Software Engineering with debugging phenomena by optimal software release policies

M.V.S.Prasad

Research Scholar, AcharyaNagarjunaUniversity,Ap,India

Abstract

For testing a project we have many software testing techniques, tools but there is no exact suitable test cases for testing a system. Testing tells about the errors and software methodologies, tools, and techniques have emerged over the last few years promising to enhance software quality. There are now numerous testing techniques available for generating test cases. In this paper I am giving new policy for Software Engineering with Debugging phenomena by optimal software release.
Abstract should contain at least 250 words.

Index Terms: *Software Testing, Level of Testing, Testing Technique, Testing Process..*

-- *** --

1. INTRODUCTION

The testing is an important means of assessing the software to identity the quality. Since testing typically consumes 40-50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering Modern software systems must be extremely reliable and correct. Automatic methods for ensuring software correctness range from static techniques, such as (software) model checking or static analysis, to dynamic techniques, such as testing. All these techniques have strengths and weaknesses: model checking (with abstraction) is automatic, exhaustive, but may suffer from scalability issues. Static analysis, on the other hand, scales to very large programs but may give too many spurious warnings, while testing alone may miss important errors, since it is inherently incomplete

IEEE STANDARDS:

Institute of Electrical and Electronics Engineers designed an entire set of standards for software and to be followed by the testers.i.e.Standard Glossary of Software Engineering Terminology, Standard for Software Quality Assurance Plan, Standard for Software Configuration Management Plan

II.OBJECTIVES

- A good test case is one that has a probability of finding an as yet undiscovered error.
- A good test is not redundant.
- A successful test is one that uncovers a yet undiscovered error.
- A good test should be “best of breed”.
- A good test should neither be too simple nor too complex.
- To check if the system does what it is expected to do.
- To check if the system is “Fit for purpose”.
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- Executing a program with the intent of finding an error

III PHASES FOR TESTING LIFE CYCLES

- 1. Requirements study
- Testing Cycle starts with the study of client‟s requirements.
- Understanding of the requirements is very essential for testing the product.
- 2. Test Case Design and Development
- Component Identification
- Test Specification Design
- Test Specification Review
- 3. Test Execution
- Code Review
- Test execution and evaluation
- Performance and simulation
- 4. Test Closure
- Test summary report
- Project De-brief
- Project Documentation
- 5. Test Process Analysis

- Analysis done on the reports and improving the application"s performance by implementing new
- technology and additional features
-

IV LEVEL OF TESTING

1. Unit testing
   - The most „micro" scale of testing.
   - Tests done on particular functions or code modules.
   - Requires knowledge of the internal program design and code.
   - Done by Programmers (not by testers).

2. Incremental Integration Testing
   - Continuous testing of an application as and when a new functionality is added.
   - Application"s functionality aspects are required to be independent enough to work separately before completion of development.
   - Done by programmers or testers. Integration Testing
   - Involves building a system from its components and testing it for problems that arise from component interactions.
   - Top-down integration:
   - —Develop the skeleton of the system and populate it with components.
   - Bottom-up integration
     - Integrate infrastructure components then add functional components.
   - To simplify error localization, systems should be incrementally integrated.

Functional Testing
- Black box type testing geared to functional requirements of an application.
- Done by testers

Software reliability growth models are mathematical models which are helping the software industry toestimate the software errors and reliability during the testing phase. Usually the implemented software is thoroughly tested to track the additional errors which cause the software to fail in future. Many software reliability growth models were proposed in literature. Software reliability growth model with constant failure detection is studied by (Goel et al, 1979). S shaped software reliability growth models which describestesters with learning process (Yamada et, al, 1984). These type of software reliability models in whichfailure is not constant, in the initial stage the failure detection is slow as the time progress the failure rate is increased.

In all above models the error correction is perfect.a software reliability growth model based on the non-homogeneous Poisson process, under the assumption that the detection of these errors can also cause the detection of some of the remaining errors without these errors causing any failure. Using the expected number of errors thus detected, we obtain the mean value function describing the failure phenomenon. Parameters of the model are estimated, and the applicability of the model is illustrated.two general frameworks for deriving several software reliability growth models based on a nonhomogeneous Poisson process in the presence of imperfect debugging and error generation. The proposed models are initially formulated for the case when there is no differentiation between failure observation and fault removal testing processes, and then extended for the case when there is a clear differentiation between failure observation and fault removal testing processes.Some extended replacement policies based on the number of failures, incorporating the concept of repair cost limit are discussed. Three models are considered as follows:(a) a unit is replaced at the nth failure, or when the estimated minimal repair cost exceeds a particular limit c;(b) a unit has two types of failures and is replaced at the nth type 1 failure, or type 2 failure, or when the estimated repair cost of type 1 failures exceeds a predetermined limit c—type 1 failures are minimal; failures

- Software Reliability Growth Models (SRGMs) play a vital role in planning and controlling the testing phase of a Software Development Life Cycle(SDLC). One of the most important decisions related to the efficient management of testing phase is to determine when to stop testing and release the software in use since no software can be tested indefinitely in order to make it bug free. The optimization problem of determining the optimal time of software release can be formulated based on goals set by the management in terms of cost, reliability and failure intensity etc subject to the system constraints. In this paper we have proposed and validated two discrete SRGMs termed as homogenous and non-homogenous fault detection rate models incorporating the effect of imperfect fault debugging and error generation. The homogenous fault detection rate model consider the case where the software system contains only one type of fault, whereas the non-homogenous fault detection rate model is formulated to cater for the case when the software system contain two types of faults viz easy and difficult to detect. Keeping in mind the current demand of multiple criteria approach Bicriterion Release Time optimization problem is formulated minimizing the expected software cost and maximizing the reliability subject to the budget constraint and reliability requirement and optimal release time is determined. Results are illustrated using a numerical example. Finally sensitivity analysis is performed to determine the effect of variations in reliability, cost, level of perfect debugging and fault generation on release time of software.which have relevance to computing systems and those wherenumerical computation was a problem. It is also well known that nearly 70% of the cost goes into software development and hence software reliability assumes special importance

Several Software Reliability Growth Models (SRGMs) have been developed in the literature assuming the debugging process to be perfect and thus implying that there is one-to-one correspondence between the number of failures observed and errors removed. However, in reality it is possible that the error which is supposed to have been removed may cause a failure again.

CONCLUSION

We presents a new model for presenting the ideal release time for a computer software in trying phase, enchanting account of the restoring time lag. In the prior works, most of software release models were well-thought-out, but it was assumed that an error detected can be removed instantaneously. In other words, none discussed quantitatively the effect of the software maintenance action in the optimal software release time. Main purpose of this work is to relate the optimal software release policy with the arrival-service process on the software operation phase by users. We use the Non-Homogeneous Poisson Process (NHPP) type of software reliability growth models as the software error detection phenomena and obtain the optimal software release policies minimizing the expected total software costs. As a result, the usage circumstance of a software in operation phase gives a monotone effect to the software release planning