バグ混入時のプログラミング行動の特徴を検出するツールの試作

柳 正純† †††, 門田 暁人†, 高田 義広†, 鳥居 宏次† ††

† 奈良先端科学技術大学院大学 情報科学研究科
†† 大阪大学 基礎工学部 情報工学科
††† 三菱電機関西コンピュータシステム (株)

〒 630-01 奈良県 生駒市 高山町 8916 番地の 5
奈良先端科学技術大学院大学 情報科学研究科
(高田 義広 takada@is.aist-nara.ac.jp)

あらまし　我々は, プログラマのキーストロークを監視・分析するシステムにより, 人手を介さずに, バグの混入を警告する方式について検討している. ところが, キーストロークより得られるデータの上に, バグの混入に関連する行動の特徴が現れるかどうかは, 従来, ほとんど調査されていない. そこで, そのようなキーストロークデータを収集し, プログラマの行動の特徴を検出するためのシステムを設計し試作した. 試作したシステムを適用した例から, 本システムがデータの収集に有効に働くことがわかった. また, バグ混入時の特徴をいくつか発見し, 上述のような警告方式が実現可能であることを示唆するデータが得られた.

和文キーワード　バグ, プログラミング行動, 分析システム, ソフトウェア品質

A Tool Detecting Patterns of Programmers' Behavior When They Likely Inject Bugs

Masazumi YANAGI † †††, Akito MONDEN †, Yoshihiro TAKADA †, Koji TORII † ††

† Graduate School of Information Science, Nara Institute of Science and Technology
†† Faculty of Engineering Science, Osaka University
††† Mitsubishi Electric Computer Systems (Kansai), Co. Ltd.

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma-Shi, Nara 630-01 (c/o Takada, takada@is.aist-nara.ac.jp)

Abstract　We are studying on a system that automatically monitors and analyzes key strokes of programmers and then warns them of injecting bugs. So far, there have been almost no research reports on features of programmers' behavior when they inject bugs. To collect data of key strokes and analyze programmers' behaviour, we have designed and partially implemented an analysis system. Through experiments of applying the system, we have found that the system effectively works and that it is possible to realize the warning system.

英文 key words　bug, programming behavior, analysis system, software quality

1　はじめに

ソフトウェアの開発コストの削減, および, 品質の向上のためには, バグの早期発見が重要であることが指摘されている [1]. 従って, テストよりもレビュー, レビューよりもそれ以前の工程でバグを発見することが有効であると言える. 従って, もし, コーディング中のプログラマの行動を観察することにより, バグを混入させそうな時点で直ちにプログラマに警告することが可能であれば, 非常に有効であると考えられる.

但し, 第 3 者がプログラマの行動を観察することは, 現実的でない. 多くの労力を要するし, プログラマに心理的な負担をかけるためである. そこで, 我々は, コーディング時のプログラマのキーストロークを自動的に監視・分析するシステムにより, 人手を介さずに, バグの混入を警告する方式について検討している.

このようなシステムを実現するためには, キーストロークの監視・分析によって得られる行動データの上で, バグの混入に強く関連する特徴を発見する必要がある. ところが, そのようなバグ混入時の特徴については, 従来, ほとんど報告がない. バグの原因を分析する研究は, 数多く報告されている [2, 3, 6]. 但し, それらの研究のほとんどでは, プログラマの書いた報告書や, プログラマへのインタビューなどに依存してデータを収集していた. また, バグの種類と原因との関係を分析しているが, バグ混入時のプログラマの行動については報告していなかった. また, キーストロークに着目して, バグを発見し修正する時の行動を分析する研究も報告されている [4]. 但し, この研究では, バグが混入する時の行動を対象としていなかった.

そこで, 我々は, 次の方針に従って, システムを段階的に開発することを計画した. 第 1 に, システムの一部として, プログラマのキーストロークを記録する部分を試作する. 第 2 に, 試作したキーストローク記録部を用いて実際にデータを収集し, 人手で分析することによって, キーストロークだけから検出できるようなバグ混入時の行動の特徴 (または, そのような特徴の候補) を見つける. この段階では, 分析を人手に頼るため, 大量のデータを分析することはできない. 第 3 に, 第 2 段階で見つけた行動の特徴を自動的に検出する部分を試作する. つまり, キーストローク記録部から送られてくるデータから, バグ混入時の特徴 (または, その候補) を検出する部分を試作する. この段階で, システムの原形が作成されることになる. 第 4 に, システムを利用して, 特徴の出現に関するデータをできるだけ多く収集する. そして, 特徴の出現とバグの混入との間の関係を定量的に評価する. また, 第 2 段階で見落としていたようなバグ混入時の特徴を探究する. この段階で, 上述の警告方式の実現可能性が明らかになると思われる. 第 5 段階以降では, プログラマに警告を発する部分などを加えて, 警告システムを完成させる. そして, 適用実験を繰り返すことによって, 警告システムの有効性を評価する.

現在, 我々は, 上述の方針に従って, 第 4 段階を始めた位置にあると考えている. つまり, システムの原型を作成し, 特徴の出現に関するデータを収集している. 本報告では, このシステムの設計・実装について述べ, そして, 適用例を通して, システムがデータの収集に有効に働いていることを示す.

以降, 2 章では, システム全体の設計の概要を説明する. 3 章では, キーストローク記録部の機構を示す. 4 章では, キーストローク記録部を利用して発見した, バグ混入時の行動の特徴について述べる. 5 章,6 章では, 4 章で述べるような行動の特徴をキーストロークから自動的に検出する部分について説明する. 7 章では, 実際にシステムを適用した例を示す. 8 章は, まとめと今後の課題とである.

なお, 我々は, 文献 [5] において, 第 2 段階までの中間成果を既に報告している. 本報告は, その拡張版に相当する.

2　設計の概要

システムは, 図 1に示すように, キーストローク記録部, 行動分析部, 特徴検出部, 警告部から構成される.

キーストローク記録部とは, プログラマがキーを打つ行動を全て記録する部分である (3 章). 各キーストロークについて, キーのコードと秒単位の時刻が記録される. この部分は, 後述するように, マルチウィンドウ環境に対応しており, マウスによる入力も, キーストロークと同様に記録する.

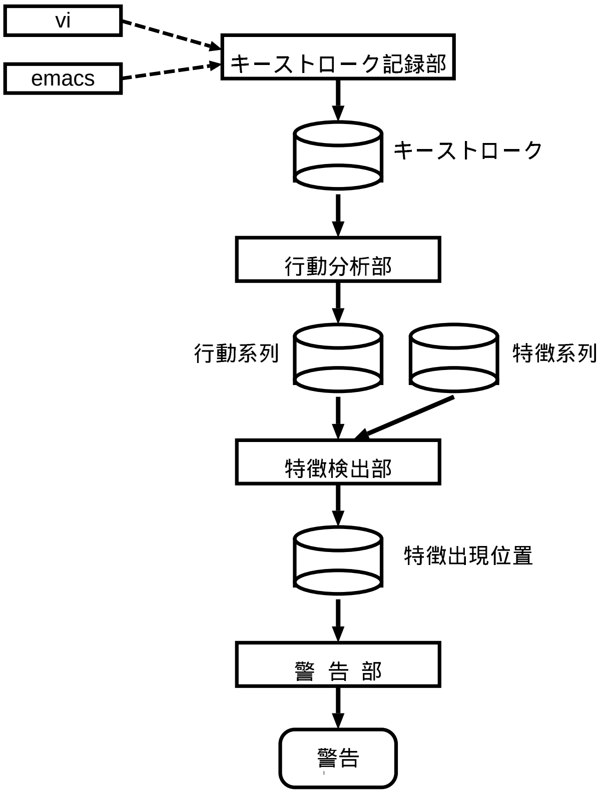

図 1: システムの構成

行動分析部とは, キーストロークデータを分析して, コーディング時のプログラマの行動を実時間で記録する部分である (5 章). ソースプログラム中の文字列の挿入, 変更, 削除, カーソルの移動などのそれぞれの行動について, その開始時刻, 終了時刻, 位置などのデータが記録される. 記録されるこのようなデータを行動系列と言う. プログラマが使用するテキストエディタは, その種類にコマンドの形式が異なる. そこで, 行動分析部は, 各エディタについて個別に対応する部分を含んでいる. 現在は, vi や emacs などのエディタに対応している. なお, キーストロークデータは, キーストローク記録部から, オンライン ·オフラインのどちらでも入力することが可能である.

特徴検出部とは, 行動系列の上に現れる特徴を実時間で検出する部分である (6 章). この部分の入力は, 行動系列と, 検出するべき行動の特徴の形式的な記述とである. 出力は, 検出した特徴の種類と時刻とである. 検出するべき特徴を外部から入力する方式を採っているので, 検出するべき特徴が変わっても柔軟に対応できる. なお, 行動系列は, 行動分析部から, オンライン ·オフラインのどちらでも入力することが可能である.

警告部とは, 特徴検出部の出力を入力として, プログラマに実時間で警告を発する部分である.

以上の設計に基づく実装の状況は, 次の通りである. まず, バグ混入時の行動の特徴を調べるために, キーストローク記録部だけを試作した. そして, 試作した部分を利用してデータを収集し, 人手で分析することによって, 4 章で述べるようなバグ混入時の特徴を発見した. 次に, 発見した特徴に基づいて, 行動系列の形式と特徴の記述の形式とを決定し, 行動分析部と特徴検出部とを設計した. そして, 行動分析部を試作し, 特徴検出部を部分的に試作した. 警告部は, 実装していない.

3 キーストローク記録部

キーストローク記録部とは, 2 章で述べたように, プログラマがキーを打つ行動を記録する部分である. 記録したキーストロークデータは, 行動分析部においてプログラマの行動を特定するために使用される. また, この記録部はマルチウィンドウ環境に対応しており, 複数のウィンドウに対するキー入力を同時に記録する. そのような環境においては, マウス操作による文字列の貼り込みも一般的に行なわれる. そのような文字列の貼り込みも, 同様に記録する.

具体的な記録の方式は次の通りである. マルチウィンドウ環境では, 端末エミュレータ (xterm, kterm) と呼ばれるウィンドウが開かれる. そして, その中でテキストエディタが起動され, ソースプログラムが編集される. そこで, それらの端末エミュレータのプログラムを改造して, キーストロークの記録に必要な機構を埋め込んでおく. 具体的には, キーが打たれる度にキーストローク記録部へデータを送信する部分を埋め込む. この送信部は, マウスによる貼り込みの際にも同様にデータを送信する. この記録部は, そのようなデータを各ウィンドウから受信し記録する. 埋め込まれる部分は, 非常に小さなプログラムであるので, 端末エミュレータの性能には影響しない.

端末エミュレータにデータ送信部を埋め込む代わりに, テキストエディタにデータ送信部を埋め込む方式も考えられる. そのような方式では,

表 1：バグ混入時の特徴

	画面やカーソル	バグ	プログラマ
a	1～20行程度の部分がプログラム中の別の場所からコピーされてきた後，その中の数箇所に変更が加わる．	コピーされた部分のどこかに誤りが残る．	ある部分を書こうとした時，その部分と似た部分がプログラム中にあったので，コピーして部分的に変更しようとした．ところが，変更し忘れたり，変更し間違えた．
b	1行が入力されている途中で，入力が中断され，別の場所へカーソルが移動する．そして，別の部分が変更された後，カーソルが元の位置に戻り，入力が再開される．	再開後に入力された部分に誤りが入る．	思考錯誤しながら入力していた途中，入力していた部分に関係する何らかの作業に気付き，それを割り込ませた．そのために，元の作業に対する注意が散慢になった．
c	同じ行の同じ箇所が何度も変更される．	最後の変更が誤りになる．	長時間，思考錯誤していたが，考えが整理できなかった．
d	1行が入力された後に，その行の1箇所が変更される．その直後に，カーソルが大きく移動し，別の作業が始まる．	初めに入力された行の中で変更箇所より後ろの部分に誤りが残る．	行の入力の直後に，その行に誤りを入れたことに気付き修正した．ところが，気付いた以外にも誤りが入っていた．
e	1行の入力に1分以上かかる．	その行のどこかに誤りが入る．	ある部分について，思考錯誤しながら入力していたが，考えが整理できなかった．
f	元々，誤りの入っていた行が，別の場所にコピーされる．	コピーされた部分に誤りも写される．	コピーする元の部分にあった誤りを見過ごした．

エディタに入力されるコマンドを直接的に記録することができるので，キーストロークデータよりも集約されたデータを収集することが可能になる．但し，プログラマが使用するエディタの種類は多いので，このような方式を採ると，多くのエディタのプログラムを変更する必要があり，システムの保守において問題が生ずる．そこで，より種類の少ない端末エミュレータにデータ送信部を埋め込む方式を採っている．

4　バグ混入時の特徴

前章のキーストローク記録部を使用してデータを収集し，分析することによって，バグ混入時の行動の特徴を見つける実験を行なった[5]．結果として6個の特徴を発見することができた．

4.1　実験手順

ステップ 1 (データ収集)．被験者に小規模なプログラムを開発(設計からテストまで)してもらい，その間のキーストロークを全て記録する．そして，そのデータから，入力されたエディタコマンドを全て検出する．
ステップ 2 (バグの位置の特定)．テスト中に行なわれたソースプログラムの変更は，全てバグの修正であると仮定する．そして，収集したデータを分析して，テスト中に変更された箇所を全て特定する．それらの修正箇所を，バグの位置と見なす．
ステップ 3 (バグの混入時刻の特定)．各バグについて，そのバグを含む部分が作成された時刻を特定する．その時刻を，バグの混入時刻と見なす．
ステップ 4 (特徴の抽出)．各バグについて，混入時刻の前後の画面の変化やカーソルの動きを，収集したデータより再現する．それを観察することにより，特徴的な行動を発見する．

4.2　実験条件

被験者は，X，Y，Zの3名である．3名とも8年以上のプログラミングの経験があった．与えたプログラム仕様は，被験者によって異なる．プログラムはC言語で書かれた．エディタはviが使用された．作成されたプログラムの規模は，200～300行であった．設計からテストまでの時間は，2～4時間であった．

表 2: 特徴の出現頻度

特徴	X	Y	Z	計
a	1	9	1	11
b	2	0	1	3
c	0	2	1	3
d	0	2	0	2
e	2	0	0	2
f	1	3	1	5
特徴を認めず	6	6	4	16
合計	12	22	8	42

4.3 実験結果

設計からテストまでの間のキーストロークの総数は, X が 11,497 回, Y が 18,986 回, Z が 20,638 回であった. テスト中に修正されたバグの数は, X が 12 個, Y が 21 個, Z が 8 個であった.

観察の結果, バグ混入時に特徴的な行動を 6 個発見することができた. 発見した特徴 a ~ f を, 表 1に列挙する. 左列には, カーソルの動きや画面の変化における特徴を示している. 中列には, バグの混入したタイミングや位置を示している. 右列には, それらの状況から我々が推定した, 被験者の行動を示している. 表からわかるように, a ~ e の 5 個の特徴は, キーストロークデータだけの情報から自動的に検出できると思われる. 特徴 f については, コピー元の誤りの有無が関係するので, 自動的に検出することが難しい.

各特徴が現れた頻度を表 2に示す. 1 個のバグの混入時に複数の特徴が現れることもあるので, 特徴の出現回数とバグの総数とは, 必ずしも一致しない. ‘特徴を認めず’ と記した項目は, a ~ f のどの特徴も現れなかった場合を表す.

なお, この実験では 6 個の特徴を発見できたが, 収集したデータの量や実験条件が限られていたことに注意する必要がある. 従って, これらの特徴が他の条件の下でも現れるかどうかや, その他の特徴が存在するかどうかについては, 更に調査を要する.

5 行動分析部

行動分析部とは, 2 章で述べたように, キーストロークデータを分析して, プログラマの行動系列を記録する部分である. 特徴検出部においては, この行動系列の中で, 与えられる特徴が現れている位置が特定される.

そこで, 行動系列の形式は, 少なくとも 4 章で述べたバグ混入時の行動の特徴を調べる上で, 十分な情報を含むように決定される必要がある. また, 4 章で述べた特徴の他にも, バグ混入時の特徴が存在する可能性があるので, それらの未知の特徴に対する配慮も必要である.

そこで, 行動系列の形式を決定するために, 我々は, 先ず, 4 章で述べた特徴を参考にしながら, 行動系列に含めるべき情報を列挙した. 次に, その結果に基づいて, 行動系列の形式を決定した.

5.1 必要な情報

4 章で列挙したような特徴を検出するためには, ソースプログラム中での文字列の挿入, 変更, 削除, 貼り込み, カーソルの移動などの行動の順序を知る必要がある.

特徴 e のような特徴を検出するためには, 更に, 時間に関する情報も必要である. 1 個の行動が開始されてから終了するまでの時間や, 2 個の行動の間の時間である. 更に, 特徴 e を検出するためには, ソースプログラム中の各行について, その入力に要した時間を知る必要がある. 従って, 複数の行が続けて入力された場合, 仮にそれを 1 個の行動と考えると, 行動の開始と終了との時間だけでは, 情報が不足することになる. そこで, そのような場合には, 複数の挿入行動が続けて行なわれたと考えることにする.

特徴 d のような特徴を検出するためには, カーソルの移動の量を知る必要がある. 但し, カーソルを移動するエディタコマンドが続けて入力される場合, 各コマンドを入力する細かい行動については, 知る必要がない. 一連のコマンドの入力を 1 個の行動とみなして, その時にカーソルの移動した行の数さえわかればよい.

特徴 a のような特徴を検出するためには, ソースプログラム中の各行がどの行動によって生成されたかを知る必要がある. 例えば, 変更された行が, 貼り込みによって生成された行かどうかである.

5.2 行動系列の形式

行動系列は, 次の 7 項組の系列である.

$$(A, T_b, T_e, L, N_b, N_e, S)$$

各記号は, 次のデータを表す.

A. 行動の種類である. {insert, change, delete, copy, paste, move} の中の値をとる. それぞれ, プログラムの部分の挿入, 変更, 削除, 写し取り, 貼り込み, カーソルの移動を意味する.

T_b, T_e. 行動の開始時刻と終了時刻である. 秒単位のデータである.

L. 行動を開始した時のカーソルの位置を表す行識別子である.

N_b, N_e. 行動を開始した時と終了した時とのカーソルの位置を表す行番号である.

S. 挿入, または, 貼り込みによって新たに行が作られた場合の, 行の集合である. 行が作られない場合は, '$\emptyset$' と書く. 作られた場合は, '$L_b \ldots L_e$' と書く. ここで, 'L_b' と 'L_e' とは, それぞれ, 先頭の行と後尾の行との行識別子を表す.

行の識別子と番号とは異なるデータである. 行番号とは, プログラムの先頭から順に数える時の行の番号である. 同一の行でも, その行の上に別の行が作成されれば, 行番号が変わる. 行識別子とは, 作成された順に数えた行の番号である. 行の挿入や削除が起こっても, 各行の行識別子は変わらない. 前節で述べたように, どちらも必要なデータである.

行の挿入については, 1 個のエディタコマンドで 1 行よりも多くの行が挿入されることがある. 但し, 前節でも述べたように, 1 行より多い挿入は, 連続した複数の挿入として表される.

6 特徴検出部

特徴検出部は, 2 章で述べたように, 検出するべき行動の特徴が変わっても柔軟に対応できるように設計されている. つまり, 検出するべき特徴の形式的な記述を, 外部から入力する方式を採っている.

この記述の形式は, 次のいくつかの条件を満たすよう決定される必要がある. 第 1 に, バグ混入時の特徴を記述する上で十分な表現能力を持た

表 3: 特徴の形式的記述

a	$(\text{paste}, -, -, -, -, -, L_{b1} \ldots L_{e1})$ $(\text{move}, -, -, -, -, -, -)^*$ $\{\ (\text{change}, -, -, L_2, -, -, -)+$ $\quad (\text{delete}, -, -, L_2, -, -, -)+$ $\quad (\text{insert}, -, -, L_2, -, -, \emptyset)\ \}$ $\{\ (\text{move}, -, -, -, -, -, -)+$ $\quad (\text{change}, -, -, L_3[i], -, -, -)+$ $\quad (\text{delete}, -, -, L_3[i], -, -, -)+$ $\quad (\text{insert}, -, -, L_3[i], -, -, \emptyset)\ \}^*$ where $L_{b1} \leq L_2 \leq L_{e1}$ and $L_{b1} \leq L_3[i] \leq L_{e1}$
b	$(\text{insert}, -, T_{e1}, L_1, -, -, \emptyset)$ $(\text{move}, -, -, -, N_{b2}, N_{e2}, -)$ $\{\ (\text{change}, -, -, -, -, -, -)+$ $\quad (\text{delete}, -, -, -, -, -, -)+$ $\quad (\text{insert}, -, -, -, -, -, -)+$ $\quad (\text{paste}, -, -, -, -, -, -)\ \}$ $(-, -, -, -, -, -)^*$ $\{\ (\text{change}, T_{b3}, -, L_1, -, -, -)+$ $\quad (\text{insert}, T_{b3}, -, L_1, -, -, \emptyset)\ \}$ where $\lvert N_{b2} - N_{e2} \rvert \geq 20$ and $T_{b3} - T_{e1} \leq 60$
c	$\{\ (\text{change}, -, -, L_1, -, -, -)+$ $\quad (\text{insert}, -, -, L_1, -, -, \emptyset)\ \}$ $(\text{move}, -, -, -, -, -, -)^*$ $\{\ (\text{change}, -, -, L_1, -, -, -)+$ $\quad (\text{delete}, -, -, L_1, -, -, -)+$ $\quad (\text{insert}, -, -, L_1, -, -, \emptyset)\ \}$ $(\text{move}, -, -, -, -, -, -)^*$ $\{\ (\text{change}, -, -, L_1, -, -, -)+$ $\quad (\text{delete}, -, -, L_1, -, -, -)+$ $\quad (\text{insert}, -, -, L_1, -, -, \emptyset)\ \}$ $\{\ (\text{move}, -, -, -, -, -, -)+$ $\quad (\text{change}, -, -, L_1, -, -, -)+$ $\quad (\text{delete}, -, -, L_1, -, -, -)+$ $\quad (\text{insert}, -, -, L_1, -, -, \emptyset)\ \}^*$
d	$\{\ (\text{change}, -, -, L_1, -, -, -)+$ $\quad (\text{insert}, -, -, L_1, -, -, \emptyset)\ \}$ $(\text{move}, -, -, -, -, -, -)^*$ $\{\ (\text{change}, -, -, L_1, -, -, -)+$ $\quad (\text{delete}, -, -, L_1, -, -, -)+$ $\quad (\text{insert}, -, -, L_1, -, -, \emptyset)\ \}$ $(\text{move}, -, -, -, N_{b2}, N_{e2}, -)$ where $\lvert N_{b2} - N_{e2} \rvert \geq 20$
e	$(\text{insert}, T_{b1}, T_{e1}, -, -, -, -)$ where $T_{e1} - T_{b1} \geq 60$

なければならない. 第 2 に, システムを利用する者にとって, 容易に記述できる形式でなければならない. 第 3 に, システムが自動的に処理できるように, 厳密に, かつ, 具体的に定義されていなければならない. 以下では, 決定した記述の形式と記述の例とについて述べる.

なお, 特徴検出部の出力は, 特徴の種類と, それを検出した時刻との組である.

6.1 特徴の記述の形式

1 個の特徴は, 次のように記述する.

系列部　where　条件部

系列部とは, 行動の系列の正規表現である. 例えば, ある部分が貼り込まれた直後に, その部分の修正が繰り返されると言う系列は, 次のように記述する.

$$
\begin{array}{l}
(\mathrm{paste}, T_{bp}, T_{ep}, L_p, N_{bp}, N_{ep}, L_{bp} \ldots L_{ep}) \\
(\mathrm{move}, T_{b0}, T_{e0}, L_0, N_{b0}, N_{e0}, \emptyset)* \\
\{ \quad (\mathrm{change}, T_{b1}, T_{e1}, L_1, N_{b1}, N_{e1}, \emptyset) + \\
\qquad (\mathrm{delete}, T_{b1}, T_{e1}, L_1, N_{b1}, N_{e1}, \emptyset) \ \} \\
\{ \quad (\mathrm{move}, T_b[i], T_e[i], L[i], N_b[i], N_e[i], \emptyset) + \\
\qquad (\mathrm{change}, T_b[i], T_e[i], L[i], N_b[i], N_e[i], \emptyset) + \\
\qquad (\mathrm{delete}, T_b[i], T_e[i], L[i], N_b[i], N_e[i], \emptyset) \ \}*
\end{array}
$$

この記述では, 最初に貼り込みが行なわれ, 次に 0 回以上のカーソル移動が行なわれ, 次に変更または削除が 1 回行なわれ, その後にカーソルの移動や変更や削除が繰り返されることが表されている. 正規表現に用いられる一般的な記法と同様に, 'X+Y' は 'X' または 'Y' を, 'X*' は 'X' の 0 個以上の繰り返しを意味する. また, 行動の種類の後ろに続く記号は, 行動系列の形式の中の項目と対応しており, それぞれ, 行動の開始時刻, 終了時刻, カーソルの位置などを表している. 鍵括弧に囲まれた添字は, 行動が繰り返される場合に付けられ, 各行動の番号を表している.

条件部とは, 各行動の属性についての条件の列挙である. 各条件は, 等式, または, 不等式で記述する. 上の系列部の例では, 変更や削除の対象となる行が, 貼り込みによって作成された行であると言う条件が表されていなかった. そのような条件を表すためには, 次のような不等式を条件部に記述する.

$$L_{bp} \leq L_1 \leq L_{ep} \quad \mathrm{and} \quad L_{bp} \leq L[i] \leq L_{ep}$$

貼り込みによって作られた行の識別子は, L_{bp}, $\ldots$, L_{ep}である. 従って, 前の不等式は, 1 回目の修正の対象がそれらの中の 1 行であることを表している. 後の不等式は, 2 回目以降の修正の対象を表している.

6.2 特徴の記述の例

4 章の表 1 に示したバグ混入時の特徴 a ~ e を, 前節の形式で記述し直した. 但し, 特徴 f だけは, 自動的に検出できないので記述していない. 結果を表 3にまとめる. 表では, 条件部に現れない記号を '—' と略している. また, '$|X|$' は, Xの絶対値を表している.

7 適用例

試作したシステムを使用して, 特徴の出現とバグの混入との関連を評価するためのデータを収集した. 具体的には, 表 1, および, 表 3の中で示した特徴 a と e とのそれぞれについて, 次のデータを収集した. 先ず, 特定の種類の行動について, 特徴が出現した回数と出現しなかった回数とを, システムを使用して測定した. 次に, それぞれの場合について, その行動によってバグが混入した回数としなかった回数とを, 人手で測定した. 最後に, 測定した 4 個の数の比率を求めた.

仮に, 特徴の出現とバグの混入とが独立の事象であるとすれば, 特徴が出現する時にバグが混入する率は, 特徴が出現しない時にバグが混入する率に近くなるはずである. 逆に, 特徴の出現とバグの混入との間に強い関連があれば, 特徴が出現する時にバグの混入する率は, 出現しない時に比べて大きくなるはずである.

7.1 手順

対象とした行動系列は, 4 章で述べた 3 名のデータである. 特徴 a については, 挿入, 変更, 削除, 貼り込みの行動の中で, 特徴が現れた場合と現れなかった場合とを測定した. 特徴 e については, 挿入の行動の中で, 特徴が現れた場合と現れなかった場合とを測定した. 特徴 a と e との形式的な記述は, 表 3中に示している.

7.2 結果

表 4: 特徴出現とバグ混入との関係

特徴 a

	バグ混入	混入せず	混入率
特徴出現	11	136	0.075
出現せず	25	1992	0.012

特徴 e

	バグ混入	混入せず	混入率
特徴出現	2	17	0.105
出現せず	4	1198	0.003

収集したデータを表4に示す. 特徴 a と e のどちらについても, 特徴が出現する時にバグの混入する率は, 出現しない時に比べて非常に大きいことがわかる. 従って, 特徴 a や e が出現する時には, バグが混入する危険性が高いと言える.

なお, 特徴が出現しない時に比べれば危険性は高いが, その確率が 1 割程度であることに注意する必要がある. もし, 特徴 a が出現する度に警告が出す方式を採り, しかも, 特徴 a の出現する頻度が高ければ, プログラマを苛立たせるかもしれない. 仮に, 特徴 a が 1 分に 1 回の頻度で現れるとすると, 無駄な警告が 10 分間に 9 回の率で現れることになる.

但し, システムを使用して収集したデータによれば, 特徴 a の現れる頻度は約 3 分, 特徴 e の現れる頻度は約 20 分である. 従って, 無駄な警告が出される頻度は, それ程, 高くない. 有効な警告が出される頻度に対して, 無駄な警告がどの程度の頻度で出されれば, プログラマにとって有害であるか, 現状では, わかっていない.

7.3 システムの効果

前節で述べたように, 試作したシステムは, キーストロークからバグの混入を警告することが可能であるかどうかを検討する上で, 有用なデータを収集できた. そして, このようなシステムを使用しない場合に比べて, 高速にデータを収集することができた.

我々は, 表 1中の特徴の出現回数を, 人手で測定することも試みている. 例えば, 特徴 a の出現回数の測定には, 1 人が約 12 時間の労力を費やした. システムを使用することにより, この測定を数分で終らせることができるようになった.

8 おわりに

キーストロークからバグ混入時の特徴を検出するシステムの設計と部分的な実装について述べた. また, システムを使用してデータを収集した例を通して, このシステムが有効に働いていることを示した.

これまでに, 4 章および 7 章で述べた実験では, キーストロークデータからバグの混入を警告する方式が, 実現可能であることを示唆するデータが得られている. 今後は, 試作したシステムを使用して, より多くのデータの収集と分析を繰り返し, 方式の実現可能性を調査していく予定である.

参考文献

[1] Boehm B. W.: “Software engineering”, IEEE Trans. Computers, pp. 1226–1241 (1976).

[2] Nakajo T. and Kume H.: “A case history analysis of software error cause-effect relationships”, IEEE Trans. Soft. Eng., **17**, 8, pp. 830–837 (1991).

[3] 下村 隆夫: “CASE ツールの開発における ソフトウェアバグの分析”, 情処論, **35**, 7, pp. 1380–1389 (1994).

[4] Takada Y., Matsumoto K. and Torii K.: “A programmer performance measure based on programmer state transitions in testing and debugging process”, Proc. 16th Int. Conf. Soft. Eng., pp. 123–134 (May 1994).

[5] 柳, 高田, 鳥居: “キーストロークに着目したバグ混入時のプログラミング行動の特徴の分析”, 情処全大, pp. 5-511–5-512 (1994 秋).

[6] Youngs E. A.: “Human error in programming”, International Journal of Man-Machine Studies,, **6**, pp. 361–374 (1974).