Reliability Growth Models for Software Testing Against Security Requirements

Shiv Kumar Sharma (Scholar)
B S A College Mathura, U.P. India

The use of the mathematical models must not distract the experts from the painstaking and responsible real software testing helping mainly at the decision making points. That is why it will be practical to classify the models in accordance with the input statistics at different life cycle stages of the software i.e. Test confidence models, Software complexity models, Reliability growth models, and Debugging models. Within Object Orientation, the Unified Modeling Language is the standard language adopted by the Object Management Group to analyze and design information systems. However, Unified Modeling Language has been criticized since its appearance due to the ambiguity and the lack of a truly formal definition of its semantics. Something which is crucial in the aerospace industry, given the high level of reliability that these systems require. Within this formal framework, specifications of software concerning aerospace systems constructed by Unified Modeling Language class diagrams can be transformed into equivalent formal representations. Thus, the diagram can be mathematically verified and manipulated by using its equivalent formal representation.

Keywords: Software complexity models, Reliability growth models, Debugging models and Control model.

1. Introduction

Currently Object Orientation is one of the most prominent approaches in software development projects. Within this approach, the Unified Modeling Language (UML) is the standard language adopted by the Object Management Group to specify and develop object-oriented information systems. The situation is due to the high structural complexity of the software systems, and dynamism of versions and technologies. The different life cycle stages of the software are as follow:

Test confidence models helping to estimate confidence in the software conformance evaluation;
Software complexity models helping to estimate the software complexity as well as related quality and security;
Reliability growth models helping to estimate software technological security depending on the testing time;
Debugging models helping to estimate software technological security depending on the input data runs in set data areas and following debugging processes.


2. Mathematical Model

The mathematical model is not shown in detail, but the aspects that we consider necessary to understand the specification are included. More details can be found in a related technical report. The approach followed in this paper is based on the different model and it is in agreement with the four-layer modelling architecture on which the definition is based as follow:

2.1 Errors Model

Introduced errors consideration model, also known as Labeled Fishes Problem of the probability theory, or the Mills model, is based on test errors introduced into the code. In testing, statistical data is collected on real and introduced errors revealed. Test errors are supposed to be introduced stochastically, to make the identification of both own and introduced errors equally probable. In this case, we can estimate the original number of the software own errors by the use of maximum likelihood method:

\[ N = \frac{sn}{v} \quad \text{...(1)} \]

where S is the number of the mistakes introduced, v is the number of identified introduced errors, and n is the number of identified own errors.
The model can be used to estimate the accuracy of the code. Assuming that the testing is repeated until all introduced errors $S=v$ are found, the likelihood of the statement that the code contains $k$ own errors has the following expression:

$$R(k, s) = \begin{cases} 1 & \text{if } n > k \\ \frac{s}{s + k + 1} & \text{if } n \leq k \end{cases} \quad \ldots(2)$$

If the test does not identify all introduced test errors, the following formula must be used:

$$R(k, v) = \begin{cases} 1 & \text{if } n > k \\ \frac{c_s^{v-1}}{c_s^{k+v}} & \text{if } n \leq k \end{cases} \quad \ldots(3)$$

where $v$ is the number of identified test errors.

In practice, this model is used to control the efficiency of the experts who test the code security. A possible chance introduction of errors is considered one of the drawbacks of this model.

### 2.2 Different Modules Model

In this model, the software is divided into two parts. Assuming that Part 1 contains $N_1$ remaining errors and Part 2 $N_2$ remaining errors, the total number of the software remaining errors will be $N = N_1 + N_2$. It is assumed that the identification of the remaining errors is equally probable, and that one error is identified per time unit specified and corrected after identification.

It can be shown that:

$$N_1(i) = N_1 - i + \sum_{j=0}^{i} X_j \quad \ldots(4)$$

$$N_2(i) = N_2 - \sum_{j=0}^{i} X_j \quad \ldots(5)$$

Where $X_j$ is the characteristic function for which:

$$X_j = \begin{cases} 0, & \text{if the } J^{th} \text{ error is found in part 1 of the software} \\ 1, & \text{if the } J^{th} \text{ error is found in part 2 of the software} \end{cases}$$

In this case, the probability of the error identification in the set testing time interval can be evaluated $(t_i, t_{i+1}]$:

$$p_1(i) = \frac{N_1(i)}{N_1(i) + N_2(i)} = \frac{N_1 - i + \sum_{j=0}^{i} X_j}{N_1 - i + N_2} \quad \ldots(6)$$

$$p_2(i) = \frac{N_2(i)}{N_1(i) + N_2(i)} = \frac{N_2 - \sum_{j=0}^{i} X_j}{N_1 - i + N_2} \quad \ldots(7)$$

$N_1$ and $N_2$ can be found with the maximum likelihood method.

### 2.3 Control Model

This model is used by the accredited laboratory in testing software for undeclared opportunities. In the process of the software run-time analysis the set percentage $p$ of the identified functional objects is being controlled. In the statistical analysis, functional objects are arbitrary selected to introduce some test errors. In
testing, test errors $s$ and own errors $n$ are identified. The software error number is evaluated with the use of the maximum likelihood method:

$$
N = n \frac{M_f - m_f + 1}{100} \frac{p}{M_f - s} 
$$

...(8)

### 2.4 Independent Testing Model

The group testing model presupposes testing by two independent testing or expert groups. In testing, the numbers of errors found by either group, $N_1$ and $N_2$, are calculated, as well as the number of coinciding errors, $N_{12}$, found by both groups. Granting that $N$ is the original number of errors, we can define the efficiency of either group as $E_1 = N_1 / N$ and $E_2 = N_2 / N$. If either group is hypothetically equally efficient, we may suppose that if Group 1 finds a certain number of errors out of the set number, Group 1 is able to define the same number of errors for any arbitrary selected subset. This yields:

$$
N_1 / N = N_{12} / N_2 
$$

...(9)

In this case, the intuitive model of the software original errors number evaluation will be:

$$
N = N_1N_2 / N_{12} 
$$

...(10)

This model is practicable when a simultaneous testing is performed by an independent expert group with their own test workbench, which is often the case in travelling tests under restricted time conditions.

### 3. Reliability Growth Models

Models of this class refer to the probabilistic dynamic models of discrete-state systems with continuous or discrete time. Such models are often referred to as time-domain reliability models. Popular models of this class are, for the most part, reducible to the homogeneous or nonhomogeneous Markov, or semi-Markov waiting line models. The homogeneous Markov models presuppose that the total number of errors is an unknown finite constant. The number of errors remaining after testing and debugging is represented by an exponential law. The error rate $\lambda(i)$ depends on the current state of the system $i$ and does not depend on its previous states.

Unlike the homogeneous Markov models, the semi-Markov models presuppose that the error rate $\lambda(t_i)$ depends not only on the number of remaining errors, but also on the time $t_i$ that the system has been in this state.

Presently, among time-dependent models, the non-homogeneous Markov models are becoming more and more popular. In these models, the total number of errors is considered a random value described by the Poisson distribution, while the error rate is not a time-dependent linear function. This is why these models are often referred to as the Poisson models. Depending on the type of the Poisson intensity function, NHPP models are classed into convex, $S$-shaped and infinite. There are modifications of the software reliability models done through the Bayesian method; these are sometimes referred to as a special type of models. It should be noted that for dynamic models calculations, the maximum likelihood method is traditionally employed, and rarely the line regression methods and cross entropy methods. Let us consider some examples of most popular time-dependent reliability growth models of each of the types.

#### 3.1. Exponential Reliability Growth Model

Exponential reliability growth model known as the Jelinski-Moranda or JM model is based on the assumption that while software is being tested, the interval lengths between two consequential error identifications are distributed exponentially, hazard rate being in direct proportion to the number of unidentified errors. All errors are considered equally probable; each identified error is cured immediately, reducing the number of remaining errors by one. Thus, the probability density function of the $i$th error identification time passed from the moment of the $(i - 1)$th error identification is as follows:
\[ p(t_i) = \lambda_i e^{-\lambda_i t_i} \] \hspace{1cm} \text{....(11)}

where \( \lambda_i = \phi (N - (i - 1)) \) is the error rate being in direct proportion to the number of unidentified errors, \( N \) is the original number of errors, \( \phi \) - is the factor of proportionality understood as error detection rate, and \( t_i \) is the time interval between identifications of \((i-1)\)th and \(i\)th errors.

The above formula permits to obtain formulae for error-free operation probability, complete debugging probability within a time given, mean time per one error identification, mean time of complete debugging, etc. For calculation of \( N \) and the maximum likelihood method is used. We provide some examples of popular Markov models; \( N \) stands for the original number of errors.

**JM- Model**
\[
\lambda_i = \phi (N - (i - 1)) \] \hspace{1cm} \text{....(12)}

**Lipov-Model**
\[
\lambda_i = \phi (N - \sum_{j=1}^{i-1} N_j) \] \hspace{1cm} \text{....(13)}

**Xui Model**
\[
\lambda_i = \phi e^{-k(N-(i-1))} - 1 \] \hspace{1cm} \text{....(14)}

**Shanti kumar model**
\[
\lambda_i = \phi (N - (i - 1))^k \] \hspace{1cm} \text{....(15)}

**Buccianico Model**
\[
\lambda_i = 1 - \phi (N-(i-1)) \] \hspace{1cm} \text{....(16)}

### 3.2. The Rayleigh Reliability Growth Model

The Shick-Wolverton or SW model is a development of the JM model based on the assumption that the error rate is in direct proportion to not only the number of unidentified errors but also the debugging time interval length:
\[
\lambda_i = \phi (N - (i - 1))t_i \] \hspace{1cm} \text{....(17)}

where \( N \) - is the original number of errors, \( i \) is the number of errors identified, is the factor of proportionality understood as error detection rate, and \( t_i \) is the time interval between identifications of \((i-1)\)th and \(i\)th errors. Thus, the Rayleigh distribution is deduced for the following probability density function:
\[
\lambda_i = \phi (N - (i - 1))t_i e^{-\phi (N-(i-1))t_i^2/2} \] \hspace{1cm} \text{....(18)}

For calculation of \( N \) and \( \phi \) the maximum likelihood method is used.

### 3.3. S-shaped Reliability Growth Model

At present, the S-shaped model is one of the most popular reliability growth models. The model assumes that the number of errors revealed per time unit is an independent random value of the Poisson distribution, the occurrence rate being in direct proportion to the expected number of remaining errors at the time given. Unlike in JM- and SW-like convex models, this model includes an additional assumption that the number of errors has an S-shaped time dependence. Qualitatively, the S-shaped time dependence of the revealed errors is explained by the fact that at the initial stage of testing the tester spends some time on studying the software.

The number of errors function is as follows:
\[ m(t) = a \left(1 - (1 + gt)e^{-gt}\right) \]  
...(19)

Where \( a \) is the factor of the expected errors, and \( gt \) is the factor of error detection rate. Therefore, the error rate looks as follows:

\[ \lambda(t) = ag^2te^{-gt} \]  
...(20)

The above formulae permit to obtain formulae of probability of identifying (or non-identifying) a certain number of errors over time given.

4. Analysis and Evaluation of Models

We must note that there is no universal model for software evaluation and testing planning. Moreover, apart from the models we have discussed, one can encounter descriptions of other types in the books, such as simulation models, structural models, fuzzy models, interval models, dynamic complexity models, software-hardware simulators, and neural nets; the latter are used for solving particular scientific problems. To choose a suitable model, we may suggest a number of qualitative and quantitative criteria. Among qualitative criteria, we may name the following:

1. The simplicity of use. First of all, this relates to how the model answers the needs for statistical data collection. The used input data must be easily obtainable; they must be representative, and the input and output data must be comprehensible for the experts.

2. Validity, that is, the model must have a reasonable accuracy needed for solving the problems of analysis or synthesis in the software security domain. The positive quality of a model is its ability to make use of apriori information and complexing other models’ data to reduce the input selection.

3. Usability for solving different problems. Some models permit to obtain an assessment of a wide range of factors that experts may need at different software lifecycle stages, for example, reliability indicators, expected rates of errors of different types, predictable time and cost expenditures, experts’ proficiency, quality of tests, accuracy indicators, software overlapping indicators, etc.

4. Simplicity of implementation including the possibility to automate the assessment process basing on the existing mathematical packages and libraries, to retrain the model following upgrades, and to allow for incomplete or incorrect input statistics or for any other restriction.

5. Conclusions

One should bear in mind that because of rapid development, complexity, and diversity of modern software kits, the above models must not be expected ever to provide high accuracy, and very often they only provide intuitive data for taking a decision in preparation of software testing on the entire array of input data. Notwithstanding this, the results of these models applications are very convenient for use in both the justification of testing labour costs and reporting records, which may be helpful for the customer to view the obtained results as reliable. Our research has revealed a great number of mathematical models that can be used to assess the technical software security at different stages of its lifecycle, which is very important for information security cost budgeting. The suggested classification of models will be practical when making the right choice or complex models on the basis of available statistics.

References:

