1M－7

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*

柳 正純† ††† 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
†††三菱電機関西コンピュータシステム (株)

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, コスト的にも人道的にも問題が多い. そこで, 我々は, 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ことが可能であるかどうかを検討している.

そのために, キーストロークデータを収集し人手で分析することにより, バグ混入時のプログラマの行動パターンを見い出すことを試みた. 本稿では, 3名のプログラマのキーストロークデータから発見した, バグ混入時の行動パターンについて報告する.

2 分析方法

2.1 分析の手順

1. 被験者に小さなプログラムの仕様を与え, 設計から, コーディング, テストまで行なってもらう. そして, この間のキーストロークを全て記録する.
2. テストが開始されてから後のプログラムの変更点をキーストロークデータから調べることにより, テスト中に修正された全てのバグを特定する. (つまり, テスト中の変更を全てバグの修正とみなす. テストの開始は, プログラムが最初にコンパイルされた時点とみなす.)
3. 各バグの混入時刻 (つまり, その部分が打たれた時刻) を, コーディングまで遡り特定する. キーストロークデータを使って秒単位まで特定する.

*"When Programmers Inject Bugs?": Programmers' Behavior Analysis by Using Key Stroke Data
Masazumi Yanagi, Yoshihiro Takada, and Koji Torii
Graduate School of Information Science, Nara Institute of Science and Technology, 8916-5 Takayama-cho, Ikoma-shi, Nara 630-01, Japan

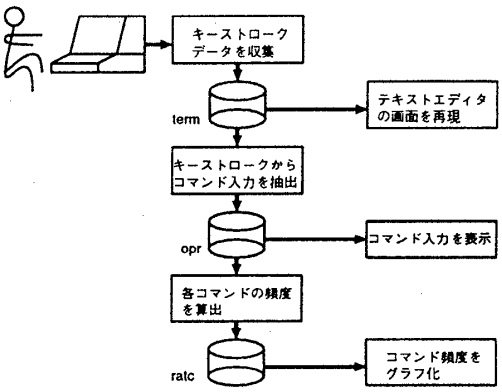

図 1: プログラミング行動観測・分析環境構成

4. 各バグについて, 混入時刻前後のテキストエディタの画面をキーストロークデータから再現し, 画面の変化やカーソルの動きを観測することによって, そのバグの入った原因を推定する.

2.2 プログラミング行動観測・分析環境

プログラマのキーストロークを監視・分析するために, 我々の構築している観測・分析環境 [2] を使用した. 使用した環境は, ネットワークを介してキーストロークデータを収集する部分と, そのデータを分析する部分からなる (図1). カーソルの動きや画面の変化を再現するツール, キーストロークデータを加工したり, 表示するツールを使用した.

2.3 実験

被験者は, A, B, C 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る. 与えたプログラム仕様は, 被験者によって異なる. どれも, 迷路の作成のような小規模なプログラムの仕様である.

プログラムは C 言語で書かれ, エディタは vi が使用された. テスト後の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. テスト中に修正されたバグの数は, A が 12 個, B が 21 個, C が 8 個であった. 設計からテストまでの間のキーストロークの総数は, A が 11,497 回, B が 18,986 回, C が 20,638 回であった.

表 1: バグ混入時のパターン

	画面やカーソル	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. この (変更箇所を含め) 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, 別の部分の入力や変更が行なわれた後, カーソルが元の位置に戻り, 行の入力が再開される. 再開後に入力された部分に誤りが入る.	ある部分を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した何かの作業をしなければならないことに気付き, それを行なった. 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. 最後に行なわれた変更が誤りになる.	ある部分について, 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. 始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.	行の入力の途中, または, 直後に, その行に誤りを入れたことに気付いた. 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残る.	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. その結果, 誤りが増える.	コピーする元の部分に誤りがないことを確認しなかったので, 誤りに気付かなかった.

3 分析結果

カーソルの動きや画面の変化を再現して観測した結果, バグ混入時の前後には, 多くの場合に, いくつかの特徴的なパターンが現れることがわかった. 発見したパターン a〜f を, 表1に列挙する. 左列に, 観測されたカーソルの動きや画面の変化のパターンを示す. 右列に, それらの観測パターンに対して, 我々が推定したプログラマの行動パターンを示す. 表からわかるように, a〜e の 5 個は, キーストロークデータだけから自動的に検出できるパターンである. 最後の f については, コピー元の誤りの有無が関係するので, 検出が難しい.

プログラマ A, B, C のそれぞれについて, バグ混入時に上記のパターンが現れた回数を表2に示す. (1 個のバグの混入時に, 複数のパターンが同時に現れる場合もあるので, パターンの出現回数とバグの総数は, 必ずしも一致しない.) 「パターンを認めず」と記した項目は, a〜f のどのパターンも現れなかった場合を表す, そのような場合は, 更に 3 種類に分けられる.

表 2: パターンの出現頻度

観測パターン	A	B	C	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
パターンを認めず	6	6	4	16
合計	12	22	8	42

1. データよりバグの混入した時刻も原因も特定できたし, 特徴的な行動が見られたが, そのような原因で発生したバグが 1 個しか観測されなかったので, 一般的であるかどうかわからなかった場合である. そのような原因としては, 次の例がある. 変数のデータタイプの宣言を変更した後, その変数の現れるいくつもの箇所を変更した. その時, ある箇所の変更を忘れた.

2. バグの混入した時刻も原因も特定できたが, その前後で, 特徴的な行動パターンが認められなかった場合である. そのような原因の例としては, 単なるキーの打ち誤りや, 設計時の誤りなどがある.

3. バグの混入した時刻は特定できたが, その原因が推定できなかった場合である.

4 むすび

キーストロークデータから検知できる, バグ混入時のプログラミング行動のパターンを報告した. 今回の実験は, 限られた条件の下で行なわれたため, 他の条件でもこれらのパターンが現れるとは, 必ずしも言えない. 今後は, 実験を拡張し, パターンの一般性を検証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] Takada, Y., Matsumoto, K. and Torii, K.: A programmer performance measure based on programmer state transitions in testing and debugging process, Proc. ICSE-16, pp. 123–134 (1994).