論　文

プログラマのデバッグ能力をキーストロークから測定する方法

正　員　高田　義広[†]　　正　員　鳥居　宏次[†]

A Method for Measuring Programmer Debugging Performance from Key Strokes

Yoshihiro TAKADA[†] *and* Koji TORII[†], *Members*

あらまし　ソフトウェア開発管理の観点から，プログラマの能力を客観的，正確，かつ，容易に測定する方法が要求されている．この問題に対して，本論文では，キーストロークからプログラマのデバッグ能力を測定する方法を提案する．ここで言うデバッグ能力とは，ソフトウェア故障を発見してからその原因を修正するまでの時間的効率である．提案する方法では，まず，キーストロークを監視して，各時刻のプログラマの活動を分類する．次に，得られた活動系列にプログラマモデルを適合させることによって，プログラマの特徴を表すパラメータを抽出する．このプログラマモデルは，活動系列をマルコフ過程とみなして定義した．最後に，これらのパラメータからデバッグ能力の評価値を算出する．主な評価値は，1個の欠陥の修正に要する時間の推定値 D である．適用実験の結果から，本方法が実際のソフトウェア開発において有効に働くことがわかった．特に，生産性に関する尺度（共通な仕様に対するプログラムの作成時間）と D との間に強い相関が見出せた．

キーワード　プログラマ，プログラマ能力，キーストローク，マルコフ過程，デバッグ

1. まえがき

ソフトウェアの生産性や品質は，その開発プロジェクトに参加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[(2),(8),(10),(11)]．文献(3)では，標準的な開発チームと比較して，能力の低い開発チームの開発時間が平均して4.2倍ほど長くなると述べている．また，文献(10)では，同一条件での実験において，プログラムの作成時間がプログラマ間で最大20倍以上異なっていたと報告している．従って，開発チームを編成し，プログラマに仕事を割り振るプロジェクト管理者にとって，組織内の各プログラマの能力を理解しておくことは重要な問題である．

このような重要性にもかかわらず，プログラマ能力を評価する従来の試みは必ずしも成功していない．共通の仕様を対象とするという制約のもとでプログラムを実際に作成させる試験を行えば，作成に要した時間などにより，能力の定量化が可能である．しかし，そのような試験には多くの費用が必要である．プログラマを経験した年数や経験したプロジェクトの回数などを能力の尺度として利用することも考えられる．しかし，そのような尺度はプログラマとしての適性を考慮していないので，正当な評価とは言えない．プログラマ適性試験と呼ばれる筆記試験を実施している組織もある．しかし，文献(11)にも報告されているように，それらは理論的，あるいは，統計的な根拠に基づく方法でなく，必ずしも適切な評価ではない．実際の企業では，プロジェクト管理者が，過去のプログジェクトの中で断片的に観察した作業の様子から直観的に評価を下すようである．しかし，このような評価には多くの経験を必要とするし，主観に左右される危険がある．以上の観点から，プログラマ能力を評価するためには，できるだけ制約のない条件のもとで実際の作業の様子を観察すること，および，観察から得られるデータを客観的に分析することが要求される．

本論文では，プログラマのデバッグの能力に注目し，上述の方針に従ってプログラマ能力を測定する方法を提案する．ここで言うデバッグ能力とは，1個の故障（software failure）を発見してからその原因を修正するまでの時間的効率である．プログラミングとは修正の積み重ねと見ることができるので，デバッグ能力を測

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Advanced Institute of Science and Technology, Nara, Ikoma-shi, 630-01 Japan

定することは重要な意味をもつ．

提案する方法では，プログラマの作業を観察するために，キーストロークに着目する．プログラマの作業の大部分は計算機に向かって行われるので，キーストロークの観察は有効な手段であると考えられる．文献(8)でもキーストロークの利用が試みられているが，頻度情報だけを利用して，エディタ操作の習熟を分析していた．提案する方法では，プログラマの入力するすべての命令をキーストロークから解析し，その結果から各時刻におけるプログラマの活動を分類する．特に，デバッグ能力に注目しているので，最初のコンパイルから以降の活動を数時間にわたって分類する．分類の結果，デバッグ作業中のプログラマ活動の時系列(プログラマ活動系列)が秒間隔で得られる．なお，文献(6)では，心拍などの生理的情報からプログラマの様子を観察することも試みられているが，プログラマ能力を評価するまでの詳細なデータは得られていない．文献(7)では，ソースプログラムのバージョンの履歴から推定したプログラムの変更量をプログラマ能力と関連づけることが試みられているが，データ収集の時間間隔が長すぎるために変更量の推定精度に問題が残されている．

熟練者と非熟練者の特徴を抽出するためには，キーストロークからこのように得られるプログラマの活動系列に対して，更なる分析が必要である．そのために，提案する方法では，プログラマの確率的な状態遷移モデルを導入する．このプログラマモデルを定義するに際しては，活動系列がマルコフ過程(Markov process)であると仮定した．マルコフ性を仮定しても実際の活動を十分に表現できるし，マルコフ性を仮定することによって後の分析が容易になると考えたからである．状態の個数や状態遷移の規則は，活動系列を実際に観測した結果に基づいて決定した．熟練者と非熟練者の特徴は，遷移確率などのモデルのパラメータにおける差として表される．観測される活動系列からこれらのパラメータを推定することにより，プログラマの特徴を抽出することができる．

提案する方法の出力は，このプログラマモデルのパラメータから算出される．1個の欠陥(バグ，software fault)の修正に要する時間の推定値をはじめとする，3種類の評価値である．

2. 方法の概略

提案する方法では，1人のプログラマがデバッグ作業を続けている様子を数時間にわたって観察・分析する．1人のプログラマが1個のプログラムを作成する場合には，最初のコンパイルから以降の作業を観察する．図1のように，観察・分析の手順は5ステップからなる．なお，ステップ1から3までは，我々の開発した測定ツール(4)によって自動的に実行できる．ステップ4，5もツールによる自動化が可能である．

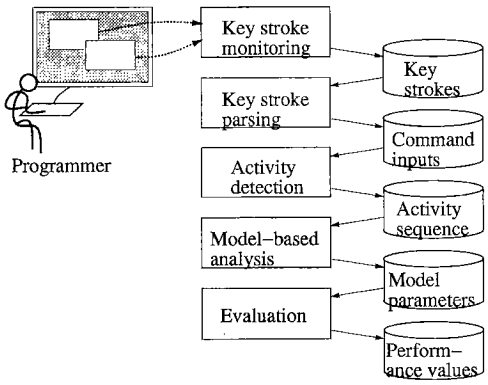

図1　方法の概略
Fig. 1　Outline of method.

[ステップ1]　キーストロークを監視する．画面上に複数のウィンドウを開くことができるプログラミング環境でも，各ウィンドウへのキーストロークをすべて記録する．なお，そのような環境では，一般にマウスの操作による文字列の複写と貼込みが行われる．そのような文字列の貼込みもキーストロークと同様に記録する．

[ステップ2]　記録したキーストロークを解析することにより，入力された命令(command)とその時刻を検出する．ここで言う命令とは，プログラムやツールを起動するための命令(シェル命令)だけでなく，各プログラムやツールの内部の命令も含む．例えばエディタ内の文字列の挿入命令や削除命令も検出する．

[ステップ3]　デバッグ作業中の各時刻におけるプログラマの活動を，入力された命令に基づいて分類する．このときのプログラマの活動はコンパイル・プログラム実行・プログム変更の3種類に分類できる(**3.1** 参照)．このステップの出力はそれら3種類の活動の秒間隔の時系列である．これをプログラマ活動系列(programmer activity sequence)と呼ぶ(**3.2**，および，図2参照)．

[ステップ4]　プログラマモデルを活動系列に適合させる(**4.1**，**4.2**，および，図3参照)．つまり，観測された活動系列からモデルのパラメータを推定する．これら

のパラメータによりプログラマの特徴が表される(**4.3, 4.4**, および表2参照). なお, パラメータを高精度で推定するためには, 活動系列が長い方がよい.

[ステップ5] 推定したモデルのパラメータから, プログラマの評価値を算出する. 1回のプログラム変更で欠陥を修正できない確率, 1回のプログラム変更に要する時間, および, 1個の欠陥の修正に要する時間の3種類を出力する(**5.**, および表3参照).

3. プログラマ活動の観測

3.1 活動の分類

外部からの観測によりプログラマの活動がどのように分類できるかを調べるために, 我々はビデオカメラを使用した予備実験を行った[4]. 結果として, デバッグ作業中の各時刻におけるプログラマの活動がコンパイル・プログラム実行・プログラム変更の3種類に分類できることがわかった. 文献(6)でも, 類似した3種類に分類できることが心理的・生理的観点から指摘されている.

提案する方法では, キーストロークを利用して, 各時刻のプログラマの活動を上述の3種類に分類する. そのために, ステップ3では, 入力されるすべての命令の中から各活動の開始時, および, 終了時に入力される命令を判別する. 以下では, 各活動の定義と具体的な判別方法を与える.

(コンパイル) ソースプログラムから実行可能ファイルを作成する間の活動である. コンパイラの起動だけでなく, ソースプログラム中の文法誤りをエディタにより除去する活動を含む. この活動の開始はコンパイラの起動命令により判定する(例: cc, gcc, make). 終了はコンパイラとエディタと以外のプログラムやツールの起動命令により判別する.

(プログラム実行) 故障を探す活動である. コンパイルしたプログラムを起動したり, 操作したり, 出力の正しさを確認することを繰り返す. この活動の開始はプログラムの起動命令により判別する(例: a.out). プログラムの名前は必ずしもa.outではないが, 作業ディレクトリ中の実行可能ファイルを検索することにより, 容易に発見できる. 終了はその他のプログラムやツールの起動命令により判別する.

(プログラム変更) 発見した不都合を修正する活動である. 故障の原因となる欠陥を探したり, 発見した欠陥の修正を試みる. この活動の開始はエディタ, または, デバッガの起動命令により判別する(例: vi, emcas, dbx). プログラマによっては, あるウィンドウにエディタを起動したまま別の活動を行うことがある. そのような場合は, エディタのウィンドウへの入力の切換えにより判別する. 終了にはコンパイラの起動命令により判別する.

3.2 観測実験

プログラマ活動系列を観測した実験について述べる. 合計で22回の実験を行った. 被験者はF, K, M, S, T, Y, Zの7名である. 7名とも情報科学を専攻する大学院生であり, K, M, S, Yは企業でソフトウェア開発に参加した経験をもつ.

各実験では, 1人の被験者が1個のプログラムを作成した. 1人の被験者について2～5回の実験を行った. 各実験では, 仕様を受け取ってからデバッグを終えるまでの全作業が, 監視者の付添のもとで, 連続して行われた. 各被験者は, 仕様の理解を含めた全作業を8時間以内に完了するよう指示された.

与えたプログラム仕様は, 1個の共通仕様を除き, 被験者ごとに異なる. どの仕様も小規模なプログラムの仕様である(例: パズルの作成, 求解, 単純な数値計算). 7名のすべてに与えた共通仕様とは, Huffman符号の作成である. 各仕様は, 自然語で注意深く書かれた2～5ページの文章と図表を使って伝えられた.

仕様の分析やプログラムの設計の方法についての指示は与えられなかった. なお, 被験者から監視者に対して仕様に関する質問はほとんどなかった. どの被験者もC言語を使用してコーディングを行った. 各被験者は, ソースコードを書き終わるまではコンパイルを行わないように指示が与えられた. 従って, 最初のコンパイルの時点で, コーディングからデバッグへ推移したとみなされた. デバッグに使用するツールは被験者が自由に選択した. 結果的に, F, S, T, Y, Zはデバッガを使用したが, 他の被験者は使用しなかった.

作成されたプログラムの行数と作成時間を表1に挙げる. いくつかの空欄が残されているが, それらは, 実験が8時間以内に終了しなかったために中止されたことを意味する. 表1から, 共通仕様に対してさえ作成時間が大きく分散していることがわかる. 被験者Tは54分で作成したが, 被験者KとMは8時間以内に完成できなかった.

観測した活動系列の例を図2に示す. 各活動系列は, 実験中の最初のコンパイルから始まり, 被験者がデバッグ作業の完了を宣言するまで続いている. 各グラフの縦軸は時間を表し, 10分間隔の目盛りが付けられてい

表1 実　　験

被験者	実験	仕様	作成行数	作成時間(分)
F	1	共通仕様	170	153
	2	仕様 W	181	216
	3	仕様 P	250	245
	4	仕様 F	—	—
K	1	共通仕様	—	—
	2	仕様 N	49	157
M	1	共通仕様	—	—
	2	仕様 G	164	288
S	1	共通仕様	166	156
	2	仕様 C	401	320
	3	仕様 A	—	—
	4	仕様 M	331	236
T	1	共通仕様	159	54
	2	仕様 L	113	77
	3	仕様 E	419	97
	4	仕様 A	387	183
	5	仕様 F	285	139
Y	1	共通仕様	127	228
	2	仕様 B	168	227
Z	1	共通仕様	137	395
	2	仕様 B	200	502
	3	仕様 F	—	—

る．3個の列は右から順にコンパイル(com.)，プログラム実行(exe.)，プログラム変更(mod.)に対応する．黒い方形印はその時間のプログラマの活動を表す．**4.1**で述べたように，プログラマごとに異なる特徴が見られる．

4. プログラマモデル

4.1 プログラマ間の共通点・相違点

図2にも現れているように，次の特徴はどのプログラマについても共通である．

(1) どのプログラマもコンパイル・プログラム実行・プログラム変更という単調な反復を繰り返す．例外は観測されなかった．

(2) 最初のコンパイルは，以降のコンパイルと比較して例外的に長い．それ以前の作業で作り込まれた多くの文法誤りが，1度に検出されるためである．

(3) プログラム変更が繰り返されるたびに，1回のプログラム実行の継続時間が長くなる傾向がある．欠陥の個数や故障の頻度が減少するためである．

プログラマ間で異なる特徴は次の点である．

(1) 状態遷移の頻度が大きく異なる．例えば，図2(d)，(e)，(g)では，活動の反復が比較的速い．一方，図2(b)，(c)では，1回のプログラム変更が平均的に長い．一般に，経験を積んだ(または，性急な)プログラマほど，反復が速い傾向がある．

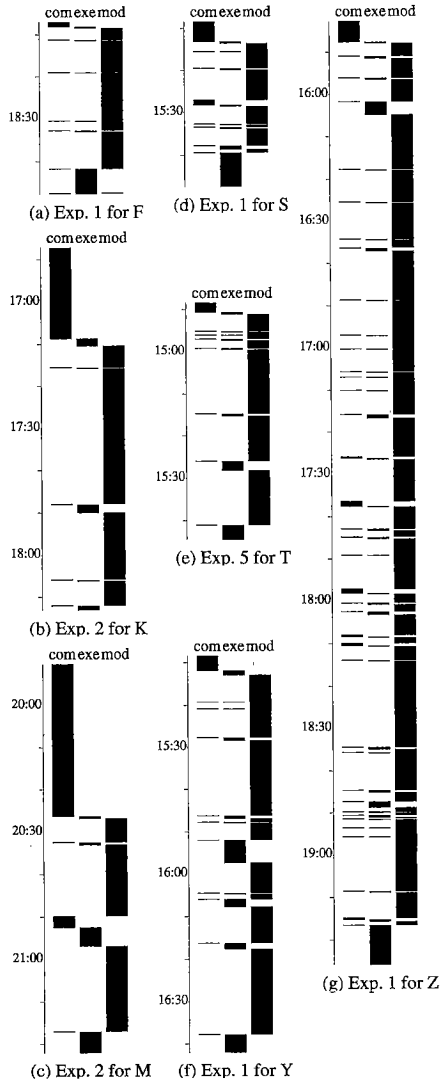

図2　プログラマ活動系列
Fig. 2　Programmer activity sequences.

(2) プログラム実行の継続時間が増加する様子が，プログラマに依存して大きく異なる．例えば，図2(c)，(e)では，1回のプログラム変更が終わるたびにプログラム実行の継続時間が確実に長くなっている．一方，図2(g)では，必ずしも増加しておらず，規則性がないようにさえ見える．この相違の原因は，実験後の被験者への質問やキーストロークの記録の追跡により以下のように判明した．一般に，経験が不足している(または，不注意な)プログラマは，欠陥を完全に修正しない

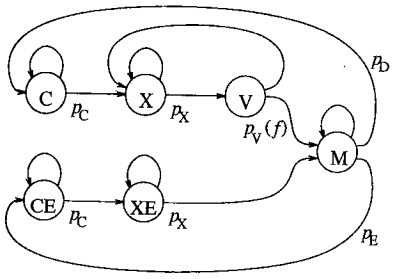

図3 テスト・デバッグ過程のプログラマモデル
Fig. 3 Programmer model in testing and debugging.

ままコンパイルを行うことがある．そのような場合，プログラム実行の開始直後に，以前に発見された故障が再発するので，プログラム変更が即座に再開されることになる．このような不完全なデバッグは不規則に短いプログラム実行として活動系列中に現れる．一方，経験を積んだ(または，慎重な)プログラマは1回のプログラム変更で1個の欠陥を確実に修正する．結果として，プログラム実行の継続時間が単調に増加することになる．

以上の分析からわかるように，デバッグ能力の高いプログラマとは，活動の反復が速く，かつ，欠陥を確実に修正するプログラマであると考えられる．逆に，能力の低いプログラマとは逆の特徴をもつプログラマである．一部のプログラマは特に興味深い特徴をもつ．典型的な例は，図2(g)の被験者Zである．活動の反復は速いが，プログラム変更に対してデバッグが不完全である回数の割合が高い．

4.2 モデルの定義

提案する方法では，各プログラマの特徴を抽出するために，観測された活動系列にプログラマモデルを適合させる．このモデルは，**4.1**で述べたプログラマ間の相違点を表現できるよう定義された．

図3のようにこのモデルでは，プログラマ活動系列をプログラマ状態C，X，V，M，CE，XEからなるマルコフ過程と仮定している．各時刻において，プログラマは6個の中の1個の状態にある．そして，1秒間隔で，ある確率に従って他の状態に遷移する．または，同一の状態に遷移する．図3の中の各矢印は1個の状態遷移を表し，矢印に付けられたラベルはその状態遷移の確率を表す．

観測されるプログラマ活動系列とプログラマ状態間の遷移との関係は次のとおりである．まず，最初のプログラム実行において，プログラマは状態Xにあり，作成したプログラムを起動する．(**4.1**で述べたように，最初のコンパイルは例外的であるので，モデルに含めないことにした．)何らかのプログラムの動作が観察されると，プログラマがそれが正しいかどうかを確認し，状態Vに遷移する．その際，もし故障が発見されれば，プログラマは状態VからMに遷移し，プログラム変更を開始する．発見されなければ，状態VからXに戻り，プログラムを再起動したり別の入力や操作を行う．状態Mにおいてプログラム変更を終了すると，プログラマは状態CかCEのどちらかに遷移する．状態MからCへの遷移は，発見された欠陥が完全に修正された後にコンパイルが開始されたことを意味する．状態Cにおいてコンパイルが終了すると，プログラマは状態Xに遷移し，プログラム実行を繰り返す．一方，状態MからCEへの遷移は，デバッグが不完全なままコンパイルが開始されたことを意味する．状態CEにおいてコンパイルが終了すると，プログラマは状態XEに遷移し，プログラム実行を開始する．しかし，すぐに同一の故障が発見されるので，直ちに状態Mに移動し，プログラム変更を再開する．

各状態遷移の確率は以下のように定義する．まず，状態CからXへの遷移と状態CEからXEへの遷移の確率は同一の定数であり，p_Cと表記する．状態XからVへの遷移と状態XEからMへの遷移の確率も同一の定数であり，p_Xと表記する．状態MからCへの遷移と状態MからCEへの遷移の確率は独立な定数であり，それぞれp_D，p_Eと表記する．状態VからMへの遷移は故障の発見を意味するので，この遷移確率は定数とするべきでない．そこで，時刻t秒までに状態VからMに遷移した回数をfとして，その遷移確率を次の関数により定義する．

$$p_V(f)=\alpha\cdot\max[\varepsilon, f_0-f] \qquad (1)$$

上式中の小正数$\varepsilon(10^{-6})$は，遷移確率が非正数になることを防ぐ役割を果たしている．定数f_0は最初のコンパイルの際に潜んでいた欠陥の個数である．つまり，f_0-fは時刻tにおいて残存する欠陥の個数であり，$p_V(f)$はその個数に比例すると仮定している．定数αは比例係数である．

4.3 モデルの制約

4.2のプログラマモデルは，実際のプログラマ活動系列の分析結果に基づいて定義されたが，**3.2**の実験と異なる条件のもとでは必ずしも妥当であるとは限らない．そこで，このモデルが妥当であり，提案する方法が有効に働くために，必要であると考えられる条件を以下

にまとめる．

（1） 故障を発見しても直ちに修正しようとせず，複数の故障を発見した後にまとめて修正しようとした場合に，本モデル，および，方法を適用してはならない．

（2） コーディングとデバッグが明確に分かれていない場合に，適用してはならない．例えば，仕様の一部だけが実現された後に，コンパイル，実行，修正，プログラムの拡張などが不規則に繰り返される場合がある．

（3） 独立性の高い複数のモジュールから構成される大規模プログラムを対象とした場合に，適用してはならない．そのような場合は，複数のプログラムを順番に，あるいは，交互にデバッグする作業となると考えられる．

（4） 仕様の誤解や設計の根本的な誤りに起因する大幅な変更があった場合に，適用してはならない．

（5） デバッグ環境の不備のために，プログラマに余計な負担がかかった場合に，適用してはならない．例えば，プログラマの希望するデバッガがインストールされていない場合である．

なお，デバッガなどのツールの使用・不使用については，プログラマが自由に選択する限りは，特に制約を設けない．有効なツールを選択できること，および，ツールの使用に習熟していることも，デバッグ能力の一部であると考えられるからである．

4.4 モデルのパラメータ

4.2 で述べたように，このプログラマモデルは6個のパラメータ $\lambda=(p_C, p_D, p_E, p_X, \alpha, f_0)$ をもつ．各パラメータの意味は次のとおりである．

p_C：コンパイルに対応する状態C，または，CEから抜け出る確率であり，1回のコンパイルの継続時間を逆数的に表す．具体的には，コンパイルが t 秒間継続する確率が $(1-p_C)^{t-1}p_C$（幾何分布）であるので，$1/p_C$ がコンパイルの平均継続時間となる．

p_X：プログラムの1回の操作に対応する状態X，または，XEから抜け出る確率であり，1回のプログラム操作の継続時間を逆数的に表す．具体的には，$1/p_X$ がプログラム操作の平均継続時間となる．プログラムの特性に依存する．

p_D，p_E：2個1組で，プログラム変更の継続時間と不完全なデバッグの割合の両方を表す．p_D+p_E は，プログラム変更に対応する状態Mから抜け出る確率であり，$1/(p_D+p_E)$ がプログラム変更の平均継続時間となる．$p_E/(p_D+p_E)$ はプログラム変更に対する不完全なデバッグの回数の割合である．

f_0：最初のコンパイルの際に潜む欠陥の個数である．

α：式(1)からわかるように，$f_0-f=1$ のときの状態VからMへの遷移確率である．つまり，プログラムに欠陥が1個だけ残っているときに1回のプログラム操作で故障が発見される確率であり，故障発見の容易さを表す．プログラマ能力に依存すると共に，仕様の難しさやプログラムの複雑さに依存すると考えられる．

各プログラマの特徴を抽出するために，ステップ4では，$\lambda=(p_C, p_D, p_E, p_X, \alpha, f_0)$ を，観測された活動系列に適合するよう推定する．但し，この λ の推定は容易ではない．観測される活動の種類よりも，プログラマ状態の種類が多いからである．例えば，コンパイルが観測されたときのプログラマ状態はC，またはCEであることがわかるが，どちらであるか特定できない．一般に，外部から観測できない状態を含むマルコフ過程は，隠れマルコフ過程（hidden Markov process）と呼ばれる．この種のパラメータの推定には，Baum-Welchアルゴリズムが応用できる[1],[9]．これは，モデルがその活動系列を生成するゆう度を基準として，それを最大化する λ を探索する方法（最ゆう推定法）である．方法の詳細は付録に示す．**3.2** の被験者に対して推定したパラメータを表2に挙げておく．なお，被験者Tの実験1については，発見された欠陥が少なすぎたので，パラメータが推定できなかった．

表2 推定したパラメータ

被験者	実験	p_C	p_D	p_E	p_X	α	f_0
F	1	.1875	.0013	.0018	.1182	.3419	2.08
	2	.1765	.0016	.0024	.0144	.3328	3.00
	3	.1867	.0013	.0017	.0940	.0816	7.00
K	2	.0816	.0007	.0004	.0942	.0342	4.12
M	2	.0169	.0008	.0004	.0338	.7750	1.15
S	1	.0613	.0044	.0050	.3322	.3536	2.41
	2	.2400	.0030	.0039	.0203	.4044	2.47
	4	.0594	.0015	.0020	.3356	.0995	6.00
T	1	—	—	—	—	—	—
	2	.3125	.0041	.0027	.1027	.4632	2.15
	3	.1429	.0169	.0085	.0860	.5708	1.68
	4	.2439	.0038	.0026	.0359	.0569	5.80
	5	.2500	.0014	.0011	.0585	.2840	3.52
Y	1	.2703	.0017	.0005	.0875	.0373	8.00
	2	.1750	.0017	.0005	.0149	.1665	6.00
Z	1	.0862	.0010	.0023	.0654	.0447	12.00
	2	.1610	.0008	.0032	.0242	.0501	19.95

5. 評価値の算出

5.1 評価尺度

4.4 で述べたパラメータを推定すると，いよいよプログラマの評価値が計算できるようになる．ステップ5では次の3種類をデバッグ能力の評価値として算出する．

r：プログラム変更に対してデバッグが不完全である回数の割合．$p_E/(p_D+p_E)$．

d：1プログラム変更に要する時間の平均．$1/(p_D+p_E)$．

D：1欠陥の修正に要する時間の平均．

1回のプログラム変更で1個の欠陥が修正される(つまり，$r=0$)とは限らないので，d と D は必ずしも同じでない．評価値 D は r と d の両方を考慮した値である．

以下では，D を p_C，p_D，p_E，p_X から算出する式の導出について述べる．まず，不完全なデバッグの割合は r であるので，デバッグの完了までに n 回のコンパイルを行う確率は $r^n(1-r)$(幾何分布)となる．従って，1個の欠陥当りのコンパイルとプログラム実行の平均回数はともに $r/(1-r)$ となる．1個の欠陥当りのプログラム変更の回数は，それより1回多いので，$1/(1-r)$ となる．1個の欠陥の修正に要する時間 D は，コンパイル・プログラム実行・プログラム変更に要する時間の和として，次のように求まる．

$$D=\left(\frac{r}{1-r}\right)\frac{1}{p_C}+\left(\frac{r}{1-r}\right)\frac{1}{p_X}+\left(\frac{1}{1-r}\right)\frac{1}{p_D+p_E}$$

$$=\frac{p_E}{p_D}\left(\frac{1}{p_C}+\frac{1}{p_X}\right)+\frac{1}{p_D}$$

5.2 評価実験

3.2 の被験者に対して計算した r，d，D の値を表3に示す．計算された値は **4.1** の分析とよく一致している．例えば，プログラマZは，不完全なデバッグの割合 r が7名中で最高であり，プログラム変更の継続時間 d が比較的短い．

以下では，計算した評価値とプログラマ能力との関係を評価するために，生産性の尺度 P と r，d，D のそれぞれとを比較する．ここでいう生産性の尺度 P とは共通仕様に対するプログラムの作成時間である．

但し，**3.2** で述べたように，被験者KとMに対しては，共通仕様に対する実験を中止したので，P を測定していない．両者とも大部分の作業を終えていたので，最初のコンパイルまでに要した時間から，P を推定することにした．具体的には，8時間以内に終了した別の実験(実験2)から，最初のコンパイルまでに要した時間と全体に要した時間との比を求め，この比が同じになるよう推定した．被験者Kの P は765分，Mは580分と推定した．

比較の結果，(推定値を含む) P に対して最も強い相関を示した評価値は D であった．図4に D と P との相関を示す．四角印はKとMの P の推定値と D の値

表3 評価値

被験者	実験	r	d (秒)	D (秒)
F	1	.574	325	773
	2	.600	248	631
	3	.558	332	760
	平均	.583	287	697
K	2	.378	879	1421
M	2	.333	852	1308
S	1	.530	107	246
	2	.563	144	337
	4	.572	286	692
	平均	.555	179	425
T	2	.400	147	248
	3	.333	39	62
	4	.405	158	269
	5	.432	393	696
	平均	.392	184	319
Y	1	.234	444	581
	2	.216	458	586
	平均	.225	451	583
Z	1	.702	304	1049
	2	.799	248	1266
	平均	.750	276	1157

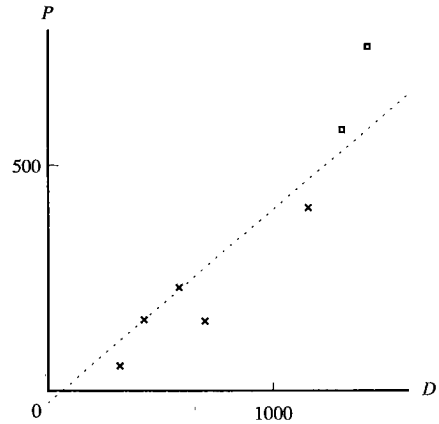

図4 D と P との相関
Fig. 4 Correlation between D and P.

を表している．十字印は他の5名の P と D の値を表している．相関係数は0.94であった．また，P と r との相関係数は−0.15であり，ほとんど相関が見られなかった．また，P と d との相関係数は0.90であり，比較的強い相関が見られたが，D よりは弱い相関であった．**5.1**で述べたように，D は r と d の両方を考慮した値であり，より正確にデバッグ能力を表していると言える．

6. むすび

プログラマのデバッグ能力をキーストロークから測定する方法を提案した．また，この方法で使用するプログラマモデルの詳細について述べた．提案した方法では，実際の作業の様子を監視するので，客観的かつ正確に能力を測定できる．また，キーストロークを監視するだけなので，プログラマにも測定者にも余分な負担をかけない．更に，測定の手順をツールによって自動的に実行でいるという利点をもつ．

また，実験から収集したデータを通して，プログラマモデルの妥当性，および方法の有効性を示した．被験者の数は7名だけであったが，図2に例を示したように詳細なデータが収集されている．**4.2**で示した制約のもとでは，1個の欠陥の修正に要する時間の推定値 D がプログラマのデバッグ能力をよく表すことがわかった．

今後の課題としては，方法の有効性を示すデータの更なる収集，**4.2**で示した制約の実際のソフトウェア開発における妥当性の検討，および，それらの制約を満たさない場合に対する方法の拡張などが挙げられる．

文献

(1) Baum L. E., Petrie T., Soules G., Weiss N. : "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains", Ann. Math. Statistic., **41**, pp. 164-172 (1970).

(2) Boehm B. W. : "Software engineering economics", IEEE Trans. Software Eng., **SE-10**, 1, pp. 4-21 (1984).

(3) Boehm B. W. : "Improving software productivity", IEEE Computer, **20**, 9, pp. 43-57 (1987).

(4) 藤田房之，高田義広，松本健一，鳥居宏次：“プログラマのテスト・デバッグ能力の自動計測環境”，信学技報，**SS93**-29 (1993).

(5) Gorla N., Benander A. C., Benander B. A. : "Debugging effort estimation using software metrics", IEEE Trans. Software Eng., **16**, 2, pp. 223-231 (1990).

(6) 石井威望，廣瀬通孝，小木哲朗：“プログラマの状態モデル—実験的検証”，情処学会第31回後期全大，3G-7, pp. 631-632 (1985).

(7) Matumoto K., Inoue K., Kudoh H., Sugiyama Y. and Torii K. : "Error life span and programmer performance", Proc. of International Computer Software and Applications Conference, pp. 259-265 (1987).

(8) 三輪和久，櫻井桂一，岡田 稔，岩田 晃：“初心者のプログラミング行動時系列分析”，信学技報，**ET93**-47 (1993).

(9) Rabiner L. R. and Juang B. H. : "An introduction to hidden Markov models", IEEE ASSP Magazine, 1, pp. 4-15 (1986).

(10) Sackman H., Erikson W. J., Grant E. E. : "Exploratory experimental studies comparing online and offline programming performance", Commun. ACM, **11**, pp. 3-11 (1968).

(11) 吉田敬一，平井利明：“プログラマ適性検査の信頼性について(第2報)”，情処学ソフトウェア工学研報，25-1, pp. 1-7 (1982).

付録

活動系列 $Y=(Y_0,\cdots,Y_T)$ に対して，$\Pr(Y\mid\lambda)$ を最大化するようにパラメータ $\lambda=(p_C,p_D,p_E,p_X,\alpha,f_0)$ を推定する方法について述べる．但し，T は活動系列の長さ，Y_t は時刻 t におけるプログラマの活動 ($Y_t\in$ {コンパイル，プログラム実行，プログラム変更})，$\Pr(Y\mid\lambda)$ はパラメータ λ をもつモデルが活動系列 Y を生成する確率(ゆう度)を表す．

まず，$X=(X_0,\cdots,X_T)$ を Y に対応するプログラマ状態の時系列と定義する ($X_t\in$ {C, X, V, M, CE, XE})．1個の Y に対しては多くの可能な X が存在するので，$\Pr(Y\mid\lambda)$ は $\sum_X\Pr(Y,X\mid\lambda)$ と展開できる．ここで，$\Pr(Y,X\mid\lambda)$ とは，モデルが X のように状態遷移し，かつ，活動系列 Y を生成する確率である．この $\sum_X\Pr(Y,X\mid\lambda)$ を直接的に最大化することは難しいので，補助関数

$$Q(\lambda,\lambda')=\sum_X\Pr(Y,X\mid\lambda)\log\Pr(Y,X\mid\lambda') \quad (A\cdot1)$$

を導入する．この関数については，$Q(\lambda,\lambda')\geqq Q(\lambda,\lambda)$ であれば $\log\Pr(Y\mid\lambda')\geqq\log\Pr(Y\mid\lambda)$ であると言う性質が証明できる．この性質により，λ に適当な初期値を与えて次の反復を繰り返せば，λ が求める値に収束することになる．つまり，$Q(\lambda,\lambda')$ を λ' に関して最大化する $\hat{\lambda}$ を求め，新たに λ と置き換えることを繰り返せばよい．以降，この $\hat{\lambda}$ を $\partial Q(\lambda,\lambda')/\partial\lambda'=0$ の解として求める方法を述べる．

まず，$\Pr(Y,X\mid\lambda)=\prod_t\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')$ より，式(A・1)を

$$Q(\lambda,\lambda')=\sum_X \Pr(Y,X\mid\lambda)\log\Big\{\prod_t \Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')\Big\}$$
$$=\sum_t\sum_X \Pr(Y,X\mid\lambda)\log\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda') \quad (A\cdot2)$$

と展開する．もちろん，$\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')$ は，時刻 t において状態 X_t から X_{t+1} に遷移する確率である．$Q(\lambda,\lambda')$ はそれらの確率の対数の加重和であることがわかる．時刻 t までの状態系列の中で状態 V から M の遷移の回数を $f(X_0,\cdots,X_t)$ と表記すると，各確率は更に，

$$\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')=\begin{cases}\rho_C' & \text{if } (X_t,X_{t+1})=(\mathrm{C},\mathrm{X})\text{or}(\mathrm{CE},\mathrm{XE})\\ 1-\rho_C' & \text{if } (X_t,X_{t+1})=(\mathrm{C},\mathrm{C})\text{or}(\mathrm{CE},\mathrm{CE})\\ p_X' & \text{if } (X_t,X_{t+1})=(\mathrm{X},\mathrm{V})\text{or}(\mathrm{XE},\mathrm{D})\\ 1-p_X' & \text{if } (X_t,X_{t+1})=(\mathrm{X},\mathrm{X})\text{or}(\mathrm{XE},\mathrm{XE})\\ p_D' & \text{if } (X_t,X_{t+1})=(\mathrm{M},\mathrm{C})\\ p_E' & \text{if } (X_t,X_{t+1})=(\mathrm{M},\mathrm{CE})\\ 1-p_D'-p_E' & \text{if } (X_t,X_{t+1})=(\mathrm{M},\mathrm{M})\\ p_V'(f(X_0,\cdots,X_t)) & \text{if } (X_t,X_{t+1})=(\mathrm{V},\mathrm{M})\\ 1-p_V'(f(X_0,\cdots,X_t)) & \text{if } (X_t,X_{t+1})=(\mathrm{V},\mathrm{X})\\ 0 & \text{otherwise}\end{cases} \quad (A\cdot3)$$

と展開できる．これを(A・2)に代入すると，

$$\begin{aligned}Q(\lambda,\lambda')=&\sum_t\{\xi_t(\mathrm{C},\mathrm{X})+\xi_t(\mathrm{CE},\mathrm{XE})\}\log p_C'\\&+\sum_t\{\xi_t(\mathrm{C},\mathrm{C})+\xi_t(\mathrm{CE},\mathrm{CE})\}\log(1-p_C')\\&+\sum_t\{\xi_t(\mathrm{X},\mathrm{V})+\xi_t(\mathrm{XE},\mathrm{M})\}\log p_X'\\&+\sum_t\{\xi_t(\mathrm{X},\mathrm{X})+\xi_t(\mathrm{XE},\mathrm{XE})\}\log(1-p_X')\\&+\sum_t\xi_t(\mathrm{M},\mathrm{C})\log p_D'+\sum_t\xi_t(\mathrm{M},\mathrm{CE})\log p_E'\\&+\sum_t\xi_t(\mathrm{M},\mathrm{M})\log(1-p_D'-p_E')\\&+\sum_t\sum_f\zeta_{t,f}(\mathrm{V},\mathrm{M})\log p_V'(f)\\&+\sum_t\sum_f\zeta_{t,f}(\mathrm{V},\mathrm{X})\log(1-p_V'(f))\end{aligned} \quad (A\cdot4)$$

が得られる．但し，

$\xi_t(s_0,s_1)=\Pr(Y,X_{t+1}=s_1,X_t=s_0\mid\lambda)$

$\zeta_{t,f}(s_0,s_1)=\Pr(Y,X_{t+1}=s_1,X_t=s_0,f(X_0,\cdots,X_t)=f\mid\lambda)$

である．動的計画法により Y と λ から $\xi_t(s_0,s_1)$ と $\zeta_{t,f}(s_0,s_1)$ を算出することは難しくない．$\zeta_{t,f}(\mathrm{X},\mathrm{V})$ を例にとる．

$\alpha_{t,f}(s_0)=\Pr(Y_0,\cdots,Y_t,X_t=s_0,f(X_0,\cdots,X_t)=f\mid\lambda)$

$\beta_{t,f}(s_1)=\Pr(Y_{t+1},\cdots,Y_T\mid X_t=s_1,f(X_o,\cdots,X_t)=f,\lambda)$

$\gamma_f(s_0,s_1)=\Pr(X_{t+1}=s_1\mid X_t=s_0,f(X_0,\cdots,X_t)=f,\lambda)$

と定義すれば，$\zeta_{t,f}(\mathrm{X},\mathrm{V})=\alpha_{t,f}(X)\cdot\gamma_f(\mathrm{X},\mathrm{V})\cdot\beta_{t+1,f}(\mathrm{V})$ となる．ここで，$\gamma_f(s_0,s_1)$ は，$(X_t,X_{t+1})=(s_0,s_1)$ かつ $f(X_0,\cdots,X_t)=f$ のときの $\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda)$ であるので，(A・3)からすぐに計算できる．$\alpha_{t,f}(s)$ と $\beta_{t,f}(s)$ は次の漸化式により計算できる．

$$\alpha_{t+1,f}(s_1)=\sum_{s0}\Big[\begin{matrix}\text{if}(s_0,s_1)=(\mathrm{V},\mathrm{M})\ \text{then}\ \alpha_{t,f-1}(s_0)\cdot\gamma_{f-1}(s_0,s_1)\\ \text{else}\ \alpha_{t,f}(s_0)\cdot\gamma_f(s_0,s_1)\end{matrix}\Big]$$

$$\beta_{t,f}(s_0)=\sum_{s1}\Big[\begin{matrix}\text{if}(s_0,s_1)=(\mathrm{V},\mathrm{M})\ \text{then}\ \gamma_f(s_0,s_1)\cdot\beta_{t+1,f+1}(s_1)\\ \text{else}\ \gamma_f(s_0,s_1)\cdot\beta_{t+1,f}(s_1)\end{matrix}\Big]$$

式(A・4)から，求める $\hat{\lambda}=(\hat{p}_C,\hat{p}_D,\hat{p}_E,\hat{p}_X,\hat{a},\hat{f}_0)$ は連立方程式 $\partial Q_1(p_C')/\partial p_C'=\partial Q_2(p_X')/\partial p_X'=\partial Q_3(p_D',p_E')/\partial p_D'=\partial Q_3(p_D',p_E')/\partial p_E'=\partial Q_4(a',f_0')/\partial a'=\partial Q_4(a',f_0')/\partial f_0'=0$ の解であることがわかる．但し，

$$Q_1(p_C')=\sum_t\{\xi_t(\mathrm{X},\mathrm{C})+\xi_t(\mathrm{XE},\mathrm{CE})\}\log p_C'+\sum_t\{\xi_t(\mathrm{C},\mathrm{C})+\xi_t(\mathrm{CE},\mathrm{CE})\}\log(1-p_C') \quad (A\cdot5)$$

$$Q_2(p_X')=\sum_t\{\xi_t(\mathrm{V},\mathrm{X})+\xi_t(\mathrm{M},\mathrm{XE})\}\log p_X'+\sum_t\{\xi_t(\mathrm{X},\mathrm{X})+\xi_t(\mathrm{XE},\mathrm{XE})\}\log(1-p_X') \quad (A\cdot6)$$

$$Q_3(p_D',p_E')=\sum_t\xi_t(\mathrm{C},\mathrm{M})\log p_D'+\sum_t\xi_t(\mathrm{CE},\mathrm{M})\log p_E'+\sum_t\xi_t(\mathrm{M},\mathrm{M})\log(1-p_D'-p_E') \quad (A\cdot7)$$

$$Q_4(a',f_0')=\sum_t\sum_f\zeta_{t,f}(\mathrm{M},\mathrm{V})\log p_V'(f)=\sum_t\sum_f\zeta_{t,f}(\mathrm{X},\mathrm{V})\log(1-p_V'(f)) \quad (A\cdot8)$$

である．式(A・5)，(A・6)，(A・7)から p_C'，p_D'，p_E'，p_X' に対する解は

$$Q(\lambda,\lambda')=\sum_X \Pr(Y,X\mid\lambda)\log\Big\{\prod_t \Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')\Big\}$$
$$=\sum_t\sum_X \Pr(Y,X\mid\lambda)\log\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda') \quad (A\cdot2)$$

と展開する．もちろん，$\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')$ は，時刻 t において状態 X_t から X_{t+1} に遷移する確率である．$Q(\lambda,\lambda')$ はそれらの確率の対数の加重和であることがわかる．時刻 t までの状態系列の中で状態 V から M の遷移の回数を $f(X_0,\cdots,X_t)$ と表記すると，各確率は更に，

$$\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda')=\begin{cases}\rho_C' & \text{if } (X_t,X_{t+1})=(\mathrm{C},\mathrm{X})\text{or}(\mathrm{CE},\mathrm{XE})\\ 1-\rho_C' & \text{if } (X_t,X_{t+1})=(\mathrm{C},\mathrm{C})\text{or}(\mathrm{CE},\mathrm{CE})\\ p_X' & \text{if } (X_t,X_{t+1})=(\mathrm{X},\mathrm{V})\text{or}(\mathrm{XE},\mathrm{D})\\ 1-p_X' & \text{if } (X_t,X_{t+1})=(\mathrm{X},\mathrm{X})\text{or}(\mathrm{XE},\mathrm{XE})\\ p_D' & \text{if } (X_t,X_{t+1})=(\mathrm{M},\mathrm{C})\\ p_E' & \text{if } (X_t,X_{t+1})=(\mathrm{M},\mathrm{CE})\\ 1-p_D'-p_E' & \text{if } (X_t,X_{t+1})=(\mathrm{M},\mathrm{M})\\ p_V'(f(X_0,\cdots,X_t)) & \text{if } (X_t,X_{t+1})=(\mathrm{V},\mathrm{M})\\ 1-p_V'(f(X_0,\cdots,X_t)) & \text{if } (X_t,X_{t+1})=(\mathrm{V},\mathrm{X})\\ 0 & \text{otherwise}\end{cases} \quad (A\cdot3)$$

と展開できる．これを(A・2)に代入すると，

$$\begin{aligned}Q(\lambda,\lambda')=&\sum_t\{\xi_t(\mathrm{C},\mathrm{X})+\xi_t(\mathrm{CE},\mathrm{XE})\}\log p_C'\\&+\sum_t\{\xi_t(\mathrm{C},\mathrm{C})+\xi_t(\mathrm{CE},\mathrm{CE})\}\log(1-p_C')\\&+\sum_t\{\xi_t(\mathrm{X},\mathrm{V})+\xi_t(\mathrm{XE},\mathrm{M})\}\log p_X'\\&+\sum_t\{\xi_t(\mathrm{X},\mathrm{X})+\xi_t(\mathrm{XE},\mathrm{XE})\}\log(1-p_X')\\&+\sum_t\xi_t(\mathrm{M},\mathrm{C})\log p_D'+\sum_t\xi_t(\mathrm{M},\mathrm{CE})\log p_E'\\&+\sum_t\xi_t(\mathrm{M},\mathrm{M})\log(1-p_D'-p_E')\\&+\sum_t\sum_f\zeta_{t,f}(\mathrm{V},\mathrm{M})\log p_V'(f)\\&+\sum_t\sum_f\zeta_{t,f}(\mathrm{V},\mathrm{X})\log(1-p_V'(f))\end{aligned} \quad (A\cdot4)$$

が得られる．但し，

$\xi_t(s_0,s_1)=\Pr(Y,X_{t+1}=s_1,X_t=s_0\mid\lambda)$

$\zeta_{t,f}(s_0,s_1)=\Pr(Y,X_{t+1}=s_1,X_t=s_0,f(X_0,\cdots,X_t)=f\mid\lambda)$

である．動的計画法により Y と λ から $\xi_t(s_0,s_1)$ と $\zeta_{t,f}(s_0,s_1)$ を算出することは難しくない．$\zeta_{t,f}(\mathrm{X},\mathrm{V})$ を例にとる．

$\alpha_{t,f}(s_0)=\Pr(Y_0,\cdots,Y_t,X_t=s_0,f(X_0,\cdots,X_t)=f\mid\lambda)$

$\beta_{t,f}(s_1)=\Pr(Y_{t+1},\cdots,Y_T\mid X_t=s_1,f(X_o,\cdots,X_t)=f,\lambda)$

$\gamma_f(s_0,s_1)=\Pr(X_{t+1}=s_1\mid X_t=s_0,f(X_0,\cdots,X_t)=f,\lambda)$

と定義すれば，$\zeta_{t,f}(\mathrm{X},\mathrm{V})=\alpha_{t,f}(X)\cdot\gamma_f(\mathrm{X},\mathrm{V})\cdot\beta_{t+1,f}(\mathrm{V})$ となる．ここで，$\gamma_f(s_0,s_1)$ は，$(X_t,X_{t+1})=(s_0,s_1)$ かつ $f(X_0,\cdots,X_t)=f$ のときの $\Pr(X_{t+1}\mid X_0,\cdots,X_t,\lambda)$ であるので，(A・3)からすぐに計算できる．$\alpha_{t,f}(s)$ と $\beta_{t,f}(s)$ は次の漸化式により計算できる．

$$\alpha_{t+1,f}(s_1)=\sum_{s0}\Big[\begin{matrix}\text{if}(s_0,s_1)=(\mathrm{V},\mathrm{M})\ \text{then}\ \alpha_{t,f-1}(s_0)\cdot\gamma_{f-1}(s_0,s_1)\\ \text{else}\ \alpha_{t,f}(s_0)\cdot\gamma_f(s_0,s_1)\end{matrix}\Big]$$

$$\beta_{t,f}(s_0)=\sum_{s1}\Big[\begin{matrix}\text{if}(s_0,s_1)=(\mathrm{V},\mathrm{M})\ \text{then}\ \gamma_f(s_0,s_1)\cdot\beta_{t+1,f+1}(s_1)\\ \text{else}\ \gamma_f(s_0,s_1)\cdot\beta_{t+1,f}(s_1)\end{matrix}\Big]$$

式(A・4)から，求める $\hat{\lambda}=(\hat{p}_C,\hat{p}_D,\hat{p}_E,\hat{p}_X,\hat{a},\hat{f}_0)$ は連立方程式 $\partial Q_1(p_C')/\partial p_C'=\partial Q_2(p_X')/\partial p_X'=\partial Q_3(p_D',p_E')/\partial p_D'=\partial Q_3(p_D',p_E')/\partial p_E'=\partial Q_4(a',f_0')/\partial a'=\partial Q_4(a',f_0')/\partial f_0'=0$ の解であることがわかる．但し，

$$Q_1(p_C')=\sum_t\{\xi_t(\mathrm{X},\mathrm{C})+\xi_t(\mathrm{XE},\mathrm{CE})\}\log p_C'+\sum_t\{\xi_t(\mathrm{C},\mathrm{C})+\xi_t(\mathrm{CE},\mathrm{CE})\}\log(1-p_C') \quad (A\cdot5)$$

$$Q_2(p_X')=\sum_t\{\xi_t(\mathrm{V},\mathrm{X})+\xi_t(\mathrm{M},\mathrm{XE})\}\log p_X'+\sum_t\{\xi_t(\mathrm{X},\mathrm{X})+\xi_t(\mathrm{XE},\mathrm{XE})\}\log(1-p_X') \quad (A\cdot6)$$

$$Q_3(p_D',p_E')=\sum_t\xi_t(\mathrm{C},\mathrm{M})\log p_D'+\sum_t\xi_t(\mathrm{CE},\mathrm{M})\log p_E'+\sum_t\xi_t(\mathrm{M},\mathrm{M})\log(1-p_D'-p_E') \quad (A\cdot7)$$

$$Q_4(a',f_0')=\sum_t\sum_f\zeta_{t,f}(\mathrm{M},\mathrm{V})\log p_V'(f)=\sum_t\sum_f\zeta_{t,f}(\mathrm{X},\mathrm{V})\log(1-p_V'(f)) \quad (A\cdot8)$$

である．式(A・5)，(A・6)，(A・7)から p_C'，p_D'，p_E'，p_X' に対する解は

$$\hat{p}_C=\frac{\sum_t\{\xi_t(\mathrm{X},\mathrm{C})+\xi_t(\mathrm{XE},\mathrm{CE})\}}{\sum_t\{\xi_t(\mathrm{X},\mathrm{C})+\xi_t(\mathrm{XE},\mathrm{CE})+\xi_t(\mathrm{C},\mathrm{C})+\xi_t(\mathrm{CE},\mathrm{CE})\}}$$

$$\hat{p}_D=\frac{\sum_t\xi_t(\mathrm{C},\mathrm{M})}{\sum_t\{\xi_t(\mathrm{C},\mathrm{M})+\xi_t(\mathrm{CE},\mathrm{M})+\xi_t(\mathrm{M},\mathrm{M})\}}$$

$$\hat{p}_E=\frac{\sum_t\xi_t(\mathrm{CE},\mathrm{M})}{\sum_t\{\xi_t(\mathrm{C},\mathrm{M})+\xi_t(\mathrm{CE},\mathrm{M})+\xi_t(\mathrm{M},\mathrm{M})\}}$$

$$\hat{p}_X=\frac{\sum_t\{\xi_t(\mathrm{V},\mathrm{X})+\xi_t(\mathrm{M},\mathrm{XE})\}}{\sum_t\{\xi_t(\mathrm{V},\mathrm{X})+\xi_t(\mathrm{M},\mathrm{XE})+\xi_t(\mathrm{X},\mathrm{X})+\xi_t(\mathrm{XE},\mathrm{XE})\}}$$

と求まる．残りの α' と f'_0 に対する求解はやや複雑である．$Q_4(\alpha',f'_0)$ が関数 $p_{V'}(f)$ を含むからである．この問題は最急こう配法により解くことができる．つまり，点 (α',f'_0) を傾斜方向の方向 $(\partial Q_4(\alpha',f'_0)/\partial\alpha',\partial Q_4(\alpha',f'_0)/\partial f'_0)$ に少し更新することを，その点が極大点に達するまで繰り返せばよい．

（平成5年11月30日受付，6年4月14日再受付）

高田　義広

平1阪大・基礎工・情報卒．平6同大大学院博士課程了．同年奈良先端科学技術大学院大学・助手．工博．ソフトウェアの信頼性予測，プログラマの能力評価の研究に従事．情報処理学会，日本音響学会各会員．

鳥居　宏次

昭37阪大・工・通信卒．昭42同大大学院博士課程了．同年電気試験所（現電子技術総会研究所）入所．昭50ソフトウェア部言語処理研究室室長．昭59阪大・基礎工・情報教授．平4奈良先端科学技術大学院大学・教授．工博．ソフトウェア工学の研究に従事．情報処理学会，日本ソフトウェア学会，人工知能学会，ACM，IEEE各会員．