CHAPTER - 8

EXPERIMENTAL STUDY OF FAULTS PREDICTION
CHAPTER 8

EXPERIMENTAL STUDY OF FAULTS PREDICTION

8.1 INTRODUCTION

As stated earlier, the main concern of the software industry is to control and reduce the ever increasing cost of software development and maintenance. Software maintenance effort requires a major share of the computing budget in most computer installations and the burden increases with time. Estimates suggest that about 40% to 50% of the total computing budget is incurred on software maintenance [2,3,12]. Software maintenance is linked with maintainability. Maintainability is the probability that a failed system will be restored to operable conditions within a specified time. Also, software development is an extremely complex activity, full of different chances to introduce errors. The increasing complexity involved in software development has increased the need for objective measurement of software quality and complexity. In order to control high maintenance costs, the primary concern of software engineers is to control the complexity and introduction of faults in software development process. This concern has motivated various researchers to define and design various metrics for complexity and faults prediction techniques [2-3,12,16,24].

Quality of a software product is directly linked with the faults present in a software module and it improves as faults are reduced. If somehow, we could predict the number of faults present in a computer program, then we can comment about its quality as well [61-62,64,72].

8.2 FAULTS PREDICTION THROUGH BEBUGGING

The process of isolating and rectifying faults in a computer program is called debugging. Bebugging is the reverse of debugging. In bebugging method, a fixed number of known faults called artificial bugs are inserted in a source program. The complete documentation of these known bugs is kept so that after the experiment, these can be identified and removed from the source program [75].

By applying bebugging method, we may know how many bugs are still present in the code i.e. we may predict the faults present in a computer program and thus in a software system [75].

Suppose P is the source program in which we want to predict the number of faults present through bebugging method.

Let

\[
\begin{align*}
I & = \text{Faults inserted knowingly (artificial faults) in program P,} \\
R & = \text{Total faults reported by code reviewer or a tester in source program P,} \\
X & = \text{Faults reported which are from I,} \\
(R-X) & = \text{New additional faults reported by code reviewer/tester} \\
\end{align*}
\]

Total number of faults predicted in the program \( P = PF = \left( \frac{R - X}{X} \right) \times I \).
The bebugging method is generally used by zoologists in estimating the number of fish in a tank or a butterfly population.

For example, draw a sample of 1000 fish from a pond. Mark them and put back into the pond for mixing these with the total population of the tank. Then take a new sample of say 1000 fish and find how many marked fish are there in it. Let 100 are marked out of these 1000. Then bebugging method tells us that there are 9000 fish in the pond. In this method, we assume that the original sample was random and that the re-mixing of the fish was homogeneous.

Similarly, if we insert 10 bugs in a program and the code reviewer report 15 faults at a particular point of debugging. Out of these 15 bugs, let 5 are out of the 10 bugs inserted by us. Then

\[ I = 10, \ R = 15, \ X = 5, \ R - X = 10. \]

So, \( \left( \frac{R - X}{X} \right) * I = 20 \) bugs are the predicted number of faults which are still present in the program.

### 8.3 EXPERIMENT DESIGN

In this experiment, the main objective is to predict expected number of faults in a program by the application of bebugging method described in the preceding section. Also, the effect of complexity on faults predicted was analysed. It seems in agreement with intuition that if a programmer is asked to find errors in two programs of different complexities, then in a given time period, he/she would detect less number of faults in the complex program than simpler one.

For this purpose, a class room experiment was conducted on scheduled date and time. An experimental method was designed. The experimental materials included seven programs written in Pascal language with varying complexities. These programs have been collected from the open literature and their cyclomatic number lies between 1 and 7. In each of the seven programs, 5-6 logical and syntax errors were inserted knowingly called artificial bugs.

Subjects for the experiment were the 15 students of first year of Master of Computer Application (MCA) course at the Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak. The experiment was conducted at the end of the academic session so that by that time students have gained a lot of programming experience in FORTRAN and PASCAL languages. The experiment was conducted in the class room on scheduled time. The language used for the experiment was Pascal. The purpose and objectives of the experiment were explained well to the students just prior to the conduct of the experiment. The students were encouraged to seek clarifications about the experiment and their cross-questioning was answered.

The experiment was conducted in 7 continuous sessions of 10 minutes duration each. In each session, one program with 5-6 artificial bugs was given to each of the 15 subjects.
They were asked to find and report as many number of logical and syntax errors as you can find in each program in specified time period of 10 minutes. After the end of first session, sheets of first program were collected from the students. In this way 7 different programs were given to the students in 7 different sessions of 10 minutes duration each. One sample program has been given in Figure 8.1

8.4 ANALYSIS AND INFERENCES

For all the 7 programs, the number of experimental predicted faults (EPF) were calculated for each of the subject (S1 - S15). Tables 8.1 summarizes the results. In chapter 6, by using Lipow’s theoretical concepts, we have calculated faults rates (number of faults per line of code) for various high level programming languages including Pascal. For Pascal the fault rate has been calculated as 0.0067. By using this value, we have also predicted the number of faults in each of the 7 Pascal programs analysed here. These faults have been called as theoretical predicted faults (TPF). We have also calculated the composite (Cw), McCabe’s and weighted (Ew) complexity metrics for the programs studied here. These values alongwith the average values of the experimental predicted faults (EPF) calculated in Table 8.1 for all the 7 programs have been tabulated in Table 8.2.

We have also calculated the correlation coefficient (r) between theoretical predicted faults (TPF) and the experimental predicted faults (EPF) and it was found positive (+0.28). From Table 8.2 and this positive value of r, we infer that there is a positive relationship between TPF and EPF. This gives an empirical support to the bebugging method for faults prediction in a computer program. However, from Table 8.2 and plots in figures 8.2 and 6.2 (chapter 6), it is clear that the experiment performed on bebugging technique leads to better results than Lipow’s method. The reason for this may be that our experiment is based on realistic data while Lipow’s method is based on assumptions which may not be correct.

We have also analysed the relationship between complexity and predicted number of faults through bebugging method. For this purpose, we have drawn plots among complexity metrics described in Table 8.2 and the faults. These plots have been given in Figures 8.2-8.3. The increasing trend of the curves in these plots establishes the fact that fault rate is directly proportional to complexity i.e. more complexity -> more possibility of faults. Similar inference was also obtained in chapter 6. Such a matching of the results with intuition is an interesting finding of this study.
Figure 8.1

Sample Program

Program: to find the highest and second highest marks.

01 PROGRAM HIGHMARK(INPUT,OUTPUT)
02 VAR
03 MARKS: ARRAY[1..4] OF INTEGER;
04 FIRST, I INTEGER;
05 BEGIN
06 FOR I := 1 TO 5 READ(MARKS[I]);
07 FIRST := 0;
08 FOR I := 1 TO 5 DO
09 IF MARKS[I] > FIRST THEN FIRST := MARKS[I];
10 SECOND := 0;
11 FOR I := 1 TO 5 DO
12 IF MARKS[I] <> FIRST
13 THEN
14 IF MARKS[I] > SECOND
15 THEN SECOND := MARKS[-I];
16 WRIELN(FIRST,SECOND);
17 END

LINE NO.  ERROR TYPE  ERROR DESCRIPTION
01  Syntax  Missing ".".
06  Logical  Do missing.
06  Logical  Array out of range.
10  Syntax  SECOND not defined.
15  Logical  Subscript I cannot be negative.
16  Syntax  WRIELN not defined.
17  Syntax  "." missing.


Table 8.1

Experimental Predicted Faults (EPF) per Subject per Program

<table>
<thead>
<tr>
<th>P. No</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
<th>S10</th>
<th>S11</th>
<th>S12</th>
<th>S13</th>
<th>S14</th>
<th>S15</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.5</td>
<td>2</td>
<td>0</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>3.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.25</td>
<td>1</td>
<td>3.3</td>
<td>0</td>
<td>1.25</td>
<td>2.5</td>
<td>3.75</td>
<td>0</td>
<td>1.25</td>
<td>1.25</td>
<td>1.25</td>
<td>1.66</td>
<td>1.66</td>
<td>2.5</td>
<td>1.25</td>
<td>23.87</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1.5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1.5</td>
<td>1.5</td>
<td>0</td>
<td>0</td>
<td>1.5</td>
<td>2</td>
<td>1.5</td>
<td>13.5</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1.67</td>
<td>7.5</td>
<td>3.75</td>
<td>2.5</td>
<td>3.3</td>
<td>5</td>
<td>1.25</td>
<td>3.75</td>
<td>1.67</td>
<td>2.5</td>
<td>1.67</td>
<td>5</td>
<td>3.4</td>
<td>1.67</td>
<td>49.63</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1.2</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>1.2</td>
<td>1.5</td>
<td>1</td>
<td>2</td>
<td>1.2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1.2</td>
<td>1</td>
<td>28.3</td>
</tr>
</tbody>
</table>

Table 8.2

Values of theoretical and predicted faults, $C_w$, $V(G)$, and $E_w$

<table>
<thead>
<tr>
<th>P. No</th>
<th>Theoretical Pred Faults (TPF)</th>
<th>Experimental Pred Faults (EPF)</th>
<th>$E_w$</th>
<th>$V(G)$</th>
<th>$C_w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.04</td>
<td>.80</td>
<td>4936</td>
<td>1</td>
<td>163</td>
</tr>
<tr>
<td>2</td>
<td>.04</td>
<td>.87</td>
<td>4241</td>
<td>1</td>
<td>166</td>
</tr>
<tr>
<td>3</td>
<td>.08</td>
<td>.45</td>
<td>8022</td>
<td>1</td>
<td>432</td>
</tr>
<tr>
<td>4</td>
<td>.05</td>
<td>1.59</td>
<td>3118</td>
<td>3</td>
<td>175</td>
</tr>
<tr>
<td>5</td>
<td>.06</td>
<td>90</td>
<td>2497</td>
<td>2</td>
<td>231</td>
</tr>
<tr>
<td>6</td>
<td>.07</td>
<td>3.31</td>
<td>10855</td>
<td>4</td>
<td>419</td>
</tr>
<tr>
<td>7</td>
<td>.07</td>
<td>1.88</td>
<td>7562</td>
<td>7</td>
<td>482</td>
</tr>
</tbody>
</table>
Figure 8.1

Plot between Experimental Errors (EPF) vs. V(G)
Figure 8.2

Plot between Experimental Errors (EPF) vs. Ew
8.5 SUMMARY

In this chapter, we have investigated a method for predicting the faults in a source program by artificially inserting faults in it. This method is named as bebugging method. Then a class-room experiment was conducted to predict faults in 7 Pascal programs of different complexities through the proposed bebugging method. A positive relationship was found between theoretical predicted faults (Lipow's method) and the faults predicted through this method. However, the experiment conducted on bebugging technique leads to better results than Lipow's method. The reason for this may be the fact that our experimentally predicted results are based on realistic data while Lipow's method is based on certain assumptions. A relationship was also established between complexity and faults. It establishes the fact that more the complexity, more the expected number of faults.