ソフトウェア工学 94－10
(1993. 9. 21)

プログラマのテスト・デバッグ能力の自動計測環境

藤田房之† 高田義広† 松本健一‡ 鳥居宏次‡

† 大阪大学 基礎工学部
‡ 奈良先端科学技術大学院大学 情報科学研究科

† 〒560 大阪府豊中市待兼山町 1-1
‡ 〒630-01 奈良県生駒市高山町 8916-5

あらまし ソフトウェアの生産性や品質は，プログラマの能力に大きく依存することが指摘されている．しかし，プログラマの能力を客観的に測定するために収集すべきデータの種類や収集の方法は明らかになっていない．

本報告では，プログラマのテスト・デバッグ能力を評価するための準備段階として，テスト・デバッグ過程の状態遷移のモデルを提案する．更に，提案するモデルのパラメータの推定に必要となる状態遷移時刻を，プログラマが入力したキーストロークのデータから自動的に推定する計測環境について述べる．評価実験の結果，開発した計測環境を用いることにより，プログラマの作業状況を撮影したビデオを観察することなしに，プログラマの状態遷移時刻の特定が可能であることが分かった．

和文キーワード 計測環境，プログラマ能力，テスト工程，プログラマ状態，キーストローク

An Environment for Automatically Measuring Programmers' Capability of Testing and Debugging

Fusayuki Fujita† Yoshihiro Takada† Ken-ichi Matsumoto‡ Koji Torii‡

† Faculty of Engineering Science, Osaka University
‡ Graduate School of Information Science, Nara Institute of Science and Technology

† 1-1 Machikaneyama, Toyonaka, Osaka 560, Japan
‡ 8916-5 Takayama, Ikoma, Nara 630-01, Japan

Abstract This paper proposes a state transition model of software testing and debugging processes in order to evaluate the capability of a programmer in a quantitative and objective way. The proposed model includes three states which correspond to typical activities of programmer, and has some parameters which can be used to define metrics of programmers' capability in software testing and debugging processes. In addition, this paper describes a measurement environment which automatically collects and analyzes data of programmers' activities from workstations in software development. The measurement environment estimates the parameters of the proposed model, especially the timing of state transition, based on sequence of key stroke collected from the workstations. Analysis of Experimental evaluation in an academic environment shows the validity and usefulness of the measurement environment.

英文 key words measuring environment, programmer capability, testing process, programmer state, key stroke

1 まえがき

ソフトウェアの生産性や品質は，ソフトウェアを開発するプログラマの能力に大きく依存することが指摘されている [3, 6, 8]．従って，プログラマの能力を客観的に評価することは，開発管理者にとって非常に重要な問題である．しかし，従来のプログラマ能力の評価は，各プログラマの経験年数や開発管理者の主観的な評価などに頼っているに過ぎず，客観的な評価が行われているとはいえない [3, 6]．

また，プログラマ能力の測定自体容易ではなく，プログラマ能力を測定するために収集すべきデータの種類や収集方法さえ明らかになっていないのが現状である [3, 6]．プログラマ能力を客観的，かつ，容易に測定することができれば，従来の開発管理技術の大幅な改善が図られるだけでなく，プログラマ教育においても非常な効果が期待される．

我々は，特にテスト・デバッグ作業に着目し，プログラマのテスト・デバッグ能力の測定を目的とした自動計測環境の構築を目指している [8]．そのために，ビデオ装置などを利用して実際のテスト・デバッグ作業を観察し，テスト・デバッグ能力を測定する上で有効なデータを見い出すことを試みている．これまでの観察結果として，プログラマの動作を撮影したビデオから，プログラマの作業は大きく 3 つの状態に分類できること，プログラマの状態の遷移が判別できること，プログラマの能力の差によってその状態遷移のパターンに差が認められることなどが分かった．また，撮影したビデオからプログラマの状態を判別するためには多大な労力を必要とするが，同様の判別をプログラマのキーストロークをリアルタイムに監視することにより自動的に行えることが分かった．

本報告では，プログラマのテスト・デバッグ能力を評価するための準備段階として，テスト・デバッグ過程の状態遷移のモデルを提案する．更に，提案するモデルのパラメータの推定に必要となる状態遷移時刻を，プログラマが入力したキーストロークのデータから自動的に推定する計測環境について述べる．

2 テスト・デバッグ過程のビデオ撮影実験

プログラマのテスト・デバッグ能力の客観的な評価に必要なデータとその収集方法を調べるために，実際のテスト・デバッグ過程をビデオ撮影し，撮影したビデオの分析を行った．

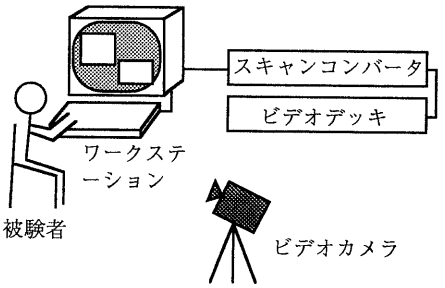


図 1　ビデオ撮影実験

2.1 ビデオ撮影

大阪大学基礎工学部情報工学科の大学院生 4 名を被験者とし，それぞれにプログラムを開発させ，そのテスト・デバッグ過程をビデオ撮影した．

被験者に与えた課題 (仕様書) は，比較的小規模なものである．課題に基づいて，被験者には 3 つの作業を以下の順序で行ってもらった．

1. 設計作業: 仕様書を理解し，プログラムの概略を設計する．
2. コーディング作業: C 言語を用いてコーディングを行う．
3. テスト・デバッグ作業: テストデータを用いたプログラムの動作確認とデバッグを行う．

即ち，テスト・デバッグ作業が開始される時点では，すべての設計作業，及び，コーディング作業は終了しているものとする．従って，テスト・デバッグ作業中に行われた設計書やプログラムの変更は，すべてデバッグを目的としたものとみなす．

撮影は，デッキ内蔵型のビデオカメラ 1 台，ビデオデッキ 1 台を用いて行った (図 1 参照)．ビデオカメラは，被験者 (プログラマ) の行動を把握するため，斜め後方からの撮影に用いた．ビデオデッキは，計算機 (ワークステーション) 上での作業内容を把握するため，計算機の画面出力の記録に用いた．スキャンコンバータは，計算機の画面出力信号をビデオデッキで記録可能な形式に変換するためのものである．

2.2 テスト・デバッグ過程の分析

撮影したビデオを観察した結果，テスト・デバッグ作業中のプログラマの状態を次の 3 つに分類することができた．

コンパイル状態 (Compiling): コンパイラを起動してからコンパイルが終了するまでの状態．ただし，コンパイルの結果，シンタックスエラーが発見された場合は，プログラムの再編集によってシンタックスエラーが完全に除去され，コンパイルによってそのことが確認できるまでの状態．

テスト実行状態 (Executing): プログラムを実行し，その実行結果を確認している状態．または，実行したプログラムに誤りがある場合は，それがどのような誤りであるかを確認している状態．

デバッグ状態 (Debugging): 確認された誤りの原因を特定している状態，または，発見後，プログラムを再編集している状態．誤りの原因を特定している状態とは，テスト実行後に結果を眺め考えている状態，デバッガを使用している状態，テキストファイルを眺めている状態である．

ある被験者のテスト・デバッグ過程における状態遷移の様子を表すタイミングチャートを図2に示す．

各被験者のタイミングチャートを観察した結果，次の3つのことが分かった．(1) すべての被験者において，状態遷移はある一定の順序で起こる．即ち，コンパイル状態，テスト実行状態，デバッグ状態，再び，コンパイル状態というサイクルを繰り返す．(2) テスト実行状態の継続時間は，サイクルの繰り返しとともに長くなる傾向がある．(3)1 サイクルに費される時間，テスト実行状態，及び，デバッグ状態の継続時間の変化は，被験者によって異なる．(1),(2) より，前述の3つの状態間をサイクリックに遷移し，各状態の継続時間が遷移を繰り返すことにより変化するというモデルにより，テスト・デバッグ過程を表すことができると考える．また，(3) に示した差は，プログラマがテストによって効率良くバグを発見しているかどうか，デバッグ作業で確実にバグを除去しているかどうかによって生じるものである．従って，この違いを表現できるモデルを構築すれば，プログラマ能力 (テスト・デバッグ能力) の評価が可能になるのではないかと考える．

3 テスト・デバッグ過程のモデル

2で述べたテスト・デバッグ過程の撮影実験より，プログラマ能力の差は，コンパイル状態，テスト実行状態，デバッグ状態の間をサイクリックに遷移するのに要する時間や，各状態の継続時間，継続時間の変化等によって評価できるものと仮定する．

本章では，こうした仮定のもとに，実験結果を形式化し，プログラマ能力の差異による状態遷移の相違を説明するための1つの仮説モデルを提案する．

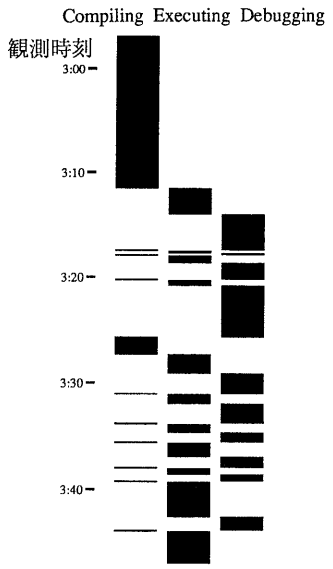


図2 テスト・デバッグ過程における状態遷移

3.1 モデルの定義

まず，状態遷移について，すべての被験者に共通な点を整理する．

(C1) コンパイル状態，テスト実行状態，デバッグ状態，コンパイル状態の順に遷移を繰り返す．

(C2) コンパイル状態からデバッグ状態への遷移，デバッグ状態からテスト実行状態への遷移，テスト実行状態からコンパイル状態への遷移は起こらない．

(C3) コンパイル状態，および，デバッグ状態の，サイクルの繰り返しに伴う継続時間の変化には，目立った傾向はない．

(C4) テスト実行状態の継続時間は，サイクルの繰り返しとともに長くなる．

以上の共通点を考慮して作成した，テスト・デバッグ過程におけるプログラマの状態遷移モデルを図3に示す．このモデルは，コンパイル状態 (C)，テスト実行状態 (T)，デバッグ状態 (D) の3状態からなっている．

プログラマ状態は単位時間 (1秒) 毎に遷移する．各矢印は1回の状態遷移を表している．共通点 (C1),(C2)

より，状態間の遷移はC→T，T→D，D→Cの3つのみである．また，ある期間，別の状態に遷移せずに，ある状態に留まることも考えられるが，そのような場合は，同じ状態に自己遷移を繰り返しているものとみなす．従って，どのプログラマ状態においても，各時刻においては，自己遷移と他の1つの状態への遷移の2通りの遷移がある．

本モデルでは，各時刻において遷移する先の状態は確率的に決定されるとする．矢印に付けられたラベルは，これらの遷移確率を表している．p_C はデバッグ状態からコンパイル状態に遷移する確率，p_T はコンパイル状態からテスト実行状態に遷移する確率，p_D はテスト実行状態からデバッグ状態に遷移する確率をそれぞれ表す．これらの確率は，各状態に留まる時間の長さを表すパラメータである．例えば，デバッグ状態からコンパイル状態に遷移する確率 p_C が高くなると，デバッグ状態に留まる長さは短くなる．

共通点(C3)より，p_C，及び，p_Tは定数とする．一方，共通点(C4)より，p_Dはある時刻までにテスト実行状態からデバッグ状態に遷移した回数を fの関数とし，$p_D(f)$ で表すことにする．具体的には，$p_D(f)$ を次式で定義する．

$$p_D(f) = \alpha(f_0 - f)$$

ここで f はテスト・デバッグ作業が始まってからのデバッグ状態への遷移回数である．f_0 は作業対象となっているプログラム中のすべてのバグを除去するために必要な，デバッグ状態への遷移回数である．αはデバッグ状態への遷移1回当たりの $p_D(f)$ の減少率を表す定数である．

3.2 モデルとプログラマ能力

3.1で定義した状態遷移モデルとプログラマ能力の関係について述べる．

プログラマ能力，特にテスト・デバッグ能力の高いプログラマとは，テストによってバグを発見するのに要する時間が短く，発見したバグを修正するのに要する時間が短いプログラマと考えられる．

テストによってバグを発見するのに要する時間はテスト実行状態に留まっている時間 T_Tに，発見したバグを修正するのに要する時間はデバッグ状態に留まっている時間 T_Dにそれぞれ対応する．T_T, T_Dの期待値はそれぞれ次式で表される．

$$E(T_T) = \frac{1}{p_D(f)} = \frac{1}{\alpha(f_0 - f)}$$

$$E(T_D) = \frac{1}{p_C}$$

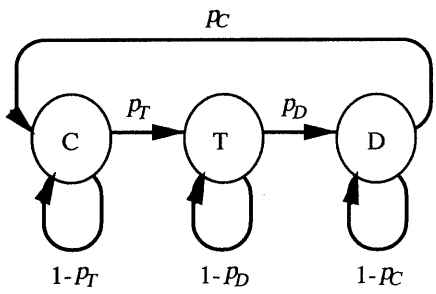


図3　プログラマの状態遷移モデル

デバッグ状態への1回の遷移によってプログラム中のバグが1個除去されると仮定すると，f_0はテスト・デバッグ作業開始時にプログラム中に存在したバグの総数となる．プログラム中に存在したバグの総数は，テスト・デバッグ能力とは直接には関係ない．従って，プログラム能力，特にテスト・デバッグ能力は，3.1で定義した状態遷移モデルのパラメータαと p_Cによって表されることになる．(α，および，p_Cの値が大きいほど T_T，及び，T_Dの期待値は小さくなり，テスト・デバッグ能力が高いことになる．)

4 プログラマ状態の自動計測環境

本章では，プログラマのキーストロークを監視し，解析することによって，各単位時間におけるプログラマの状態を自動的に判別する目的で実験的に開発した環境について述べる．この環境では，各単位時間におけるプログラマの状態を，2.2で示した3つの状態に分離することができる．

自動計測環境は，キーストローク記録部，キーストローク系列解析部，状態判別部からなる(図4参照)．

キーストローク記録部は，計算機に対するキーストロークをリアルタイムに取り込み，ファイルに記録する．キーストローク記録部は，マルチウィンドウ作業環境に対応しており，プログラマによって開かれたウィンドウ毎に，すべてのキーストロークを記録する．

キーストローク系列解析部は，記録されたキーストロークデータの系列から，コマンドやツールの起動を行うためにプログラマが入力した部分(文字列)を入力時刻とともに抽出する．解析部は，シェル(csh)，エディタ(vi, emacs)，デバッガ(dbx)などの内部コマンドについても同様に抽出可能である．

状態判別部は，解析部によって抽出された文字列に基づいて，各単位時間におけるプログラマの状態を判

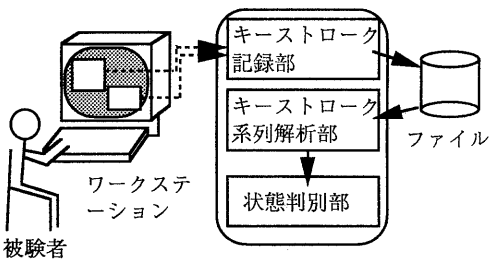


図4　状態遷移時刻の自動計測環境

別する．具体的には，各状態のキーストローク系列に現れる特有の部分系列(コマンド名など)を，解析部が抽出した文字列の中から見つけ出し，各状態の開始時刻と終了時刻を特定する．以下に，プログラマ状態毎にその開始時刻と終了時刻の特定法の詳細を述べる．

コンパイル状態の開始時刻は，コンパイラ(cc, gcc, make)を起動するためのシェルコマンドの入力時刻とする．終了時刻は，コンパイラとエディタ以外のツールを起動するためのシェルコマンドが入力された時刻とする．コンパイラが続けて起動された場合は，さらにコンパイル状態が続くものとする．エディタが起動された場合は，シンタックスエラーを除去するためにエディタを起動したとみなされる．

テスト実行状態の開始時刻は，プログラマによって作成されたプログラムを起動するためのシェルコマンドの入力時刻とする．シェルコマンド名は作成されたプログラム名により異なるが，プログラマが作業を行っているディレクトリ内の実行可能なファイル名を探すことにより容易に特定できる．テスト実行状態終了時刻は，エディタ，あるいは，デバッガを起動するためのシェルコマンドの起動時刻とする．(エディタやデバッガは，バグを発見するために用いられるコマンドである．)ただし，テスト実行後(作成されたプログラムを起動するためのシェルコマンドの実行終了後)一定時間以上エディタやデバッガの起動が行われない場合には，デバッグ状態に遷移したとみなし，テスト実行状態の終了時刻は，プログラムを起動するためのシェルコマンドの実行終了時刻とする．これは，エディタやデバッガを用いずにバグを発見した場合，プログラマはキー入力をせずにバグの除去方法について考え込む(デバッグ状態となる)ことが多いからである．

デバッグ状態の開始時刻は，エディタ，あるいはデバッガを起動するためのシェルコマンドの入力時刻とする．ただし，プログラムを起動するためのシェルコマンドの実行終了後，一定時間以上，エディタやデバッガが起動されなかった場合は，シェルコマンドの終了時刻を状態の開始時刻とする．デバッグ状態の終了時刻は，コンパイラを起動するためのシェルコマンドの入力時刻とする．

5　評価実験

本章では，自動計測環境の性能を評価する．具体的には，ビデオを観察して特定した各状態の遷移時刻と，自動計測環境を用いて特定した各状態の遷移時刻の誤差を調べる．

5.1　実験の概要

実験方法は2で述べたビデオ撮影実験と同じである．ただし，被験者には自動測定環境上でプログラム開発を行わせ，キーストロークを収集した．

被験者は奈良先端科学技術大学院大学情報工学研究科の大学院生4名である．課題はボーリングのスコア計算問題など比較的小規模で，数時間でテスト・デバッグが行えるものである．課題名，プログラムサイズ，テスト・デバッグに要した時間を表1に示す．

5.2　比較結果

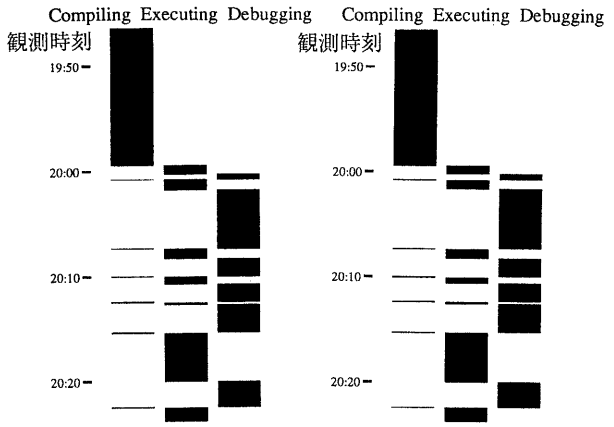
被験者Bの，ビデオを観察して判別した状態遷移のタイミングチャートと，自動計測環境を用いて判別したタイミングチャートを図5に示す．

状態の遷移系列は，両者全く同じである．即ち，自動計測環境は，発生したすべての状態遷移を正しく検出し，かつ，実際は遷移が発生していない時刻に遷移が起こったと誤って判定することもなかった．また，他の3名の被験者のデータに対しても同様の結果が得られた．従って，自動計測環境の状態遷移検出性能は非常に高いといえる．

次に，自動計測環境による状態遷移の検出精度を評価する．ビデオを観察して特定した状態遷移時刻と自動計測環境を用いて特定した状態遷移時刻の差(誤差)の最大値，平均値，標準偏差を表2に示す．

表2より，誤差は最大でも8秒，平均では1秒前後と，自動計測環境の検出精度が非常に高いことが分かる．誤差が±4秒以下である割合を表3に示す．±4秒とは，状態遷移の1サイクルに要する時間の平均値467秒の約1%であり，モデルのパラメータ推定等に及ぼす影響が非常に小さいと思われる誤差の範囲である．

表3より，ほとんどの場合，誤差は±4秒の範囲に



(a)ビデオ観察による判別　(b)自動計測環境による判別

図5　被験者Bのタイミングチャート

表1　課題とテスト・デバッグ時間

被験者	課題名	プログラムサイズ(行)	テスト・デバッグ時間(分)
A	ボウリングのスコア計算	168	75
B	万年カレンダー	401	63
C	ボウリングのスコア計算	200	195
D	戦闘開始ゲーム	164	87

表2　全データに対する検出誤差

コンパイル状態への遷移			テスト実行状態への遷移			デバッグ状態への遷移		
最大	平均	標準偏差	最大	平均	標準偏差	最大	平均	標準偏差
3	-0.46	1.15	5	0.57	2.34	8	-1.20	2.20

(最大誤差，平均誤差の単位：秒)

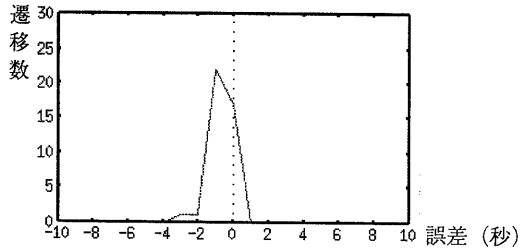
表3　|誤差|≦4秒となる割合

コンパイル状態への遷移	テスト実行状態への遷移	デバッグ状態への遷移
100.0%	93.1%	90.7%

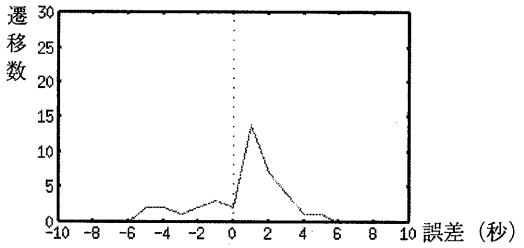
表4　被験者ごとの検出誤差

被験者	サイクル数	コンパイル状態への遷移			テスト実行状態への遷移			デバッグ状態への遷移		
		最大	平均	標準偏差	最大	平均	標準偏差	最大	平均	標準偏差
A	7	3	1.57	1.11	4	2.12	1.05	8	-2.14	3.02
B	6	2	-0.71	0.70	5	-1.29	2.43	4	2.00	2.24
C	38	3	-0.66	0.49	5	0.05	2.37	5	-1.74	1.52
D	3	3	-1.80	0.98	3	2.00	2.12	7	-2.00	3.37

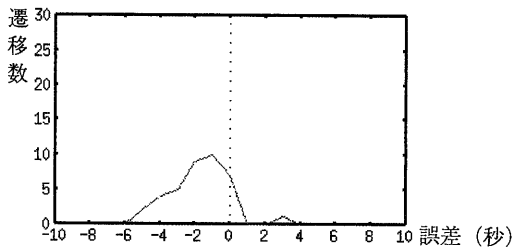
(最大誤差，平均誤差の単位：秒)



(a)コンパイル状態への遷移時刻



(b)テスト実行状態への遷移時刻



(c)デバッグ状態への遷移時刻

図6　被験者Cにおける誤差の分布

入ることが分かる．特に，コンパイル状態の遷移に関しては，すべての場合において誤差がこの範囲に入っている．もっとも誤差が大きい，デバッグ状態への遷移においても，約90%という高い割合になっている．

サイクル数がもっとも大きい被験者Cにおける誤差の分布を図6に示す．図6(a)より，デバッグ状態からコンパイル状態への遷移においては，自動計測環境はわずかに早い時刻を遷移時刻とする傾向にあることが分かる．これは，ビデオ観察では，コンパイルのためのコマンドを入力し始めた時点で状態が遷移したとみなすためと考えられる．逆に，コンパイル状態からテスト実行状態への遷移においては，わずかに遅い時刻を遷移時刻としている．これは，ビデオ観察では，コンパイラ終了のプロンプトが表示された時点で遷移するとしているのに対し，自動計測環境ではテストコマンドの入力により遷移したとみなしているからである．

テスト実行状態からデバッグ状態へ遷移する時刻の誤差は，他の2つの遷移に比べて大きい．図6(c)より，自動計測環境では，デバッグ状態への遷移において，実際の時刻よりわずかに早い時刻を遷移時刻とみなしていることが分かる．これは，4.2で述べたように，特定のコマンドの入力によってデバッグ状態が開始したと判定できるわけではないためである．精度をあげるためには，テスト実行終了後の数秒間はバグの原因特定の時間として，デバッグ状態への遷移時刻を数秒遅らせる等の改良が考えられる．

以上の結果から，自動計測環境を用いて状態の遷移時刻を特定しても，ビデオを観察して状態の遷移時刻を特定した場合とほとんど同じ結果が得られることが分かる．

6　まとめ

本報告では，プログラマのテスト・デバッグ能力を評価するための準備段階として，テスト・デバッグ過程におけるプログラマの作業をビデオを用いて分析し，その結果に基づいて，テスト・デバッグ過程のモデルを提案した．更に，提案したモデルのパラメータ推定に必要となる状態遷移時刻を，プログラマが入力したキーストロークのデータを用いて自動的に計測する環境を実験的に開発した．

評価実験の結果，自動計測環境を用いて状態の遷移時刻を特定しても，ビデオを観察して状態の遷移時刻を特定した場合とほとんど同じ結果が得られることが分かった．

これにより，モデルのパラメータの推定に必要なデータである各状態への遷移時刻が，ビデオを観察することなしに自動的に収集することが可能となった．

今後は，モデルのパラメータである各状態への遷移確率と，プログラマの能力との関連について，さらに考察を行い，テスト・デバッグ能力の自動計測環境への拡張を行いたい．

参考文献

[1] Boehm B. W.: “Software engineering economics,” IEEE Trans. on Software Eng., vol. SE-10, no. 1, pp. 4–21, 1984.

[2] 前田, 中野:“キー入力を記録するための一方法”, 電子情報通信学会技術研究報告, ET88-7, pp.13-16, 1988.

[3] Matsumoto K.: “A programmer performance model and its measurement environment”, Ph. D. dissertation, Faculty of Engineering Science, Osaka University, Japan, 1990.

[4] 三輪, 櫻井, 岡田, 岩田:“初心者のプログラミング行動時系列分析”, 電子情報通信学会技術研究報告, ET93-47, pp.9-16, 1993.

[5] 三輪, 櫻井, 熊谷, 岡田:“プログラミング過程における初心者のエディト行動”, 電子情報通信学会技術研究報告, ET90-97, pp.73-80, 1990.

[6] 宮本 勲:“ソフトウェアエンジニアリング: 現状と展望”, TBS 出版会, 1982.

[7] 森村英典, 高橋幸雄:“マルコフ解析”, 日科技連出版社, 1979.

[8] 瀬部 紹夫:“テスト工程におけるプログラマの確率的な状態遷移モデルの提案”, 大阪大学基礎工学部情報工学科特別研究報告, 1993.