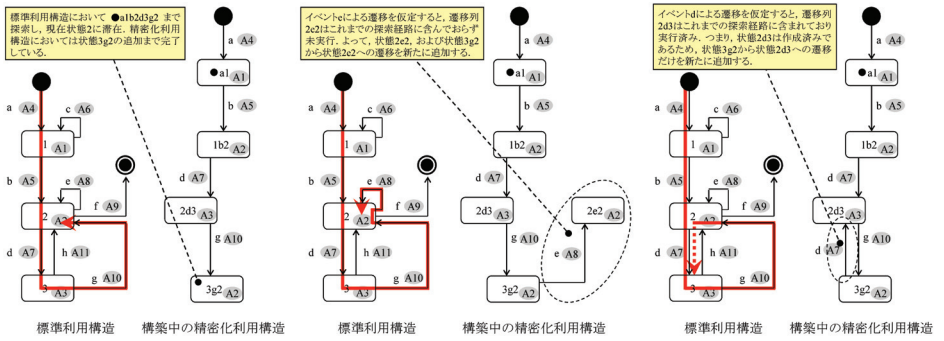


図 4　図 3 (c) における精密化利用構造構築アルゴリズムのステップ 2 の例
Fig. 4　Example of step 2 of the detailed usage structure construction algorithm on Fig. 3 (c).

ステップ 3.　標準利用構造の現在状態において，生起可能なイベントがなくなるか，未実行の長さ $(m-1)$ の遷移列が新たに見つからなくなるまでステップ 2 を繰り返す．

ステップ 4.　生起可能なイベントがあり未仮定の遷移を持つような状態（すなわち，未実行の長さ $(m-1)$ の遷移列が新たに見つかる可能性のある状態）が標準利用構造に残っていれば，その状態まで経路をさかのぼる．

ステップ 5.　以降同様に，すべての長さ $(m-1)$ の遷移列を実行し尽くすまでステップ 2〜5 を繰り返す．

4.　適　用　例

本章では 2 つの適用例を示す．1 つ目は，図 3 の例題への適用を通して，利用モデルの正確さとソフトウェア信頼性の評価精度の観点から従来手法の限界や本手法の有効性を明

らかにすることに主眼がある．2つ目は，商用ソフトウェアへの適用実験の結果についてであり，本手法が実際のソフトウェア開発に適用可能なことを確認するためのものである．なお，前者は 1 章で述べた本稿の主目的に，後者は副目的に対応している．

4.1　オンライン書店システム

図 3 のオンライン書店システムの例題を用いて，利用モデルの正確さとソフトウェア信頼性の評価精度の観点から従来手法の限界や本手法の有効性を示す．

図 3 (g) は，図 3 (b) の標準利用構造，および図 3 (d) に示す 8 本の遷移列（または図 3 (e) に示す 8 本のイベント列）で構成されるフィールドデータから作成した従来の利用モデルである．従来の利用モデル構築方法ではフィールドデータを標準利用構造に直接対応付けるため，結果として得られる利用モデルは単純マルコフ連鎖であった．一方，本手法を用いて構築した多重度 2 の精密化利用モデルが図 3 (f) である．図 3 (f) は，図 3 (g) と同じフィールドデータから構築したにもかかわらず，図 3 (g) より正確である．たとえば，図 3 (f) における等価状態「1b2」「2e2」「3g2」「3h2」，およびそれらの終了状態への遷移を参照されたい．これらは，図 3 (g) における状態「2」，および状態「2」から終了状態への遷移に対応している．等価状態「1b2」「2e2」「3g2」「3h2」から終了状態への遷移回数の合計（すなわち，状態「2」から終了状態への遷移回数）は 8 回であり，その内訳は「1b2」が 1 回，「3g2」が 7 回である．図 3 (g) においては等価状態が存在せず，それゆえ 8 回の遷移を前述のように区別できない．結果として，「1b2」「2e2」「3g2」「3h2」から終了状態への遷移確率について，従来の利用モデルの場合はすべて 28.6%になるが，多重度 2 の精密化利用モデルの場合はそれぞれ 12.5%，0.0%，46.7%，0.0%となる．「ユーザが注文確定直後にログアウトする傾向にある」という利用特性が，2 重マルコフ連鎖を用いることで明らかになった．

ここで，利用モデルの正確さを定量的に評価するための尺度として，利用分布網羅率[18]を導入する．従来研究においては，利用モデルの特性を表す尺度としてエントロピー[4]が用いられてきた．エントロピーは情報源としてのマルコフ連鎖の不確実さを表す尺度であり，この値が小さいほど利用モデル上の確率分布に偏りがあると見なすことができる．しかしながら，確率分布の偏りが真にユーザの利用特性によるものか否かを評価できないという問題がある．これを解決するのが利用分布網羅率である．利用分布網羅率は，想定されるテスト対象ソフトウェアの使われ方をテストケースがどれだけ網羅しているかを表す尺度として提案された．利用モデルに基づいて計算する点ではエントロピーと同様である．具体的には，個々のテストケースについて遷移確率の総積を求めておき，全テストケース（ただし重複するテストケースは除外する）においてそれらの総和をとるという方法で算出する．

表 1　オンライン書店システムの利用モデルの概要
Table 1　Summary of usage models for an online book store system.

多重度	構築時間 ⋆1 (ミリ秒)	状態数	遷移数	状態網羅率 ⋆2	遷移網羅率 ⋆2	エントロピー ⋆3	利用分布網羅率 ⋆4
1 ⋆5	0	5	8	1.000	1.000	N/A	0.367
2	31	9	19	1.000	0.842	0.689	0.408
3	62	20	39	0.850	0.590	0.611	0.422
4	94	40	82	0.600	0.378	0.500	0.512
5	297	83	166	0.386	0.217	0.442	0.637
6	453	167	337	0.222	0.113	0.356	0.906
7	906	338	677	0.115	0.058	0.356	0.906
8	3,484	678	1,360	0.059	0.030	0.356	0.906
9	26,813	1,361	2,724	0.031	0.015	0.356	0.906
10	187,047	2,725	5,455	0.016	0.008	0.329	1.000

⋆1　実行環境は，3 GHz の CPU と 760 MByte の RAM を搭載したパーソナルコンピュータである．
⋆2　状態/遷移網羅率は，フィールドデータが精密化利用構造の状態/遷移をどれだけ網羅したかを表す．
⋆3　エントロピーは，利用モデルにおける確率分布にどれだけ偏りがないかを表す．
⋆4　利用分布網羅率は，利用モデルがフィールドデータと同様のテストケースを生成できる可能性を表す．
⋆5　多重度 1 は，単純マルコフ連鎖であることを意味する．

ただし，本研究ではテストケースではなくフィールドデータについての利用分布網羅率を求める．利用分布網羅率が 0.0 のときは当該利用モデルがフィールドデータと同じテストケースを生成できる可能性がないことを，1.0 のときはフィールドデータと同じテストケースしか生成できないことを意味する．つまり，フィールドデータの利用分布網羅率が高いほど，当該利用モデルはフィールドデータに近い（すなわちユーザの利用特性を反映した）テストケースを生成できるということであり，正確であるといえる．

さらに本研究では，図 2 のプロセス 1”，4’，5 に相当する作業を自動化するツール HOUMA（High-Order Usage Model Analyzer）を開発した．これを用いて，図 3 (b) の標準利用構造と図 3 (e) のイベント列から多重度 1〜10 の利用モデルを自動構築したので，その結果を**表 1** にまとめる．多重度 1 は単純マルコフ連鎖であることを意味する．多重度が高くなるにつれてエントロピーは小さく，利用分布網羅率は大きくなっている．よって，精密化利用モデルは従来の利用モデルよりも正確であり，また多重度が高いほど正確になることが明らかである．

正確な利用モデルを構築する目的は，正確にソフトウェア信頼性を評価できるようにするためであった．そこで次に，ソフトウェア信頼性の評価精度に与える影響について示す．以下の 3 つのケースを仮定し，「1 回の利用（1 つのテストケース）において欠陥が顕在化する

回数の期待値」としてのソフトウェア信頼性[5] について図 3 (f) と図 3 (g) の間で比較する．

ケース 1.　状態「2」から終了状態への遷移で欠陥が顕在化する．

ケース 2.　状態「3g2」から終了状態への遷移で欠陥が顕在化する．

ケース 3.　状態「1b2」から終了状態への遷移で欠陥が顕在化する．

ケース 1 について計算すると，図 3 (f)，図 3 (g) ともに 1.00 となり，本手法を適用しなくても正確にソフトウェア信頼性を評価できる．これは，図 3 (f) において等価状態「1b2」「2e2」「3g2」「3h2」から終了状態への 4 つの遷移すべてで欠陥が顕在化するため，利用モデルを精密化した効果が結果的に得られなかったためである．他方，ケース 2 について計算すると，図 3 (f) では 0.88，図 3 (g) では 0.54 となる．最大でも 1.00 までの値しかとりえないことを考えると，これらの間には無視できない差がある．図 3 (g) では，現実よりも欠陥顕在化の頻度が低くなるので，その分，当該欠陥のソフトウェア信頼性への影響を過小評価してしまう．またケース 3 について計算すると，図 3 (f) では 0.13，図 3 (g) では 0.28 となる．これはケース 2 とは逆で，図 3 (g) を使用すると当該欠陥のソフトウェア信頼性への影響を過大評価してしまう．このように，ケース 2 とケース 3 では多重度 2 の精密化利用モデルを使用することによって従来よりも正確にソフトウェア信頼性を評価できるようになる．ただし，ケース 1 からも分かるように，ケース 2 やケース 3 において多重度を 3 以上にしてもこれ以上評価精度は向上しない．一般に，多重度 m の精密化利用モデルを用いると，長さ m 以下の状態遷移列で顕在化する欠陥（欠陥顕在化のトリガとなる遷移を実行してから実際に欠陥が顕在化するまでの状態遷移数が $(m-1)$ 以下であるような欠陥）について正確な評価ができる．

4.2　はがき作成ソフトウェア

あるソフトウェア開発企業の品質管理部では，ソフトウェアの品質をより高めるための二次的な手法について検討している．そこで筆者らは品質管理部の協力の下，商用ソフトウェアであるはがき作成ソフトウェアのサブシステムに対して，本手法を実験的に適用した．このサブシステムは約 13 KLOC の標準的な規模であり，品質管理部が開発部から受け入れた当時の古い版（受け入れ版），および，品質管理部におけるテストをほぼ完了した版（出荷候補版）が存在した．本適用実験では，(a) シナリオテスト，組合せテスト，状態遷移テストなどの系統的テスト法によって受け入れ版をテストする場合，(b) 多重度 2 の精密化利用モデルに基づいて受け入れ版をテストする場合，(c) 多重度 2 の精密化利用モデルに基づいて出荷候補版をテストする場合，の 3 パターンを試行することで，本手法の開発現場における適用可能性を確認した．なお，ここでいう適用可能性とは以下の 3 点を意味する．

- 本手法のソフトウェア信頼性が，ソフトウェア開発における品質管理や進捗管理のためのメトリクスとして有効であること．
- 時間的にも人員的にも制約があるテスト工程において，本手法に要する労力が許容できる範囲であること．
- 欠陥を検出できること．欠陥を検出できなければ，効果的にソフトウェア信頼性を評価し，作業担当者のモチベーションを向上させることが困難である[⋆1]．

(a) は，受け入れ版から出荷候補版を得るために品質管理部がかつて実施した作業に相当するため，作業履歴に基づいてテスト結果や要した労力などを明らかにした．一方，(b), (c) については以下のような作業を筆者らが行った．まず，筆者が当該サブシステムの仕様書を参照しつつ標準利用構造を作成した．標準利用構造の規模は，状態数 31，遷移数 155 であり，作成には 2 人日相当の労力が必要であった．次に，筆者が試験運用を行う様子をビデオ撮影しイベント列に変換した．これには 0.5 人日を要した．以上のものを HOUMA への入力として多重度 2 の精密化利用モデルを構築し，1 人日の作業量に相当する 50 個のテストケースを自動生成した．精密化利用モデルの構築時間は約 520 ミリ秒，テストケースの生成時間は約 30 ミリ秒であった．テストケースの状態/遷移網羅率は，それぞれ 72.4%と 23.5%（標準利用構造上では 87.1%と 72.3%）であった．最終的に，品質管理部の技術者が (b), (c) それぞれにおいてこのテストケースを実行し，テスト結果をまとめた．

すべてのパターンのテスト結果を**表 2** に示す．(a) は，品質管理部が受け入れ版から出荷候補版を完成させるまでに 7 つの欠陥を検出したことを示している．系統的テスト法を用いているのでソフトウェア信頼性は明らかにできていない．これに対して，(b) では 5 つの欠陥を検出したうえで，ソフトウェア信頼性（MTTF）が 432 秒であることを明らかにしている．試験運用において 1 回の平均利用時間が 473 秒であることが分かっており，1 回の利用で欠陥が平均 1 回以上顕在化することになるので，出荷レベルには至っていない．欠陥検出効率は (a) より 8%劣る．要した労力は 3.5 人日で，(a) より 1 人日少ない．(c) は出荷候補版に対する適用であったため，欠陥の検出は 1 つだけであったが，これは (a) でも検出されなかったものである．なお，この欠陥が (b) で検出されなかったのは，顕在化の仕方に再現性がなかったためである．(b) と比較すると，ソフトウェア信頼性が約 260 分改善しており，品質管理部の活動（すなわち (a)）の成果を定量的に示している．また，ここで明らか

⋆1 すでに述べたように，統計的テスト法はソフトウェア信頼性を評価するための技法である．ゆえに本研究は，従来の統計的テスト法や系統的テスト法よりも欠陥を検出できるようにすることは目指していない点に注意されたい．

表 2　はがき作成ソフトウェアのサブシステムのテスト結果
Table 2　Result of testing the subsystem of postcard writing software.

	テスト対象	テストケース設計方法	労力（人日）	欠陥検出効率 ⋆1	MTTF⋆2	検出した欠陥 ⋆3
(a)	受け入れ版	各種の系統的機能テスト法に基づく設計	4.5	1.56 (7/4.5)	N/A	(1)(2)(3)(4)(5)(6)(7)
(b)	受け入れ版	多重度 2 の精密化利用モデルからの生成	3.5	1.43 (5/3.5)	24.4 遷移 432 秒	(1)(2)(4)(5)(7)
(c)	出荷候補版	多重度 2 の精密化利用モデルからの生成	3.5	0.29 (1/3.5)	902.0 遷移 15,999 秒	(8)

⋆1　欠陥検出効率は，人日あたりの欠陥検出数を意味する．下段に，欠陥検出数と人日を（欠陥検出数/人日）の形式で記載した．
⋆2　MTTF（Mean Time To Failure）は，最も使用されるソフトウェア信頼性の尺度の 1 つである．ここでは，遷移回数を単位とするもの[5] を上段に，実時間（秒）を単位とするものを下段に示した．後者は試験運用における 1 遷移あたりの時間を基に算出している．
⋆3　検出した個々の欠陥を括弧つきの数字 (1)〜(8) によって識別する．テストにおいては，同じ欠陥が複数回顕在化することがある．

になったソフトウェア信頼性は，出荷候補版が真に出荷レベルに達したかを判定する出荷判定に用いられる．これまでの経験やユーザの要望などに鑑みてソフトウェア信頼性が 14400 秒（4 時間）以上でなければならなかったが，出荷候補版はこれを満たすことが分かった．

以上に基づいて本手法の適用可能性を評価する．まず，ソフトウェア信頼性を評価し，品質管理部の活動成果や出荷判定のメトリクスとして活用できた．労力については，従来の系統的テスト法と同等以下であり，二次的なテスト法として許容できるものであった．また，欠陥検出効率は系統的テスト法と比較して若干劣る場合があるものの，出荷候補版においても欠陥を検出することができた．よって本手法は開発現場への適用が可能であり，効果が期待できると考えられる．

5. 考　　察

本章では，コスト対効果や多重度について考察を行う．

5.1 コスト対効果

本手法の比較対象としては，系統的テスト法と従来の統計的テスト法（従来の利用モデル構築方法）の 2 つが考えられる．

まずは系統的テスト法との比較を行う．系統的テスト法の弱点は，ソフトウェア信頼性の評価が困難であるということであった．4 章で本手法によるソフトウェア信頼性の評価が可能であることを示したので，本手法は系統的テスト法を補完する能力を有しているといえる．欠陥検出能力については一般的に系統的テスト法が優れているといわれており[2]，本研究も欠陥検出能力の改善を目的とはしていない．4.2 節の適用実験でも，本手法による統計的テスト法の欠陥検出能力は若干劣る結果であった．これは，系統的テスト法ではソフトウェア全体を満遍なくテストするのに対して，本手法では使用頻度の高い機能にテストが集中する傾向があるためである．ただし，今回の欠陥検出の結果は，当初の技術者の期待を上回るものであった．「使用頻度の高い機能は十分にテストされているはずである」という想定は一般的であり，統計的テスト法による欠陥検出が困難であることの理由としてしばしば指摘される．しかしながら，この想定がつねに正しいという保証はなく，この想定が正しくないときに統計的テスト法は当初の期待を上回る欠陥検出能力を発揮する可能性がある．実施に必要な労力については系統的テスト法に比較して同等以下であり，導入するうえで大きな障害にはならないと考えられる．

次に，従来の統計的テスト法（従来の利用モデル構築方法）との比較を行う．統計的テスト法においてソフトウェア信頼性を正確に評価できるようにするために，正確な利用モデルを構築することが本研究の目的であった．4.1 節では利用モデルの「正確さ」の尺度として利用分布網羅率を導入し，本研究が提案する精密化利用モデルが，従来の単純マルコフ連鎖としての利用モデルよりも正確であることを示した．さらに，精密化利用モデルを用いることで従来よりも正確にソフトウェア信頼性を評価できることを具体例を通して示した．精密化のために，精密化利用構造の構築という新たなプロセスが必要になったが，これは 3.2 節のアルゴリズムによって自動化される．したがって，新たな労力は必要とされない．また 2.2 節では従来の手作業による精密化方法を示したが，精密化の対象を再訪可能状態に限定しない点，等価状態や関連する遷移を自動生成できる点などで本手法は優れている．ゆえに本手法は従来の精密化方法よりも少ない労力でより精密な利用モデルを構築できる．

総括すると，本手法は従来の利用モデル構築方法より優れている．また，本手法による統計的テスト法は，系統的テスト法を補完する二次的なテストとして導入できる．

5.2 多重度の決定方法

精密化利用モデルの多重度をいくつにするかは重要な問題である．4.1 節では，多重度が高いほど正確な精密化利用モデルを構築できることを示した．しかしながら多重度が高すぎると，究極的にはフィールドデータと同じテストケースしか生成できなくなる．4.1 節にお

ける多重度 10 がその例である．一般的に，フィールドデータの利用特性を反映しつつ多様なテストケースを生成できる点に利用モデルの価値がある．

精密化利用モデルは，その元となる標準利用構造やフィールドデータの内容によって様々であり，また，ソフトウェアによって欠陥の特徴も異なる．よって，あらゆるソフトウェアについて最適な多重度というものは存在しない．個々の状況に応じた適切な多重度を求めるには，次の 3 つの方法を併用する．

欠陥に関する統計データに基づく方法　4.1 節で述べたとおり，多重度 m の精密化利用モデルを用いると，長さ m 以下の状態遷移列で顕在化する欠陥についてソフトウェア信頼性の正確な評価ができる．そこで，テスト対象ソフトウェアにおいて欠陥が顕在化するであろう状態遷移列の長さの分布を過去の統計データから推定し，ソフトウェア信頼性を正確に評価するための適切な多重度を明らかにする．この分布についての研究報告は現在のところ存在しないが，開発組織や分野などによって異なると考えられる．4.2 節の適用実験では，すべてが長さ 2 以上であった（ただし，欠陥の顕在化の仕方に再現性のないものを除く）．このような欠陥に関する統計データを蓄積しておく必要がある．

利用分布網羅率に基づく方法　4.1 節で述べたとおり，フィールドデータの利用分布網羅率は，当該利用モデルがフィールドデータと同様のテストケースを生成できる可能性を表している．たとえば利用分布網羅率が 0.5 のときでは，確率上，当該利用モデルが生成するテストケースの 2 つに 1 つはフィールドデータとまったく同じになる．そこで，テストの現場で要求されるソフトウェア信頼性の精度やテストケースの多様性に基づいて，妥当な利用分布網羅率となるように多重度を設定する．テストケースの多様性を重視すべきという現場の判断があれば，多重度をあえて低めに設定することもありうる．

生成したテストケースを目視する方法　上記の 2 つの方法で求めた多重度の妥当性を確認するために，精密化利用モデルからテストケースを仮に生成し技術者が目視する．そして，ユーザの利用シナリオとして不自然なものが多数混入している場合は多重度を高く設定し直し，また，同様の状態遷移列が高い頻度で出現する場合は多重度を低く設定し直すことを検討する．

6. 関 連 研 究

本研究と関連のある研究についてまとめる．

ソフトウェア信頼性の尺度としては，信頼度，MTTF，Rocof，アベイラビリティなど[19]が利用可能であり，特に統計的テスト法では，遷移回数を単位とした MTTF，1 利用あたりの欠陥顕在化回数，Kullback 判別式による利用モデルとテストモデルの同等性など[5),20)]が提案されている．しかしながら，高信頼性が要求される特別な分野以外で，これらが品質評価の基準として定められ用いられることはほとんどないのが実状である．一因は，正確な利用モデルを作成することの困難さにあると考えられ，その解決法の 1 つを示した点に本研究の意義があるといえる．

統計的テスト法によってソフトウェア信頼性を正確に評価するためには，正確な利用モデルを構築する必要がある．そのため，Whittaker ら[4]はフィールドデータから確率分布を導出するのが最善であると述べている．本研究はこの考えに基づき，フィールドデータとして遷移列またはイベント列を用いる方法を採用した．正確な利用モデルを構築するための従来研究に関しては，2.2 節を参照されたい．

ソフトウェアには，複数のユーザが相互に関連しながら並行に利用すること（すなわち，並行に動作する利用モデルが複数相互に関連すること）がある．この場合，個々の利用モデルに対して本手法を適用するか，それとも，すべての利用モデルを 1 つのプロダクトマシン（product machine）[3]に合成したうえで本手法を適用するか，選択しなければならない．もし，利用特性が利用モデル間で独立でないなら，前者よりも後者の方が正確である．しかしながら，文献 21) が指摘しているようにプロダクトマシンには状態爆発の可能性があるため，利用モデルが大きい場合は適用が困難となる可能性もある．並行性を特に扱う必要があれば，標準利用構造としてステートマシンではなくペトリネット[22]を用いることも考えられる．確率を導入したペトリネットは確率ペトリネットとして知られており，利用モデルの記述方法に用いることができる．この場合，ペトリネットを精密化利用構造に相当するモデルに拡張するための手法が必要になる．

利用モデルを応用した技術として近年注目を集めているのが，アーキテクチャに基づくソフトウェア信頼性分析[23]である．これは，ソフトウェア信頼性はソフトウェアを構成するコンポーネントの信頼性および利用確率によって推定できるという考えに基づいている．アーキテクチャを表すフローグラフあるいはステートマシンにコンポーネントの信頼性情報と利用確率を付加したモデルを用いる．このモデルは統計的テスト法におけるテストモデルに近いものであり，これに本手法を適用することでソフトウェア信頼性の推定精度の向上が期待できる．

本研究では HOUMA を開発したが，統計的テスト支援ツールはすでにいくつか存在している．よく知られたものとして，たとえば Prowell[21),24),25)]が作成している JUMBL（J

Usage Model Builder Library）がある．JUMBL は，単純マルコフ連鎖としての利用モデルを定義するために TML（The Modeling Language）を用いる．確率分布の導出方法に関して複数のオプションが用意されており，アクションをプログラミング言語で記述しておくと最終的にテストドライバを生成できる．さらに，TGL（Test Generation Language）を用いることによって，複数の利用モデルからのテストケース生成を総合的に管理できる．JUMBL に対する HOUMA の優位性は，4 章で示したように，多重マルコフ連鎖による精密化利用モデルの構築機能や利用分布網羅率の計測機能を有している点である．

より広い視点では，本手法による統計的テスト法は，ステートマシンを用いたモデルベースドテスト（model-based testing）の一種と位置づけることができる．この場合の従来研究としては，まず N スイッチテスト[26] があげられる．これは，ステートマシンにおける長さ $(N+1)$ の遷移列を網羅するようにテストケースを設計する方法である．多重度 m の精密化利用構造の状態空間は，標準利用構造において $(m-2)$ スイッチ基準を満たす遷移列集合によって構成されるため，この精密化利用構造の全状態を網羅すれば $(m-2)$ スイッチ基準を満たすことができる．Offutt ら[27] はステートマシンのテスト網羅基準として，遷移網羅レベル，遷移ペア網羅レベル，完全述語網羅レベル，全列網羅レベルを提案している．全列網羅レベルについては，現実のほとんどのソフトウェアで無限に列（テストケース）が存在するため，テスト技術者が重要と考えるものだけを網羅する．利用確率の高い列（あるいは低い列）が重要であるとする場合には，本手法の精密化利用モデルを活用するとよい．また，ステートマシンの拡張であるプッシュダウンオートマトン（PDA）を応用したテスト技法もある[28]．このテスト技法では，多くのソフトウェアが備えている Undo 機能についてモデル化しテストケースを生成するために，独自の PDA を導入している．履歴を保持するという意味では本手法の精密化利用構造と類似しているが，履歴として状態だけを扱う点，1 つの PDA で 1 つの記憶（スタック）しか持たない点などが異なっている．なお，文献 26)–28) のいずれも，本稿でいう系統的テスト法に属するテスト技法である．

7. おわりに

統計的テスト法をより効果的なものとするために，多重マルコフ連鎖に基づく正確な利用モデル（精密化利用モデル）を構築する手法を提案した．精密化利用モデルは，標準利用構造とフィールドデータから構築する．標準利用構造は仕様の一部として定義されるステートマシンのことであり，また，フィールドデータはソフトウェアの利用状況を表す遷移列またはイベント列のことである．精密化利用モデルが従来の利用モデルと異なるのは，すべての状態が過去の状態遷移に関する記憶を持ち，イベントの生起確率がその記憶の影響を受けるという点である．これによって，ユーザの次の振舞いがそれまでの振舞いの内容に影響されるような状況を表現できるようになった．本稿では，そのような精密化利用モデルが従来の利用モデルよりも正確であり，ソフトウェア信頼性を正確に評価するうえで役立つこと，そして本手法による統計的テスト法が実際のソフトウェア開発に適用できることを示した．

本研究の適用実験において，ソフトウェア開発企業の品質管理部に所属する技術者と議論を行った結果，以下のような意見が得られた．

- ソフトウェア信頼性による評価は新たな試みであったが，従来とは異なる視点を持ったメトリクスとして期待できる．テスト作業の成果を測定したり，出荷判定したりする際に活用できる．
- より有効化するための取り組みとして，着眼点や依存関係によって利用モデルを分割し再利用する方法を確立したり，ソフトウェアのログや障害レポートなどのフィードバックをユーザから受ける環境を整備したりすることなどが考えられる．

今後の研究では，ソフトウェア開発現場での適用をより広範囲，長期間にわたって実施し，本手法による統計的テスト法の有効性を検証する予定である．

謝辞　本研究の適用実験を行うにあたって，株式会社ジャストシステムの三橋尊志，山本知，藤井禎，丸山玄樹の諸氏に多大なるご協力をいただいた．ここに深く感謝の意を表する．

参考文献

1) Myers, G.J.: *The Art of Software Testing*, John Wiley & Sons (1979).
2) Beizer, B.: *Software Testing Techniques, 2nd edition*, Van Nostrand Reinhold (1990).
3) Binder, R.V.: *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison-Wesley (1999).
4) Whittaker, J.A. and Poore, J.H.: Markov Analysis of Software Specifications, *ACM Trans. Software Engineering and Methodology*, Vol.2, No.1, pp.93–106 (1993).
5) Whittaker, J.A. and Thomason, M.G.: A Markov Chain Model for Statistical Software Testing, *IEEE Trans. Softw. Eng.*, Vol.20, No.10, pp.812–824 (1994).
6) 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).
7) Musa, J.D.: Operational Profiles in Software Reliability Engineering, *IEEE Software*, Vol.10, No.2, pp.14–32 (1993).
8) Musa, J.D.: The Operational Profile, *Computer and System Sciences*, NATO ASI

Series F, Vol.154, pp.333–344 (1996).
9) Ching, W. and Ng, M.K.: *Markov Chains: Models, Algorithms and Applications*, Springer (2006).
10) 高木智彦，古川善吾：多重マルコフ連鎖を用いた統計的テストのための精密化利用モデルについて，情報処理学会研究報告，Vol.2007, No.157, pp.71–77 (2007).
11) Takagi, T. and Furukawa, Z.: Construction Method of a High-Order Markov Chain Usage Model, *Proc. 14th Asia-Pacific Software Engineering Conference*, pp.120–126 (2007).
12) 佐藤武久，大槻　繁，金藤栄孝：ソフトウェアクリーンルーム手法，日科技連，東京 (1997).
13) Chruscielski, K. and Tian, J.: An Operational Profile for the Cartridge Support Software, *Proc. 8th International Symposium on Software Reliability Engineering*, pp.203–212 (1997).
14) Hartmann, H., Bokkerink, J. and Ronteltap, V.: How to reduce your test process with 30% – The application of Operational Profiles at Philips Medical Systems, *Supplementary Proc. 17th International Symposium on Software Reliability Engineering*, CD-ROM (2006).
15) Kallepalli, C. and Tian, J.: Usage measurement for statistical web testing and reliability analysis, *Proc. 7th International Software Metrics Symposium*, London, UK, pp.148–158 (2001).
16) Shukla, R., Carrington, D. and Strooper, P.: Systematic Operational Profile Development for Software Components, *Proc. 11th Asia-Pacific Software Engineering Conference*, pp.528–537 (2004).
17) Object Management Group: *UNIFIED MODELING LANGUAGE*, Object Management Group (online). http://www.uml.org/ (accessed 2009–12–22)
18) Takagi, T., Nishimachi, K., Muragishi, M., Mitsuhashi, T. and Furukawa, Z.: Usage Distribution Coverage: What Percentage of Expected Use Has Been Executed in Software Testing?, *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Studies in Computational Intelligence, Vol.209, Springer, pp.57–68 (2009).
19) Rook, P.: *Software Reliability Handbook*, Elsevier Science (1990).
20) Sayre, K. and Poore, J.H.: Stopping criteria for statistical testing, *Information and Software Technology*, Vol.42, No.12, pp.851–857 (2000).
21) Prowell, S.J.: Using Markov Chain Usage Models to Test Complex Systems, *Proc. 38th Hawaii International Conference on System Sciences*, USA, p.318c (2005).
22) 青山幹雄，内平直志，平石邦彦：ペトリネットの理論と実践，朝倉書店 (1995).
23) Gokhale, S.S.: Architecture-Based Software Reliability Analysis: Overview and Limitations, *IEEE Trans. Dependable and Secure Computing*, Vol.4, No.1, pp.32–40 (2007).
24) Prowell, S.J.: TML: A description language for Markov chain usage models, *Information and Software Technology*, Vol.42, No.12, pp.835–844 (2000).
25) Prowell, S.J.: JUMBL: A tool for model-based statistical testing, *Proc. 36th Hawaii International Conference on System Sciences*, USA, p.337c (2003).
26) Chow, T.S.: Testing Software Design Modeled by Finite-State Machines, *IEEE Trans. Softw. Eng.*, Vol.SE-4, No.3, pp.178–187 (1978).
27) Offutt, J. and Abdurazik, A.: Generating Tests from UML Specifications, *Proc. 2nd International Conference on the Unified Modeling Language*, pp.416–429 (1999).
28) Takagi, T. and Furukawa, Z.: GB Coverage Criteria: The Measurement for Testing a "Go Back" Function Based on a Pushdown Automaton, *Proc. 19th International Symposium on Software Reliability Engineering*, pp.293–294 (2008).

(平成 22 年 1 月 6 日受付)
(平成 22 年 6 月 3 日採録)

高木　智彦（正会員）

2002 年香川大学工学部卒業．2004 年同大学院工学研究科修士課程修了．2007 年同大学院博士後期課程修了．2008 年香川大学工学部助教，現在に至る．博士（工学）．ソフトウェア工学，特にソフトウェアテスト法に興味を持つ．電子情報通信学会会員．

古川　善吾（正会員）

1975 年九州大学工学部卒業．1977 年同大学院工学研究科修士課程修了．同年日立製作所システム開発研究所勤務．1986 年九州大学工学部情報工学科助手，1990 年九州大学工学部講師，1992 年九州大学情報処理教育センター助教授，1998 年香川大学工学部教授，2007 年香川大学総合情報センター長，現在に至る．博士（工学）．ソフトウェア工学，特にソフトウェアテスト法，分散システム/インターネットの運用管理等の研究に従事．電子情報通信学会，ソフトウェア科学会，ACM，IEEE-CS，ISOC 各会員．