Structural Approach to the Estimation of the Number of Residual Software Faults Based on the Hyper-Geometric Distribution

YOSHIHIRO TOHMA, FELLOW, IEEE, KENSHIN TOKUNAGA, SHINJI NAGASE, AND YUKIHISA MURATA

Abstract—New models based on the hyper-geometric distribution for estimating the number of residual software faults are proposed. The applications of these models to real data are demonstrated.

Index Terms—Debugging, growth, hyper-geometric distribution, modeling, reliability, software, test.

I. INTRODUCTION

The more important roles computer systems play in highly computerized society, the more serious damage we may suffer upon the failure of computer systems. Therefore, computer systems must be very reliable.

Recent reduction of hardware cost has made redundancy techniques feasible so that hardware faults are well tolerated by these redundancy techniques. Contrarily, the software is still a simplex system in most applications and therefore, computer systems are quite vulnerable to faults in the software. Further, since the software used in most computer systems today is very huge and complicated, it is quite likely to suffer from faults. Thus, the reliability of software has recently become one of major issues in the realization of highly reliable computer systems.

A variety of models to evaluate the software reliability have been proposed. (See, for example, [1].) Many of them are those to estimate probabilistically the time duration between occurrences of errors in operation with the analogy to hardware faults [2]–[4]. Another approach is to argue the number of software faults still residual after debugging process. This paper presents new models by which we can estimate the number of residual faults in the software, observing the growth of the cumulative number of faults detected by the test. The applications of these models to real data will be demonstrated.

Several papers already argued the number of residual faults [5]–[7]. However, they only viewed the form of the growth curve of the cumulative number of detected faults, looking for appropriate functions which could fit the observed curve. However, the lack of reasoning why such functions were used would make us skeptical if their methods could be applied to other cases in general.

In view of the test process, we exploit the hyper-geometric distribution as the basis of our model. This probability distribution was considered earlier in the context of the capture-recapture sampling technique. (See, for example, [8].) While earlier studies noted the recaptured faults, we deal here with newly detected faults. They did not consider the debugging process, but we do in this paper. Thus, we claim that our use of the hyper-geometric distribution is different from the earlier ones.

In the usual process of the development of a software, programmers first test their products by themselves, using some debugging tools. Then, their products will be passed to test workers with the programmers’ confidence that the software is completely fault-free. However, more faults will be found by the test workers in most cases. Here we focus our attention on this test stage only.

Section II shows the definitions and assumptions made in this paper. Section III describes the basic model, and its two extensions will be argued in Sections IV and V. Section VI gives the concluding remarks.

II. DEFINITIONS AND ASSUMPTIONS

A. Notations and Definitions

The test of a software is a set of test instances. A test instance is a couple of an input test data and the observed test result. This set is denoted by \( D = \{ t(i) | i = 1, 2, \ldots, n \} \), where \( t(i) \) is the ith test instance, \( n \) is the total number of test instances.

In some cases, a test may be a collection of subsets of test instances. We call these subsets test classes, because each subset will have a characteristic nature common to all test instances of the subset, or it is aimed at a specific test item. For a moment, however, we regard all test instances to be of a single class. Sections IV and V will discuss the test made of several test classes.

A set of faults initially resident in the software (which will be simply referred to as initial faults) is expressed by \( B = \{ b(j) | j = 1, 2, \ldots, m \} \), where \( b(j) \) is a fault. Thus, the total number of initial faults is \( m \), which is to be estimated by our model.
Here we define the response function of a fault. When an error due to \( b(j) \) is observed in the test result of \( t(i) \), \( b(j) \) is said sensed by \( t(i) \), and the value of the response function \( r(i, j) \) of \( t(i) \) and \( b(j) \) is defined to be 1. Otherwise, \( r(i, j) = 0 \). \( S(i) \) is a set of faults which are sensed by \( t(i) \). That is, \( S(i) = \{ b(j) \in B \mid r(i, j) = 1 \} \). Note that the response function is defined for initial faults, before any maintenance action is taken place. Further, we distinguish here the sensitization of faults from the detection of faults.

In the subsequent, \( E \theta \) would denote an estimate of any parameter \( \theta \).

B. Assumptions

In debugging, test instances \( t(1), t(2), \cdots \) will be exercised. At a test instance, some initial faults will be newly attached to the detection will be omitted hereafter for simplicity) detected and the maintenance of faults from the detection of faults. The number of faults detected by a test instance may vary at actual test instances. Further, new faults may be inserted, when the detected faults are removed. In the subsequent analysis, however, we assume the following for simplicity:

1) Faults detected by a test instance are surely removed, before the next test instance is exercised.

2) No fault is newly inserted during the removal of the detected faults. Thus, the software reliability grows with the exercise of test instances.

3) A test instance \( t(i) \) senses \( w(i) \) initial faults, where \( w(i) \) may vary with the condition of test instances over \( i \).

4) Of course, initial faults actually sensed by \( t(i) \) depend on \( t(i) \) itself. However, they are regarded to be such \( w(i) \) initial faults that are taken randomly out of \( m \) initial faults, since the order of the exercise of test instances is indefinite or determined rather randomly within a class of tests.

III. BASIC METHODOLOGY

A. Model-I

The number of faults detected by the first test instance \( t(1) \) is obviously \( w(1) \) faults of \( S(1) \) by the definition. However, the number of faults detected by \( t(2) \) is not necessarily \( w(2) \), because some faults of \( S(2) \) may be removed already by \( t(1) \). Similarly, faults detected by \( t(3) \) are those of \( S(3) \) that are not yet sensed by \( t(1) \) and \( t(2) \). Thus, the number of faults detected by \( t(i) \), denoted by \( N(i) \), can be written as

\[
N(i) = \left| S(i) - \bigcup_{j<i} S(j) \right|
\]  

where \( | \cdot | \) represents the cardinality of a set. Then, the cumulative number of faults detected so far by test instances from \( i(1) \) to \( t(i) \) is

\[
C(i) = \sum_{j=1}^{i} N(j). \tag{2}
\]

Let the probability that \( t(i) \) detects \( x \) faults be \( \text{Prob}(x|m, C(i-1), w(i)) \). Then,

\[
\text{Prob}(x|m, C(i-1), w(i)) = \binom{m}{x} \left( \frac{C(i-1)}{w(i)} \right) \left( \frac{m-C(i-1)}{w(i)-x} \right) \tag{3}
\]

where \( x \) must be \( 0 \leq x \leq \min \{ w(i), m-C(i-1) \} \) (this minimum value will be denoted hereafter by \( U_x \)). The probability distribution of \( (3) \) is called the hyper-geometric distribution. The expected value \( \overline{N(i)} \) of \( N(i) \) is

\[
\overline{N(i)} = \text{ROUND} \left[ \sum_{x=0}^{U_x} x \text{Prob}(x|m, C(i-1), w(i)) \right] = \text{ROUND} \left[ \frac{(m-C(i-1))w(i)}{m} \right] \tag{4}
\]

where \( \text{ROUND} \) represents an appropriate round operator to make the expected value an integer [12]. If we use the following estimate \( EC(i-1) \) in place of actual \( C(i-1) \),

\[
EC(i-1) = \sum_{j=0}^{i-1} \overline{N(j)}, \overline{N(0)} = 0 \tag{5}
\]

we obtain step-by-step the estimate of the number of faults detected by \( t(i) \) for \( i = 1, 2, \cdots \) and accordingly the estimate of the cumulative number of faults detected by test instances from \( t(1) \) to \( t(i) \).

On the other hand, by using such estimate \( EC(j-1) \) for \( j - 1 \geq i \) as

\[
EC(j-1) = C(j-1) + \sum_{k=i}^{j-1} \overline{N(k)} \tag{6}
\]

we have the estimates for \( j-1 \geq i \), taking the actual \( C(j-1) \) into account. In the sequel, however, only the former case will be considered.

Instead of \( \overline{N(i)} \) of (4), we may use such \( MLN(i) \) as

\[
MLN(i) = \min \{ \text{Prob}(x|m, C(i-1), w(i)) \}
\]

is maximum at \( x \). \tag{7}

That is, \( MLN(i) \) takes the maximum likely \( x \).

Let us denote \( \text{Prob}(x|m, C(i-1), w(i)) \) simply by \( L(x) \). Then, \( L(x) \) takes its maximum value nearly at such \( x \) that

\[
\frac{L(x)}{L(x-1)} = 1. \tag{8}
\]
Solving this equation for \( x \), we obtain the approximation of \( MLN(i) \) as

\[
MLN(i) = \text{ROUND} \left[ \frac{m - C(i - 1) + 1}{m + 2} \left( w(i) + 1 \right) \right].
\] (9)

Parameters to be estimated are \( m \) and \( w(i) \). In order to get these estimates, \( EC(i) \)'s will be evaluated with the following measures.

\[
EF1 = \frac{1}{n} \sum_{i=1}^{n} |C(i) - EC(i)| \]

\[
EF1' = \frac{1}{n} \sum_{i=1}^{n} |i|C(i) - EC(i)| \]

\[
EF2 = \max |C(i) - EC(i)| \]

\[
EF3 = \frac{1}{n} \sum_{i=1}^{n} |C(i) - EC(i)| \]

\[
EF4 = \left( \frac{1}{n} \sum_{i=1}^{n} |C(i) - EC(i)| \right)^2 \] (10)

\( Em \) and \( Ew(i) \) will be determined so as to give the minimum value of the measure. The possible change of \( w(i) \) over \( i \) makes it difficult to get \( Em \) and \( Ew(i) \) analytically. Therefore, we rely on an exhaustive scan method over possible \( m \) and \( w(i) \), looking for \( Em \) and \( Ew(i) \).

### B. Observation

Here we show how Model-I fits real data. Table I is an example of the field report of the test of a software for monitoring and real-time control. The software consists of about 200 modules, of which size is about 1000 lines each in a high-level language like Fortran. Since these data are not intentionally gathered for evaluating our model, the detailed conditions of test instances are unfortunately not recorded in the data. Therefore, we must interpret the data in accordance with our model.

1) **Faults:** In real circumstances, how to count faults is a problem. A single fault sometimes causes multiple associated errors in operation. It is natural to count, as a fault, a modification of the software to remove the associated errors in these cases. In more complicated cases, however, an error may originate from a combination of faults. Two ways will be possible in these latter cases. The one is to regard the combination of multiple modifications for removing the associated error to be a single fault. The alternative is to count the modifications themselves.

However, it does not matter which way is used. The most important is the consistency in the way of counting faults. If this consistency is kept, we can interpret faults with the implicitly assumed definition. Hence, we do not argue the detailed meaning of faults in Table I.

2) **Test Instance:** As shown in Table I, \( N(i)'s \) were recorded day by day. How many test instances were exercised in a day was not known, nor the condition on which test instances were carried out. Therefore, we interpret the test exercised in a day as a test instance. Again, the detail of test instances is of no matter, provided the way of the interpretation is kept consistent.

3) **\( w(i) \):** We regard accordingly \( w(i) \) to be the number of initial faults sensed by \( t(i) \) defined above. However, \( w(i) \) is not given in Table I. Only information available is the number of workers who were engaged in a test instance. The more workers are involved, the more initial faults will be likely sensed by a test instance. Therefore, we assume that \( w(i) \) is a function of the number of workers at \( t(i) \) in the form

\[
w(i) = A_e \left( \text{tester}(i) \right)^p \] (11)

where \( \text{tester}(i) \) represents the number of workers involved in \( t(i) \), \( A_e \) and \( p \) being constants.

The reason why we consider the exponent \( p \) is that the test efficiency will not necessarily be doubled exactly, even if the number of workers in a test instance is increased two times.

As described previously, \( w(i) \) will be scanned over an
TABLE II
ESTIMATES OF m, p, AND maxw [(4) WAS USED]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range</th>
<th>Increment</th>
<th>EF1</th>
<th>EF2</th>
<th>EF3</th>
<th>EF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Em</td>
<td>450 - 600</td>
<td>3</td>
<td>597</td>
<td>600</td>
<td>588</td>
<td>600</td>
</tr>
<tr>
<td>Ep</td>
<td>1.0 - 3.0</td>
<td>0.2</td>
<td>1.6</td>
<td>1.6</td>
<td>2.4</td>
<td>2.2</td>
</tr>
<tr>
<td>Emaxw</td>
<td>20 - 45</td>
<td>3</td>
<td>79</td>
<td>79</td>
<td>38</td>
<td>29</td>
</tr>
<tr>
<td>(Ap)</td>
<td></td>
<td></td>
<td>(0.561)</td>
<td>(0.543)</td>
<td>(0.256)</td>
<td>(0.047)</td>
</tr>
</tbody>
</table>

Fig. 1. EC(i) and C(i) [(4) was used].

The estimates obtained by using (9) were not so much different from those obtained by (4). They are summarized in Table III. The values of the measure obtained by using (4) and (9), respectively, are compared in Table IV. Since little difference is observed, (9) will not be used in the following sections.

IV. SEGMENTATION

A. Model-2

Looking at Fig. 1 in more detail, we see that EC(i) does not follow C(i) very well over the ranges of the
The application of Model-1 to this data is not successful as shown in Fig. 2. (Since the number of test workers was represented by $n$.) We assume the parameters to take different values over a random distribution as before, but with different $m(j)$ and $w(i)$. However, if we consider each test class still to test the software as a whole, $m(j)$ should be the same constant for all test intervals.

Further, at the beginning of $T(j)$, $C(n(j-1), j-1)$ must be taken into account, where $n(j-1)$ is the number of test instances of $T(j-1)$. We assume that $C(0,j)$ is not equal to 0, but $C(n(j-1), j-1)$. In the actual estimation, however, $E(m(j-1), j-1)$ is used in place of $C(n(j-1), j-1)$.

In general, the last interval will be devoted to an overall test. In such a case, the estimate of $m(j)$ obtained there will be $E_m$ itself.

**B. Observation**

The application of the above segmentation technique to the test data of Table I revealed that the use of the same constant $m(j)$ over test intervals was not so favorable [10]. We speculate that the same constant $m(j)$ generally...
makes plots of $EC(i)$ rather smooth and accordingly, the use of the segmentation technique does not fit sharp bends of the growth curve of $C(i)$ well. Therefore, we assumed in the application of the segmentation technique to the test data of Table V that $m(j)$ would take different values over $j$.

Viewing test classes of Table VI and the bends on the plots of $C(i)$ of Fig. 2, we determined the test intervals as shown in Table VII. As before, we assumed $w(i, j)$ to remain constant in $T(j)$.

The result of the estimation is summarized in Table VIII. $EC(i, j)$'s and $C(i, j)$'s are plotted in Fig. 3. The improvement made by the segmentation technique is remarkable.

**V. Composite Estimation**

**A. Model-3**

In the above two sections, we did not consider what modules of the software were exercised by test instances. Here, a test class $TC(i)$, $i = 1, 2, \cdots$ is assumed to exercise only a part of the software. This part, denoted by $D(i)$, will be called simply the domain of $TC(i)$.
$D(i)$ may or may not have a common part with $D(j)$, $j \neq i$. Test classes and their domains can be determined from the structure of the software and the test items. However, $D(i)$ may not necessarily correspond to the modular structure of the software. Even if the software consists of modules, some of them may be exercised always together by a test class. In such a case, these modules exercised together should be regarded as a whole to be a domain of the test class.

Suppose that the test consists of three test classes $TC(a)$, $TC(b)$, $TC(c)$ and their respective domains $D(a)$, $D(b)$, $D(c)$ are as shown in Fig. 4. $D(a)$ is divided into four subdomains: $A$, $A \cap B$, $A \cap C$, and $A \cap B \cap C$. $A$ is the area proper to $TC(a)$, while $A \cap B$ is the area included commonly in $D(a)$ and $D(b)$. This means that faults in this area are sensed by both test instances of $TC(a)$ and $TC(b)$. Likewise, $A \cap B \cap C$ is the area common to $D(a)$, $D(b)$, and $D(c)$. Let us denote the numbers of initial faults in $D(a)$, $D(b)$, $D(c)$, $A$, $B$, $C$, $A \cap B$, $A \cap C$, $B \cap C$, $A \cap B \cap C$ by $m(a)$, $m(b)$, $m(c)$, $m(A)$, $m(B)$, $m(C)$, $m(A \cap B)$, $m(B \cap C)$, $m(C \cap A)$, and $m(A \cap B \cap C)$, respectively.

The detection of faults by test instances of $TC(a)$ is, of course, influenced by faults detected in the common area by test instances of $TC(b)$ and/or $TC(c)$. Just before the $i$th test instance of $TC(a)$ [which will be denoted by $t(i, a)$], let the cumulative numbers of faults detected in $A \cap B$, $A \cap C$, and $A \cap B \cap C$ by test instances of $TC(b)$ and $TC(c)$ be $C_a(i - 1, A \cap B)$, $C_a(i - 1, A \cap C)$, $C_a(i - 1, A \cap B \cap C)$, respectively, and the cumulative number of faults de-
ected by test instances of $TC(a)$ be $Ca(i - 1)$. Then, the faults detected by this $i(i, a)$ will be some of the remaining
\[
m(a) = C_a(i - 1) - C_a(i - 1, A \cap B) - C_b(i - 1, A \cap B \cap C) - C_c(i - 1, A \cap B)
\]
faits in $D(a)$. Therefore, the probability that the number of the detected faults is $x$ can be evaluated by (3) with $m(a)$ and
\[
C_a(i - 1) + C_b(i - 1, A \cap B) + C_c(i - 1, A \cap B \cap C)
\]
in place of $m$ and $C(i - 1)$, respectively. Except this substitution, $Em(a)$ will be obtained in the same way as described in Section III. $Em(b)$ and $Em(c)$ will be obtained likewise.

Obviously, the estimate of the total number of initial faults in the software may not be equal to the summation of $Em(a)$, $Em(b)$, and $Em(c)$. Note, for example, that both $Em(a)$ and $Em(c)$ include faults in $A \cap B$. Therefore, we must obtain $Em(A)$, $Em(B)$, $Em(C)$ as well as $Em(A \cap B)$, $Em(B \cap C)$, $Em(C \cap A)$, and $Em(A \cap B \cap C)$.

Let $C_a$ and $C(A \cap B)$ be the numbers of faults actually detected in $D(a)$ and $A \cap B$, respectively. We define coefficient $K_a(A \cap B)$ to be the ratio of $C(A \cap B)$ to $C_a$. Other coefficients will be defined similarly. Then, it seems reasonable to evaluate $Em(A \cap B)$ by
\[
Em(A \cap B) = Em(a) K_a(A \cap B).
\]
Similarly, $Em(A \cap B)$ may also be evaluated by
\[
Em(A \cap B) = Em(b) K_b(B \cap A)
\]
where $K_b(B \cap A)$ is the ratio of $C(B \cap A) = (C(A \cap B) \cap C)$.

If $Em(a)$ and $Em(b)$ would be accurately equal to $m(a)$ and $m(b)$, respectively, and further $Em(A \cap B)/Em(a)$ and $Em(A \cap B)/Em(b)$ would also be $K_a(A \cap B)$ and $K_b(A \cap B)$, respectively, both (13) and (14) would give us exactly the same value. However, it is not always the case, because $Em(a)$ and $Em(b)$ are only the estimates and possibly include errors. Further, due to the rather limited number of test instances, the relationship that $Em(A \cap B)/Em(a) = K_a(A \cap B)$ and $Em(A \cap B)/Em(b) = K_b(A \cap B)$ may not hold exactly. Thus, the above two equations may give us slightly different values. Therefore, we estimate $Em(A \cap B)$ by
\[
Em(A \cap B) = \left\{ Em(a) K_a(A \cap B)
+ Em(b) K_b(A \cap B) \right\}/2.
\]

$Em(B \cap C)$ and $Em(C \cap A)$ will be estimated similarly. $Em(A \cap B \cap C)$ will be estimated by
\[
Em(A \cap B \cap C) = \left\{ Em(a) K_a(A \cap B \cap C)
+ Em(b) K_b(A \cap B \cap C)
+ Em(c) K_c(A \cap B \cap C) \right\}/3.
\]

Then, $Em(A)$ is given by
\[
Em(A) = Em(a) - Em(A \cap B)
\]
\[
Em(A \cap C) - Em(A \cap B \cap C).
\]

The following remark should be in order. The cumulative number of faults actually detected in $A \cap B$, $A \cap C$, and $A \cap B \cap C$ by test instances of $TC(b)$ and $TC(c)$, for example, is used in the calculation of the estimate of the number of faults detected by a test instance of $TC(a)$. In Section III-A, however, $Em(i - 1)$ was used in place of the actual $C(i - 1)$. In this context, the way of calculation here is a little inconsistent with those in the preceding sections.

In principle, we can use $Em(a)(i - 1, A \cap B)$, $Em(b)(i - 1, A \cap B \cap C)$, $Em(c)(i - 1, A \cap C)$, and $Em(c)(i - 1, A \cap B \cap C)$. However, it makes extremely large the number of cases of scanning parameters to find their optimum values. Therefore, this approach may not be feasible in real applications.

B. Observation

Here we present the result of the composite estimation described above. The software used in this estimation was developed for monitoring and real-time control. It consists of about 90 modules. The size of modules is about 1 kiloline of PL/1, Fortran, and Assembler languages in average.

The data was collected on what modules were exercised by test instances and where faults were found. Since the software was tested on its three functional properties, the test instances were classified into $TC(a)$, $TC(b)$, and $TC(c)$. Three domains $D(a)$, $D(b)$, and $D(c)$ were identified accordingly. These domains have common areas as shown in Table IX. There was no module which was tested by $TC(b)$ and $TC(c)$ (that is, $B \cap C = \text{null}$).

The table shows the number of detected faults for each test date and for each subdomain. No faults were detected in $A \cap C$ by $TC(a)$, in $B \cap C$ by $TC(b)$, nor in $C \cap A$ and $C \cap B$ by $TC(c)$. The total number of workers in a day is divided for each test class in reference to the time spent for each test class. In some days, a test class (test classes) was (were) not exercised. These cases are indicated by "-" in the table. The number of detected faults, the cumulative number of detected faults, and the number of workers for each day are respectively summarized in the table.
From the table, we obtain

\[ C_a = 64, \ C_b = 76, \ C_c = 115, \]

\[ C(A \cap B) = 13, \ C(B \cap C) = C(C \cap A) = 0, \]

\[ C(A \cap B \cap C) = 22. \]

\[ K_a(A \cap B) = 13/64 = 0.203, \]

\[ K_a(a) = 0, \]

\[ K_a(A \cap B \cap C) = 22/64 = 0.344, \]

\[ K_b(B \cap C) = 0, \]

\[ K_b(B \cap C \cap A) = 13/76 = 0.171, \]

\[ K_b(B \cap C \cap A) = 22/76 = 0.289, \]

\[ K_c(C \cap A) = K_c(C \cap B) = 0, \]

\[ K_c(C \cap B \cap A) = 22/115 = 0.191. \] (18)

We estimated

\[ Em(a) = 71, \]

\[ Em(b) = 99, \]

\[ Em(c) = 148. \] (19)

Therefore, \( Em(A \cap B), \ Em(A \cap C), \ Em(A \cap B \cap C) \) are given by

\[ Em(A \cap B) = (1/2) \times (71 \times 0.203 + 99 \times 0.171) = 15.7, \]

\[ Em(A \cap C) = 0. \]

\[ Em(A \cap B \cap C) = (1/3) \times (71 \times 0.344 + 99 \times 0.289 + 148 \times 0.191) = 27.1. \] (20)

Others are summarized in Table X. \( Em \) is defined by

\[ Em = Em(A) + Em(B) + Em(C) + Em(A \cap B) + Em(B \cap C) + Em(C \cap A) + Em(A \cap B \cap C). \] (21)

It is 248. When the detailed structure of the software is ignored, its value (given by applying Model-1) is 230.

In order to estimate the growth of the cumulative number of detected faults, \( EC(i, a) \), the estimate of the cumulative number of faults detected by test instances from \( t(1, a) \) to \( t(i, a) \) of \( TC(a) \), as well as \( EC(i, b) \) for \( TC(b) \) and \( EC(i, c) \) for \( TC(c) \) were calculated, respectively, by the method described above. Note, however, that they must be allotted in accordance with their test date, because \( i \) is only the sequential number of a test instance and does not necessarily correspond to the date of the test instance. Then, the estimates of the cumulative number of detected faults in \( TC(a), TC(b), \) and \( TC(c) \) for each test date were added to each other, and the sum was compared to \( C(i) \) of Table IX. These are plotted in Fig. 5.

VI. CONCLUDING REMARKS

Based on the hyper-geometric distribution, we have proposed the basic model (Model-1) for estimating the number of initially resident faults in a software. Examples of the application of this model indicate that the fitness of Model-1 to real data is fairly good, when the cumulative

<table>
<thead>
<tr>
<th>Date</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>N(i) detected by TC(a) in (A)</td>
<td>0</td>
<td>8</td>
<td>13</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>N(A∩B)</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>N(A∩B∩C)</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subtotal</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>No. of workers in TC(a)</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>2</td>
<td>-</td>
<td>2.5</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>4</td>
<td>-</td>
<td>13.3</td>
<td>7</td>
<td>9.4</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>N(i) detected by TC(b) in (B)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>12</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>N(B∩C)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N(B∩C∩A)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subtotal</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5</td>
<td>11</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>12</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>No. of workers in TC(b)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.5</td>
<td>7</td>
<td>0.6</td>
<td>1.5</td>
<td>-</td>
<td>12</td>
<td>16</td>
<td>8.7</td>
<td>10</td>
<td>5.7</td>
<td>4.10</td>
</tr>
<tr>
<td>N(i) detected by TC(c) in (C)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>13</td>
<td>1</td>
<td>7</td>
<td>23</td>
<td>19</td>
<td>-</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>N(C∩A∩B)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subtotal</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>13</td>
<td>1</td>
<td>11</td>
<td>23</td>
<td>22</td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>No. of workers in TC(c)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>9.5</td>
<td>2.5</td>
<td>9.4</td>
<td>9.5</td>
<td>25</td>
<td>12</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>11.9</td>
<td>14</td>
</tr>
<tr>
<td>N(i)</td>
<td>6</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>18</td>
<td>13</td>
<td>12</td>
<td>26</td>
<td>23</td>
<td>22</td>
<td>9</td>
<td>19</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Subtotal</td>
<td>6</td>
<td>16</td>
<td>22</td>
<td>32</td>
<td>20</td>
<td>63</td>
<td>75</td>
<td>101</td>
<td>124</td>
<td>146</td>
<td>155</td>
<td>174</td>
<td>151</td>
<td>182</td>
<td>193</td>
<td>196</td>
</tr>
<tr>
<td>Total No. of workers</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>2</td>
<td>12</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>28</td>
<td>28</td>
<td>21</td>
<td>30</td>
<td>26</td>
<td>37</td>
<td>32</td>
<td>28</td>
</tr>
</tbody>
</table>
number of detected faults grows smoothly with the exercise of test instances.

We have also shown ways of improving Model-1, i.e., the segmentation technique and the composite estimation. The application of the segmentation technique (Model-2) looks quite effective, particularly when the growth curve of the cumulative number of detected faults bends sharply.

In determining the test intervals, we noted in part that the sharp bend occurred a little later than the time of the change of test classes. We observed similar delays in other examples. Thus, some mechanisms should be incorporated in the segmentation technique so that we could determine test intervals only by observing the change of test classes.

Further, test instances of a test class are not necessarily exercised in succession. In such a case, the division of the whole test duration into successive test intervals may not be justified. Some rearrangement of test instances will be required.

Here, we considered only the number of test workers in defining \( w(i)/w(i, j) \). \( w(i)/w(i, j) \) could be improved, if additional factors such as the number of test items in a test instance and/or the progress of the skill in exercising test instances were taken into account. We are currently conducting further studies on this line.

The composite estimation looks promising. As we see in Table IX, however, the number of test instances for each test class was rather small to make the estimation properly. Although we hope the composite estimation as well as the segmentation technique will be effective in
cases where the plot curve of \( C(i) \) bends sharply, further investigation with more data of test instances will be needed to evaluate our models definitely.

The composite estimation and the segmentation technique should be combined with each other.

ACKNOWLEDGMENT

We express our thanks to Railway Technical Research Institute, Japan National Railways, and Toshiba Corporation for their cooperation and support in providing us with real data.

REFERENCES


Dr. Tohma is an Advisory Board Member of TC on Fault-Tolerant Computing, IEEE Computer Society, and Vice-Chairman of IFIP WG10.4. He served as Program Chairman of FTCS-18 and will serve as Conference Chairman of FTCS-19 to be held in Tokyo in 1988. He is a member of the Information Processing Society of Japan, the Japan Society for Software Science and Technology, the Institute of Electronics, Information, and Communication Engineers (IEICE), and other technical societies in Japan. Among other awards from IEICE, he received the Excellent Book Award and the Okabe Memorial Award.

Kenshin Tokunaga was born in Yamaguchi, Japan, on February 11, 1961. He received the B.Eng. and M.Eng. degrees in computer science from Tokyo Institute of Technology, Tokyo, Japan, in 1983 and 1985, respectively. Since 1987, he has been with the Department of Computer Science, Tokyo Institute of Technology. His current research interests include natural language processing and knowledge representation.

Mr. Tokunaga is a member of the Information Processing Society of Japan and the Japan Society for Software Science and Technology.

Shinji Nagase was born in Kuwana, Japan, on February 28, 1962. He received the B.Eng. and M.Eng. degrees in computer science from Tokyo Institute of Technology, Tokyo, Japan, in 1984 and 1986, respectively.

Since 1986, he has been associated with Chubu Electric Power Company, Inc. His research interests include fault-tolerant computing and software engineering.

Yukihisa Murata was born in Hyogo, Japan, on October 23, 1954. He received the B.Eng. degree in information engineering from Kyoto University, Kyoto, Japan, in 1977.

Since 1977, he has been engaged in designing and developing operating systems of general purpose computers at Mitsubishi Electric Corporation. His research interests include operating systems, computer architecture, and software engineering.

Mr. Murata is a member of the Information Processing Society of Japan.