A Systematic-Testing Methodology for Software Systems

SEHEON HWANG and HOANG PHAM

Department of Industrial and Systems Engineering, Rutgers University, Piscataway, New Jersey USA

(Received on January 25, 2006)

Abstract: In this paper, a software testing methodology called two-level testing is developed to improve testing effectiveness by reducing the testing efforts and at the same time, by ensuring the predetermined quality of software products. The testing procedure including the criteria for alternating 100% and sampling testing is proposed by incorporating the characteristics of testing behavior into the well-known sampling method. The metrics of the testing performance are derived based on the transition probability. Various combinations of controllable parameters ensuring equivalent quality are also provided for the purpose of effective application. A numerical example is provided to illustrate the testing effectiveness of the proposed testing method.

Key Words: software testing, system test, continuous sampling plan, two-level testing

1. Introduction

As the functionality of today’s software applications becomes more critical and complicated, the modern commercial software products have become larger in size, which typically contain millions of lines of code [1, 2]. In addition, the increasing complexity of the program along with the large size requires more time and effort for its development. The decreasing average life expectancy as well as early market arrival before competitors may encourage, more than ever, the use of less time and resources.

Among the software development activities, testing is considered as the most challenging process from the developers’ point of view since it is commonly known that cost estimates the finding and removing of program faults ranges from 40% to 80% of total development cost [3]. Testing also plays an important role to ensure its reliable operation by finding and removing the faults from the program. In particular, system test is one of the most important processes in software development since it can finally verify and validate the products before release to the customers [4, 5].

* Communicating author’s email: hopham@rci.rutgers.edu
In addition, system test provides the failure-time data so that future failures that might occur during the operational phase can be predicted. For developing a large software system under today’s circumstances, insufficient time and limited budget often lead software developers to make hard decisions on how to allocate the time and resources for software testing [6, 7]. Delays in software release can possibly happen during the software development process when the development team spends most of scheduled time on solving unexpected technical problems or more often, on modifying the program to add new functions requested later by marketing team or customers. This may result in deficiency of the time for system test which usually starts only after the entire system has been integrated. The developer’s decision is getting harder if the penalty cost incurred by the manufacturer for the delay in software release is huge or early market arrival before competitors may be the most critical issue for the success.

Software developers, therefore, are obliged to choose either keeping the delivery time by reducing the testing efforts or suffering great losses by conducting all the scheduled testing. In the scenario to reduce the testing efforts, selective testing based on a certain sampling method is one of the possible ways to be chosen, but, it can bring about concerns that cannot ensure the quality of the software product when comparing it with one conforming to the scheduled testing. This, as the result, may cause the failure in delivering quality products, which result in loss of reputation and loss of future business for the company.

This paper proposes a systematic approach for the performance of system test which aims to not only reduce the testing effort by executing a part of scheduled test cases but also ensure a competitive product quality by choosing a set of predetermined parameters. We develop a testing procedure including the criteria for alternating 100% and sampling testing by incorporating the characteristics of software testing into the well known continuous sampling plan (CSP), usually used in the quality control area. We then derive several metrics of performance for the proposed testing method. Even though the testing process conducted is only for partially selected test cases, it guarantees the percentage of remaining defects will not exceed a predetermined value as it reduces testing efforts in terms of testing time.

A brief review on various CSPs is presented in the following section. Detail procedure and transition policy of the proposed testing is discussed in section 3. The model formulation is obtained based on transition probability and the metrics of testing performance are derived in section 4. Analytical assessments are discussed in section 5 and a numerical example is given to illustrate the testing effectiveness in section 6 and the conclusions are drawn in section 7.

2. Literature Review

Since Dodge [8] introduced the first continuous sampling plan designated as CSP-1, many research papers have been published to deal with the extension of CSP-1 plan [9, 10, 11, 12, 13, 14]. Dodge and Torrey [10] subsequently introduced CSP-2 and CSP-3 plan, in which the transition into 100 % inspection defers until finding other evidence of poor quality in a defective unit. Lieberman and Solomon [12] presented multi-level CSP plans that allows for any number of sampling levels, subject to the provision that transition can only occur between adjacent levels. All the above variations of the CSP-1 plan aim to
reduce inspection effort, by introducing more than one sampling frequency. The average outgoing quality limit (AOQL) is one of the indices to measure the CSP-1 plan and other continuous sampling plans.

In order to apply CSP to the plants with very high production rates, Wang and Chang [15] have proposed a new continuous sampling plan, referred to as CSP-C-1. Govindaraju and Kandasamy [11] also proposed the generalized CSP-C plan to achieve a reduction in the average fraction inspected at good quality levels. The tightened CSP plans have been developed by several researchers to reduce the amount of units inspected compared with other CSP plans, when the incoming fraction of defective units is relatively small [13, 16, 17]. Derman et al. [13] proposed tightened multi-level CSP where transition can occur between more than 1 level (generally r levels) when a defective unit is found in any sampling level. Govindaraju & Balamurali [17] have proposed the tightened single-level continuous sampling plan designated as TCSP-1 plan to lower the AOQL by increasing the screening inspection.

The procedure of the TCSP-1 plan is the same as that of the CSP-1 plan except that, during the sampling phase, once k inspected units are found that conform, the sampling inspection is stopped and 100% inspection restarted. They point out the advantage of the TCSP-1 plan is that it provides consumer protection against poor process quality and improved administration of the plan. Balamurali and Govindaraju [14] also proposed a modified tightened 2-level continuous sampling plan based on the existing tightened multi-level continuous sampling plan of Derman et al. [13].

3. The Procedure of Two-level Testing

The two-level testing performs 100% testing and sampling testing alternately in accordance with the frequency of failure occurrence. Although our method is constructed based on the existing CSP, some notable features draw a distinction between the proposed plan and the CSP plans. First, there are four different characteristics in two-level testing that lead the conversion in the 100% testing phase and each level of the sample testing phases, whereas there are basically only two, defective or non-defective throughout all the variations and modifications of CSP plans. In two-level testing, the software failures detected are classified as minor, major and critical according to the severity of impact on its functional features. Secondly, though a failure occurs during level 2 sample testing phase, the next transition changes by the type of faults. Such a transition policy cannot be found among any other CSPs.

According to Lieberman and Solomon [12] the main objective of multi-level CSP is to reduce the amount of units inspected compared with the single level when the incoming fraction of defective is relatively small. We basically design the sampling testing phase to have two different sampling frequencies, the simplest case in the possible multi-level plans, with intent of improving sampling effectiveness as mentioned above. For the transition policy, both 100% testing and level 1 sample testing phase have the same value of consecutive number to continue testing without failure when they convert to the next sampling testing phases.

The proposed testing method consists of three testing phases: a 100% testing, level 1 testing, and level 2 testing, as discussed follows:
I. 100% Testing Phase

1. At the beginning, select a test case at random from the scheduled test cases and test one at a time. It continues until obtaining \(i\) consecutive number of test cases without failure.

2. When \(i\) consecutive test cases are obtained without failure, then 100% testing is stopped and switched to level 1 testing.

II. Level 1 Testing Phase (the frequency of testing is \(f_1\))

1. In level 1 testing phase, the frequency of selection is \(f_1\) (< 1). Technically, select one test case at random from \(1/f_1\) of scheduled test cases and test it. The rests are passed on without replacement. This is defined as the level 1 testing phase.

2. When \(i\) consecutive test cases are obtained without failure during the level 1 testing phase, then, level 1 testing is stopped and switched to level 2 testing. If a failure occurs before obtaining \(i\) consecutive test cases without failure, then, the 100% testing is resumed regardless of the types of faults.

III. Level 2 Testing Phase (the frequency of testing is \(f_2\))

1. In level 2 testing phase, the frequency of selection is \(f_2\) (< \(f_1\) < 1). Select one test case at random from \(1/f_2\) of the scheduled test cases and test it. The rests are passed on without replacement. This is defined as the level 2 testing phase and continued until a failure is found.

2. If a critical fault is detected during level 2 testing, 100% testing is resumed immediately. If the detected fault is either minor or major, then, level 2 testing is switched back to level 1 testing.

The flow chart, which describes the procedure for a two-level testing plan is given in Fig. 1. As we can see, the proposed plan has a specific feature and the type of detected fault plays an important role in determining the next testing phase. In the existing CSP plans, the criteria for decision is labeled as just ‘non-defective’ or ‘defective’ as Dodge [8] stated on a “Go-NoGo” basis, whereas the decision in the proposed method is classified into four criteria including three different types of faults in the software program according to the difficulty for correcting them. The types of faults can be defined, for example, as:

- Minor fault: Suggestion for improvement, e.g., cosmetic issues.
- Major fault: Minor deviation in the functionality.
- Critical fault: Critical function is not working.

4. Methodology Formulation

In software engineering, many researchers have been focusing on how to select the test set or test cases from the entire input domain to build a more reliable software product by finding the failure region more effectively [18]. The proposed testing methodology, however, assumes that the test cases are already selected using the traditional methods such as random method or subdomain method. Therefore, we restrict our consideration on how to reduce the testing effort more effectively by selecting a portion of the pre-determined test cases. Our formulation is based on the following two assumptions: (1) the probability of failure occurrence is constant, and (2) perfect debugging is assumed.
The first assumption’s basis is as follows: A group of scheduled test cases, which consists of \( N \) test cases at the beginning of system testing. Assume that there are \( m \) failure-causing test cases or a proportioned rate of failure \( p = m/N \) within the group of test cases, then, we can deduce that \( p \) exists when we take a random sample size \( r \) whether with and without replacement policies. First, we can sample sequentially without replacement.

**Fig. 1: Flow chart for Two-level testing**

Let \( Y_i \) be the number of successes on the \( i^{th} \) trial. The \( Y_i \) are binary valued since upon each draw we may conclude either with a success (\( Y_i = 1 \)) or a failure (\( Y_i = 0 \)). In this case, the
Y_i are not independent so the trials are not Bernoulli trials. Clearly \( P(Y_1 = 1) = r/N = p = \) the proportion of successes in the population. Let us now consider \( Y_2 \). We have

\[
P(Y_2 = 1 | Y_1 = 1) = \frac{r - 1}{N - 1}, \quad P(Y_2 = 1 | Y_1 = 0) = \frac{r}{N - 1}
\]

From Bayes' theorem,

\[
Pr(Y_2 = 1) = Pr(Y_2 = 1 | Y_1 = 1) Pr(Y_1 = 1) + Pr(Y_2 = 1 | Y_1 = 0) Pr(Y_1 = 0)
\]

\[
= \frac{r - 1}{N - 1} \cdot \frac{r}{N} + \frac{r}{N - 1} \cdot \frac{N - r}{N} = \frac{r}{N} = p
\]

By induction we can show that \( Pr(Y_i = 1) = r/N = p \) for all trials.

**Notation**

- \( p \) probability of failure occurrence by executing a test case at random
- \( q = 1 - p \)
- \( p_1 \) probability of failure occurrence due to minor fault
- \( p_2 \) probability of failure occurrence due to major fault
- \( p_3 \) probability of failure occurrence due to critical fault
- \( i \) consecutive number of test cases causing no failure obtained until switching to the next level, integer
- \( f_1 \) frequency of selection in level 1 testing phase
- \( f_2 \) frequency of selection in level 2 testing phase
- \( FTT \) long run percentage of test cases that have been executed
- \( FTN \) long run percentage of test cases that have not been executed
- \( PRD \) long run percentage of remaining defects
- \( PRD_{max} \) threshold value of \( PRD \)

We now model the two-level testing plan to determine the performance metrics using transition probabilities of all possible states. Fig. 2 depicts the transition states which are mutually exclusive. The definitions of all states are discussed below.

**Definitions of the model state**

- \( A_j \) 100% testing is being performed. The consecutive number of test cases that has been tested without failure is \( j \).
- \( Ig_0 \) Level 1 testing is performed. A selected test case was tested without failure, as the result, \( j \) consecutive number of test cases has been tested without failure so far.
- \( Ie_j \) Level 1 testing is performed. A selected test case was tested and caused a failure. \( j \) consecutive number of test cases have been tested without failure thus far.
- \( N_j \) Level 1 testing is performed. The test case selected was not tested due to the feature of sampling testing and still \( j \) consecutive number of test cases has been tested without failure.
- \( I^g_k \) Level 2 testing is performed. The test case selected was tested without failure.
- \( I^e_k \) Level 2 testing is performed. The test case selected was tested and caused a failure due to a minor or major fault.
Level 2 testing is performed. The test case selected was tested and caused a failure due to critical fault.

Level 2 testing is performed. The test case selected was not tested due to the feature of sampling testing.

Based on the transition diagram in Fig. 2, the steady-state probability of each state is formulated as follows:

\[ P(A_i) = p[P(A_i) + P(A_j) + \sum_{j=2}^{i-1} P(A_j) + P(Ie_j) + \sum_{j=1}^{i-1} P(Ie_j) + P(I^2\epsilon^c)] \]  \( \text{(1)} \)

\[ P(A_i) = q[P(A_i) + P(Ie_i) + \sum_{j=1}^{i-1} P(Ie_j) + P(I^2\epsilon^c)] \]  \( \text{(2)} \)

\[ P(A_i) = q[P(A_{i-1})] \text{ for } j = 2, 3, 4, \ldots, i \]  \( \text{(3)} \)

\[ P(Ig_i) = f_i q[P(A_i) + P(N_0) + P(I^2\epsilon)] \]  \( \text{(4)} \)

\[ P(Ig_j) = f_i q[P(Ig_{j-1}) + P(N_{j-1})] \text{ for } j = 2, 3, 4, \ldots, i \]  \( \text{(5)} \)

\[ P(N_0) = (1 - f_i)[P(A_i) + P(N_0) + P(I^2\epsilon)] \]  \( \text{(6)} \)

\[ P(N_j) = (1 - f_i)[P(Ig_j) + P(N_j)] \text{ for } j = 1, 2, 3, \ldots, i-1 \]  \( \text{(7)} \)

\[ P(Ie_0) = f_i p[P(A_i) + P(N_0) + P(I^2\epsilon)] \]  \( \text{(8)} \)

\[ P(Ie_j) = f_i p[P(Ig_j) + P(N_j)] \text{ for } j = 1, 2, 3, \ldots, i-1 \]  \( \text{(9)} \)

\[ P(I^2g) = f_2 q[P(Ig_0) + P(I^2g) + P(N^2)] \]  \( \text{(10)} \)

\[ P(N^2) = (1 - f_2)[P(Ig_0) + P(I^2g) + P(N^2)] \]  \( \text{(11)} \)

\[ P(I^2\epsilon) = f_2 (p - p_3)[P(Ig_0) + P(I^2g) + P(N^2)] \]  \( \text{(12)} \)

\[ P(I^2\epsilon^c) = f_2 p_3[P(Ig_0) + P(I^2g) + P(N^2)] \]  \( \text{(13)} \)

The sum of the probabilities for all states is obviously equal to 1, that is

\[ \sum_{i=0}^{1} P(A_i) + \sum_{j=1}^{i} P(Ig_j) + \sum_{j=0}^{i} P(Ie_j) + \sum_{j=1}^{i} P(N_j) + P(I^2g) + P(N^2) + P(I^2\epsilon) + P(I^2\epsilon^c) = 1 \]  \( \text{(14)} \)

Solving the (1) \- (14), we obtain the probabilities of all states expressed in terms of \( P(Ig_1) \). See Hwang and Pham [19] for detailed derivations. Then, by substituting all equations into (14), we can get the following result.

\[ P(Ig_0) = \frac{f_1 f_2 p q^{i+1}}{f_1 f_2 (1-q^i)(p-(p-p_3)q^i) + (f_2 + (f_1-f_2)q^i) pq^i} \]  \( \text{(15)} \)
The steady-state probabilities of all the states which have been already formulated as the functions of \( P(Ig_1) \) and \( P(A_1) \) can be now written in terms of controllable parameters \( i, f_1, f_2, p, \) and \( p_3 \). Let \( P(100\%) \) be the long run probability that testing is conducted in 100% testing phase, then,

\[
P(100\%) = P(A_0) + P(A_1) + \sum_{j=2}^{i} P(A_j) = \frac{f_1 f_2 (p - (p - p_3)q^j)(1 - q^j)}{D}
\]

where \( D = f_1 f_2 (1-q^j)(p - (p - p_3)q^j) + (f_2 + (f_1 - f_2)q^j)pq^j \).

The long run probability that testing is conducted in level 1 sampling testing phase is,

\[
P(\text{Level 1 testing}) = \sum_{j=1}^{i} P(Ig_j) + P(1e_0) + \sum_{j=1}^{i} P(Ie_j) + P(N_0) + \sum_{j=1}^{i} P(N_j) = \frac{f_2 p (1 - q^j)q^j}{D}
\]

Likewise, long run probability testing conducted in the level 2 testing phase is,
\[ P(\text{Level 2 testing}) = P(I^2 g) + P(N^2) + P(I^2 e) + P(I^2 e') = \frac{f_i p q^{2i}}{D} \] (18)

Now, we can derive the metrics of performance of this plan in terms of the controllable parameters, \(i, f_1, f_2, p\) and \(p_3\). First, the long run percentage of test cases that is not executed is obtained as follows:

\[ FTN = P(N_a) + \sum_{j=1}^{i} P(N_j) + P(N^2) \]
\[ = \frac{pq^i \{(1 - f_1) f_2 + (f_1 - f_2) q'\}}{D} \] (19)

The long run percentage of test cases executed is

\[ FFTT = 1 - FTN = \frac{f_1 f_2 \left[ p - (p - p_3) q' (1 - q') \right]}{D} \] (20)

Hence, \(PRD\), the percentage of remaining defects in the software product after completing this testing plan, can be obtained as follows:

\[ PRD = p \cdot (FTN) \]
\[ = p \left\{ \frac{pq^i \{(1 - f_1) f_2 + (f_1 - f_2) q'\}}{D} \right\} \] (21)

5. Discussion

The measures of performance derived in the previous section are expressed as the function of \(i, f_1, f_2, p\), and \(p_3\). The variable \(p\) and \(p_3\) are basically unknown which are dependent on the various factors of the development process such as program complexity, programmers’ skill, etc. Therefore, we start applying the proposed testing method by determining the pertinent values of \(i, f_1,\) and \(f_2\), which are called controllable parameters. Before discussing how to apply the proposed method for system test in depth, we need to analyze the relationship among the controllable parameters and the performance measures in the next section so as to obtain the benefits of the proposed method by choosing appropriate combination of the controllable parameters.

Effects of controllable parameters on \(PRD\)

The percentage of remaining defects, \(PRD\), implies how reliable the software product is when two-level testing is completed. Since it is formulated by multiplying \(FTN\), the long run percentage of test cases that is not executed, by probability of failure occurrence, \(p\), as seen in (21), the plot of \(PRD\) with respect to \(p\) shows a curved line that has a maximum point, called \(PRD_{\text{max}}\) at a specific value of \(p\).

The shape of the \(PRD\) curve can vary with the values of controllable parameters. Many combinations of these controllable parameters with the same value of \(PRD_{\text{max}}\) can be obtained even though they may have different shapes of width and slope. Figure 4 shows the \(PRD\) curves of several sets of \(i, f_1,\) and \(f_2\) with same \(PRD_{\text{max}}\) value. Each combination of \(i, f_1,\) and \(f_2\) selected in this figure can ensure that the percentage of remaining defects...
does not exceed $PRD_{\text{max}}$ value no matter what the quality of the initial product is. If $p$ is very small, it is difficult for testers to detect the faults by executing only a portion of test cases. Accordingly, the undetected faults are never corrected, which leads the value of $PRD$ higher. As $p$ is increased, however, more faults are detected and as the result, they are corrected, which makes $PRD$ decrease with the increase of $p$ from the point of $PRD_{\text{max}}$.

The relatively small value of $i$ makes the curve wider, which implies that testing effort is less sensitive to the change of $p$. Table 2 provides the various sets of $i$, $f_1$, and $f_2$ values and the corresponding $PRD_{\text{max}}$ assuming $f_2 = 0.5f_1$ and $p_3 = 0.1p$. This table might help the practitioners determine the pertinent values of controllable parameters when the level of quality has been specified before testing.

Effects of controllable parameters on $FTT$

$FTT$ can be considered as an important criterion to measure the performance of the proposed testing method because it represents the percentage of test cases conducted during the testing work. $FTT$ is also impacted by the controllable parameters just as $PRD$, thus, it would be helpful to find the effects for each of the controllable parameters on $FTT$, especially when we focus on reducing testing efforts. As we can see in Fig. 3 and Fig. 4, $f_1$ and $f_2$ influence the degree of increasing rate around the range of large $p$ and small $p$, respectively, while the integer number $i$ impacts on the overall slope of $FTT$. Fig. 4 also shows the shapes of $FTT$ in terms of:

(a) $f_2$ when $i$ and $f_1$ are fixed and
(b) $f_1$ when $i$ and $f_2$ are fixed, with respect to $p$.

We also can see that $f_2$ determines the contact on a vertical axis. Fig. 5 depicts the change of $FTT$ with three different sets of $i$, $f_1$, and $f_2$ values that have the same $PRD_{\text{max}}$ as 0.01. At small $p$, in this case for instance, is below 0.0167 and the $FTT$ is getting lower as the value of $i$ is larger. While increases are more rapid with the increase of $p$ as the value of $i$ is larger. Thus, if a software product is assured such a small value of $p$, then, the larger value of $i$ makes $FTT$ be much smaller, which means that testing can be completed by executing small amount of test cases. On the other hand, the small value of $i$ should be employed when $p$ is expected to be relatively large.

5.1 Advantageous Application

Zhang and Pham [20] have pointed out the specific results of their survey investigations to identify the factors that may impact software reliability. In their research, the top five factors that significantly impact software reliability were found to be program complexity, programmers’ skills, testing coverage, testing effort, and testing environment. These results provide important aspects on how to apply the two-level testing plan for the practical software testing process.
It is common for a large software product to be programmed by several programmers. According to the survey results, each computer program constructed by different skilled programmers might have different probability of failure occurrence when it is tested. For instance, if the scheduled test cases are divided into number of subgroups according to the programmers, or its functional complexity, etc., then, each subgroup of test cases may have different value of $p$. 

Fig. 3: PRD Curves of Several Sets of $i$ and $f_1$ with same $PRD_{\text{max}}$ (Assuming $f_2 = 0.5 f_1$ and $p_3 = 0.1 p$)

Fig. 4: FTT in Terms of (a) $f_2$ when $i$ and $f_1$ are Fixed (b) $f_1$ when $i$ and $f_2$ are Fixed
Due to the properties of two-level testing, the subgroup with smaller $p$ experienced less testing effort while the one with the larger $p$ has performed more testing effort. In other words, more faults are found and removed from the subgroup with higher $p$ by executing more test cases. Therefore, we can achieve benefits from the two-level testing plan by applying the proposed method to each subgroup of test cases independently whenever the identical combination of the controllable parameters is used. We next show a numerical example and compare the testing effectiveness of the proposed method with the traditional ones.


Two-level testing has originally been proposed as an alternative testing method for a large software product when the scheduled testing is not allowed due to some restrictions, as the result, sampling testing is assumed to be the only way under the circumstance. Therefore, in order to present the benefits of the proposed testing method, we first need to consider another type of testing method to be compared, the partition testing, which is known to guarantee a better chance of detecting at least one failure than random testing [21, 22].

Consider a set of test cases for a large program that can be divided into five parts in accordance with programmers for system testing that will confirm the functionality of the integrated program. Based on the survey [20], each part is assumed to have different probability of failure occurrence when a test case is executing at random. In practice, it is not possible for us to be aware of the probability of failure occurrence for each part until the testing process is completed. For the purpose of comparison, however, we assumed the probability of failure occurrence for each part as shown in Table 3. The metrics of performance for the proposed testing method are obtained by using the formula derived in
Section 4. The desired quality for the final product is also assumed never to drop below 0.99, in other words, we determine the $PRD_{\text{max}}$ as 0.01. Among various choices of the combination in selecting $i, f_1,$ and $f_2$ that has the same $PRD_{\text{max}}$ of 0.01, we choose $i, f_1,$ and $f_2,$ to be 121, 0.1, and 0.05, respectively. Once these controllable parameters are determined, the performance measures containing $PRD$, $FTT$, and $FTN$ are only the function of $p$ and $p_j$.

Benefit of low $PRD$

Suppose each test case has a fraction of size, such as 0.3, 0.2, 0.1, 0.2, and 0.2, from part I to part V, respectively. The percentage of remaining defects in the software product, when the proposed testing is applied, is given as

$$PRD_{\text{two-level}} = \sum_{j=1}^{V} F_j \cdot PRD(p_j)$$

$$= 0.3(0.0023) + 0.2(0.0046) + 0.1(0.0081) + 0.2(0.0099) + 0.2(0.0025)$$

$$= 0.00490$$ (22)

where $F_j$ is a fraction of the size for part $j$ and $p_j$ is the probability of failure occurrence for part $j$.

Due to the property of two-level testing that the subgroup with smaller $p$ experienced less testing effort while the one with the larger $p$ has performed more testing effort, each part results in the different $FTT$ value as shown in Table 3. As the results of numerical analysis, the overall percentage of test cases that have been executed is calculated at 0.3115 and the percentage of remaining defects of the software product through this plan, $PRD_{\text{two-level}}$, is found at 0.00490.

An equivalent percentage of test cases is assumed to be executed when the partition testing method is applied. In this case, the overall percentage of test cases that have been executed, $FTT_{\text{two-level}} = FTT_{\text{partition}} = 0.3115$, is applied for each part regardless of its probability of failure occurrence, $p$. Therefore, the percentage of remaining defects obtained by the partition testing method is given below

$$PRD_{\text{partition}} = \sum_{j=1}^{V} F_j \cdot p_j \cdot (1 - FTT_{\text{partition}})$$ (23)

Accordingly, the numerical result of Eq. (24) is as follows:

$$PRD_{\text{partition}} = \sum_{j=1}^{V} F_j \cdot p_j \cdot (1 - FTT_{\text{partition}})$$

$$= (0.3)(0.0025)(0.6885) + (0.2)(0.005)(0.6885) + (0.1)(0.010)(0.6885) + (0.2)(0.015)(0.6885) + (0.2)(0.040)(0.6885)$$

$$= 0.00947$$ (24)

In this example, the results shows that the application of two-level testing plan can reduce the percentage of remaining defect in the software product from 0.00947 to 0.00490,
which brings a 93% improvement in software quality, when compared to the partition testing strategy with the same testing efforts.

Table 3. The Performance Measures of Two-Level Testing Plan when \( i = 121, f_1 = 0.1, f_2 = 0.05 \) (assuming \( p_3 = 0.1p \))

<table>
<thead>
<tr>
<th>Fraction of size</th>
<th>( p_1 )</th>
<th>( p_3 )</th>
<th>Two-level Testing</th>
<th>Partition testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>( FTT ) ( PRD )</td>
<td>( FTT ) ( PRD )</td>
</tr>
<tr>
<td><strong>Part I</strong></td>
<td>0.3</td>
<td>0.0025</td>
<td>0.0025</td>
<td>0.0639 0.0023</td>
</tr>
<tr>
<td><strong>Part II</strong></td>
<td>0.2</td>
<td>0.0050</td>
<td>0.0050</td>
<td>0.0897 0.0046</td>
</tr>
<tr>
<td><strong>Part III</strong></td>
<td>0.1</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.1864 0.0081</td>
</tr>
<tr>
<td><strong>Part IV</strong></td>
<td>0.2</td>
<td>0.0150</td>
<td>0.0150</td>
<td>0.3403 0.0099</td>
</tr>
<tr>
<td><strong>Part V</strong></td>
<td>0.2</td>
<td>0.0400</td>
<td>0.0400</td>
<td>0.9387 0.0025</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>1.0</td>
<td></td>
<td>0.0049</td>
<td>0.3115 0.0095</td>
</tr>
</tbody>
</table>

7. Conclusion

In this study, a systematic approach to perform system testing, called two-level testing plan, for large software systems is proposed based on a continuous sampling plan. In addition to developing a testing procedure which includes the criteria for alternating 100% and sampling test by incorporating the characteristics of software testing into CSP, we also derived the metrics of testing performance such as long run percentage of test cases that have been executed and long run percentage of remaining defects using the transition probability.

An advantageous application of the proposed testing plan is obtained when a set of scheduled test cases is divided into several parts, which results from the feature that the one with smaller \( p \) will experience less testing effort while the one with larger \( p \) will receive more testing effort. A numerical example is provided to illustrate the results, and consequently in this specific example, the two-level testing method can be fairly effective in ensuring the predetermined quality of the software products as well as reducing the testing efforts when compared with one of the traditional sampling testing methods.
Table 2: Various Sets of $i, f_1$, and $f_2$ Values and the Corresponding $PRD_{max}$. Assuming $f_2 = 0.5f_1$ and $p_3 = 0.1p$.

<table>
<thead>
<tr>
<th>$f_i$</th>
<th>0.002</th>
<th>0.003</th>
<th>0.004</th>
<th>0.005</th>
<th>0.006</th>
<th>0.007</th>
<th>0.008</th>
<th>0.009</th>
<th>0.010</th>
<th>0.012</th>
<th>0.014</th>
<th>0.016</th>
<th>0.020</th>
<th>0.030</th>
<th>0.040</th>
<th>0.050</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.500 (1/2)</td>
<td>225</td>
<td>149</td>
<td>112</td>
<td>90</td>
<td>75</td>
<td>64</td>
<td>56</td>
<td>50</td>
<td>44</td>
<td>37</td>
<td>32</td>
<td>28</td>
<td>24</td>
<td>22</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>0.333 (1/3)</td>
<td>314</td>
<td>209</td>
<td>156</td>
<td>125</td>
<td>104</td>
<td>89</td>
<td>78</td>
<td>69</td>
<td>62</td>
<td>52</td>
<td>44</td>
<td>39</td>
<td>34</td>
<td>31</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>0.250 (1/4)</td>
<td>380</td>
<td>253</td>
<td>189</td>
<td>151</td>
<td>126</td>
<td>108</td>
<td>94</td>
<td>84</td>
<td>75</td>
<td>63</td>
<td>54</td>
<td>47</td>
<td>42</td>
<td>37</td>
<td>25</td>
<td>18</td>
</tr>
<tr>
<td>0.200 (1/5)</td>
<td>433</td>
<td>288</td>
<td>216</td>
<td>173</td>
<td>144</td>
<td>123</td>
<td>108</td>
<td>96</td>
<td>85</td>
<td>71</td>
<td>61</td>
<td>53</td>
<td>47</td>
<td>43</td>
<td>28</td>
<td>21</td>
</tr>
<tr>
<td>0.167 (1/6)</td>
<td>478</td>
<td>318</td>
<td>238</td>
<td>190</td>
<td>159</td>
<td>136</td>
<td>119</td>
<td>105</td>
<td>95</td>
<td>79</td>
<td>67</td>
<td>59</td>
<td>52</td>
<td>47</td>
<td>31</td>
<td>23</td>
</tr>
<tr>
<td>0.143 (1/7)</td>
<td>517</td>
<td>344</td>
<td>258</td>
<td>206</td>
<td>172</td>
<td>147</td>
<td>128</td>
<td>114</td>
<td>103</td>
<td>85</td>
<td>73</td>
<td>64</td>
<td>57</td>
<td>51</td>
<td>34</td>
<td>25</td>
</tr>
<tr>
<td>0.125 (1/8)</td>
<td>551</td>
<td>367</td>
<td>275</td>
<td>220</td>
<td>183</td>
<td>157</td>
<td>137</td>
<td>122</td>
<td>109</td>
<td>91</td>
<td>78</td>
<td>68</td>
<td>60</td>
<td>54</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>0.111 (1/9)</td>
<td>582</td>
<td>387</td>
<td>290</td>
<td>232</td>
<td>193</td>
<td>165</td>
<td>145</td>
<td>128</td>
<td>116</td>
<td>96</td>
<td>82</td>
<td>72</td>
<td>64</td>
<td>57</td>
<td>38</td>
<td>28</td>
</tr>
<tr>
<td>0.100 (1/10)</td>
<td>609</td>
<td>406</td>
<td>304</td>
<td>243</td>
<td>203</td>
<td>173</td>
<td>152</td>
<td>135</td>
<td>121</td>
<td>101</td>
<td>86</td>
<td>75</td>
<td>67</td>
<td>60</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>0.083 (1/12)</td>
<td>659</td>
<td>439</td>
<td>329</td>
<td>263</td>
<td>219</td>
<td>188</td>
<td>164</td>
<td>146</td>
<td>131</td>
<td>109</td>
<td>93</td>
<td>81</td>
<td>72</td>
<td>65</td>
<td>43</td>
<td>32</td>
</tr>
</tbody>
</table>
References


Seheon Hwang received his M.S. degree in Materials Engineering from Hanyang University, Korea and M.S. Industrial Engineering from Rutgers University, New Jersey. He is completing his PhD in the Department of Industrial and Systems engineering at Rutgers University. His research interests include reliability modeling with an emphasis toward software system and software development engineering.

Hoang Pham is a Professor in the Department of Industrial and Systems Engineering at Rutgers University. Before joining Rutgers, he was a senior engineering specialist at the Boeing Company, Seattle, and the Idaho National Engineering Laboratory, Idaho Falls. His research interests include software reliability, system reliability modeling, maintenance, and environmental risk assessment. He is the author of two books: *Software Reliability* (Springer-Verlag, 2000) and *System Software Reliability* (Springer, 2006), and editor of the *Handbook of Reliability Engineering* (Springer-Verlag, 2003) and has published more than 90 journal articles. He is editor-in-chief of the *International Journal of Reliability, Quality and Safety Engineering*, associate editor of the *IEEE Trans. on Systems, Man and Cybernetics*, and also the editor of Springer Series in Reliability Engineering. He is a fellow of the IEEE.