DEVELOPING SOFTWARE ANALYZERS TOOL USING SOFTWARE RELIABILITY GROWTH MODEL FOR IMPROVING SOFTWARE QUALITY

Mr. Sandeep P. Chavan, Dr. S. H. Patil, Mr. Amol K. Kadam

1 Department of Computer Engg Bharati Vidyapeeth Deemed University College of Engineering Pune.
2 Department of Computer Engg Bharati Vidyapeeth Deemed University College of Engineering Pune.
3 Department of Computer Engg Bharati Vidyapeeth Deemed University College of Engineering Pune.

ABSTRACT

Software reliability is defined as the probability of free failure operations for a specified period of time in a specified conditions. Software reliability growth models (SRGM) have been developed to estimate software reliability measures such as number of remaining faults, software failure rate & software reliability. Software testing is a measure face which takes place during software development process. Testing is process of finding errors. It involves any activity aimed at evaluating an attribute or capability of the program for system & determining that it meets its required result. Software testing is a critical element software quality & ensures & represents the ultimate review of specification, design & code generation. But testing of the software for long time may not ensure a bug free software & high reliability. Optimum amount of code also needs to be covered to make sure that software may not be removed perfectly; this is mainly due to complexity of software or nature of testing team. This phenomenon is known as imperfect debugging. When the faults are not removed perfectly & leads to further generation of fault this process is known as error generation.

In the event of flawed blame debugging the flaw substance of the programming is not updated, however only in light of fragmented comprehension of the programming, the located issue is not uprooted altogether. At the same time in the event of blunder era the issue substance builds as the testing advances and evacuation brings about presentation of new blames while uprooting old ones. n. The model has been validated, assessed and contrasted and other existing discrete NHPP demonstrates by applying it on real washout / issue
evacuation information sets refered to from true programming advancement ventures. The effects show that the proposed model furnishes enhanced goodness of fit and prescient legitimacy for programming flop / shortcoming evacuation information.

**Keywords:** Software reliability growth models (SRGM), ENHPP (Enhanced Non-Homogeneous Poisson Process), SDLC(Software Development Life Cycle), SRS(Software Requirement Specification).

1. **INTRODUCTION**

SRGM is mathematical model. It shows how software reliability improves as faults are detected and repaired. SRGM can be used to predict when a particular level of reliability is likely to be attained. Thus, SRGM is used to determine when to stop testing time. All these activities require complex software systems. It is important that these systems are thoroughly tested before implementation. There is a huge cost attached with fixing failures, safety concerns, and legal liabilities therefore organizations need to produce software that is reliable. There are several methodologies to develop software but questions that need to be addressed are how many times will the software fail and when, how to estimate testing coverage, when to stop testing, and when to release the software.[1] Also, for a software product we need to predict/estimate the maintenance effort; for example, how long must the warranty period must be, once the software is released, how many defects can be expected at what severity levels, how many engineers are required to support the product, for how long, and so forth.[5]

The prospective disastrous failure of software and the consequential damage is a scenario that is of great concern in terms of scientific disaster preparedness. Proper vigilance for such an event to enable mitigation of adverse impacts to the greatest degree possible requires modeling of software reliability to capture the risk of software failure. Programming inadequacies may be because of mistakes, ambiguities, oversights or error of the detail that the programming is expected to fulfill, lack of regard or awkwardness in composing code, deficient testing, off base or surprising utilization of the programming or other unforeseen issues. These errors can hamper the performance of the software and can adversely affect its reliability. Therefore software reliability is of great concern for software developers.[2]

Reliability techniques can be divided into two categories namely Trending and Predictive .Trending reliability tracks the failure data produced by the software system to develop a reliability operational profile of the system over a specified time Whereas Predictive reliability assigns probabilities to the operational profile of a software system. In practice, reliability trending is more appropriate for software, whereas predictive reliability is more suitable for hardware. Trending reliability can be further classified into four categories:[4] Error Seeding, Failure Rate, Curve Fitting, and Reliability Growth. Error Seeding: Estimates the number of errors in a program by using multistage sampling. Errors are divided into indigenous and induced (seeded) errors. The unknown number of indigenous errors is estimated from the number of induced errors and the ratio of errors obtained from debugging data. Failure Rate: Is used to study the program failure rate per fault at the failure intervals. As the number of remaining faults change, the failure rate of the program changes accordingly.[2]

Curve Fitting: Uses statistical regression analysis to study the relationship between software complexity and the number of faults in a program, as well as the number of changes,
or failure rate. Reliability Growth: Measures and predicts the improvement of reliability programs through the testing process. Reliability growth also represents the reliability or failure rate of a system as a function of time or the number of test cases.

As reliability is of great concern for software products this field is catching attention of various researchers and practitioners. This has lead to a new concept Software Reliability Growth Modeling. A SRGM is characterized as a device that might be utilized to assess the programming quantitatively, advance test status, plan status, and screen the progressions in dependability exhibition. Software reliability assessment and prediction is important to evaluate the performance of software system. The reliability of the software is quantified in terms of the estimated number of faults remaining in the software system. During the testing phase, the emphasis is on reducing the remaining fault content hence increasing the software reliability.[3] The SRGMs developed in literature are either dependent on testing time, testing effort or testing coverage. Testing time is the calendar time or the CPU time whereas testing effort takes into account the manpower time and the CPU time. We discuss some novel software reliability growth models dependent on time. Testing Coverage plays a very important role in predicting the software reliability. Testing Coverage is actually a structural testing technique in which the software performance is judged with respect to specification of the source code and the extent or the degree to which software is executed by the test cases. TC can help software developers to evaluate the quality of the tested software and determine how much additional effort is needed to improve the reliability of the software besides providing customers with a quantitative confidence criterion while planning to use a software product. Hence, safety critical system has a high coverage objective. The basic testing coverage measures are [1].

1. Statement Coverage: It is defined as the proportion of lines executed in the program. If we assume that the faults are uniformly distributed throughout the code, then percentage of executable statements covered shows the percentage of faults discovered.
2. Decision / Condition Coverage: This measure indicates whether Boolean expressions tested in control structures evaluated to both true and false.
3. Path Coverage: This measure shows the percentage of all possible paths existing in the code exercised by the test cases.
4. Function Coverage: This measure indicates the proportion of functions/ procedures influenced by the testing Software reliability growth models based on testing coverage have been developed by Inoue et al and Pham et al in literature based on various testing coverage functions. These testing coverage functions are based on time and are utilized to measure the coverage of the software.

2. LITERATURE SURVEY

Various investigative programming dependability shows have been proposed for evaluating the unwavering quality development of a programming item. In this paper we present an Enhanced non-homogeneous Poisson process (ENHPP) model and show that awhile ago reported Non-homogeneous Poisson process (NHPP) based models, with limited mean esteem capacities, are extraordinary instances of the (ENHPP) model. The (ENHPP) model contrasts from past models in that it consolidates unequivocally the time differing test scope capacity in its analytical definition, and accommodates damaged blame discovery in the testing stage and test scope throughout the testing and operational stages. The (ENHPP) model is validated utilizing a few accessible inadequacy information sets.
SDLC Phases

I] Feasibility analysis
   Includes analysis of project requirements in terms of input data and desired output, processing required to transform input into output, cost-benefit analysis, and schedule of the project. The feasibility analysis also includes the technical feasibility of a project in terms of available software tools, hardware, and skilled software professionals. At the end of this phase, a feasibility report for the entire project is created.

II] Requirement analysis and specification
   Includes gathering, analyzing, validating, and specifying requirements. At the end of this phase, the Software Requirement Specification (SRS) document is prepared. SRS is a formal document that acts as a written agreement between the development team and the customer. SRS acts as input to the design phase and includes functional, performance, software, hardware, and network requirements of the project.

III] Design
   Includes translation of the requirements specified in the SRS into a logical structure that can be implemented in a programming language. The output of the design phase is a design document that acts as an input for all the subsequent SDLC phases.

IV] Coding
   Includes implementation of the design specified in the design document into executable programming language code. The output of the coding phase is the source code for the software that acts as input to the testing and maintenance phase.

V] Testing
   Includes detection of errors in the software. The testing process starts with a test plan that recognizes test-related activities, such as test case generation, testing criteria, and resource allocation for testing. The code is tested and mapped against the design document created in the design phase. The output of the testing phase is a test report containing errors that occurred while testing the application.

VI] Maintenance
   Includes implementation of changes that software might undergo over a period of time, or implementation of new requirements after the software is deployed at the customer location. The maintenance phase also includes handling the residual errors that may exist in the software even after the testing phase.

   Testing of the software for a long time may not ensure a bug free software and high reliability. Optimum amount of code also needs to be covered to make sure that the software is of good quality. Testing time alone may not give the correct picture of the number of faults removed in the software.

   Therefore proposed Analysis and Design of Software reliability growth model in concurrence with SDLC. Real needs for improving software reliability to improve system dependability and reduce maintenance cost. So to make software reliable we enhanced testing phase of SDLC to testing time, testing coverage and testing quality.[5]
3. STATEMENT OF AIM AND OBJECTIVE:

Aim of this model is to achieve the reliability of the software and increase the role of software in real-life systems. Objectives are to manage and improve:

a) The reliability of the software
b) Check the efficiency of development activities
c) Evaluate the software reliability at the end of validation activities and in operation
d) Estimate the maintenance effort to “correct” faults activated during development and residual faults in operation.

4. PROPOSED MODEL

The new SDLC model that is developed to minimize the test effort of software projects. This model is an enhanced version of Non-Homogeneous Poisson Process (NHPP) model. The main feature of our model is to use the stubs and drivers for reusability of code to minimize the test effort.

Main Goals of the proposed model
- Enhance the reusability of code.
- Minimize the test effort estimation.

A software testing model summarizes how you should think about test development. It tells you how to plan the testing effort, what purpose tests serve, when they’re created, and what sources of information you use to create them.


The ENHPP model provides a unifying framework for finite failure software reliability growth models. According to this model, the expected number of faults detected by time $t$, called the mean value function, $n(t)$, is of the form:

$$n(t) = b * c(t)$$

Where $b$ is the needed number of shortcomings in the programming (before testing/debugging starts), and $c(t)$ is the scope capacity. The ENHPP model utilized by SREPT gives by default to reflect four sorts of inadequacy event rates for every fault. Interflop times information acquired from the testing stage could be utilized to parameterize the ENHPP (Enhanced Non-Homogeneous Poisson Process) model to acquire gauges of the inadequacy force, number of flaws remaining, dependability after discharge, and scope for the programming.

When complexity metrics are available, the total number of faults in the software can be estimated using the fault density approach or the regression tree model. If the number of lines of code in the software is $NL$, the expected number of faults can be estimated as $[5]$:

$$F = NL * FD$$

The relapse tree model is an objective arranged statistical method, which endeavors to foresee the amount of deficiencies in a programming module dependent upon the static unpredictability measurements. The structure of the ENHPP model might additionally be utilized to join the appraisal of the aggregate number of issues acquired after the testing stage dependent upon unpredictability measurements (parameter $a$), with the scope informative content got.
throughout the testing stage \((c\ (t))\). The ENHPP model can also use inter failure times from the testing phase to obtain release times (optimal time to stop testing) for the software on the basis of a specified release criteria. Release criteria could be of the following types -

Number of remaining faults - In this case, the release time is when a fraction of all detectable faults has been removed.

Failure intensity requirements - The criterion based on failure intensity suggests that the software should be released when the failure intensity measured at the end of the development test phase reaches a specified value \(f\).

Reliability requirements - This criteria could be used to specify that the required conditional reliability in the operational phase is, say \(R_r\) at time \(t_0\) after product release.

Cost requirements - From a knowledge of the expected cost of removing a fault during testing, the expected cost of removing a fault during operation, and the expected cost of software testing per unit time, the total cost can be estimated. The release time is obtained by determining the time that minimizes this total cost.

Availability requirements - The release time can be estimated based on an operational availability requirement [2].

5. CONCLUSION

We have proposed SRGM in testing phase of SDLC to ensure the most reliable software and quality. We have to develop ENHPP (Enhanced Non-Homogeneous Poisson Process) model to find out errors and faults in the current and existing system. It will improve the quality and reliability of the software and moreover, it will eliminate the errors and faults in the current and existing system. Therefore the SRGM testing will become more authentic.

6. REFERENCES