A Study in Release Time Problems

Somkant Khare*

Research Scholar, Department of Mathematics, SVN University Sagar (MP)

----------X----------

1. INTRODUCTION

Very early in the development of computers, people referred to the actual physical components – the tubes and relays, the restores and wires, and chassis – as computer hardware. The word software was then coined to describe the non-hardware components of the computer, in particular the programs that were needed to make the computers perform their intended tasks. The word caught on rapidly, and was in quite general use by 1960.

One speaks of software shops (i.e organizations that produce software), software maintenance, and more recently, software engineering. Although the word software can be used in connection with all kinds of programs, it is usually used to denote programs whose use in not limited to one particular job or application. Thus, one speaks of system software, of software systems, of mathematical software, of software for business applications, etc.

Growth in software engineering technology has led to production of software for highly complex situations occurring in industry, scientific research, defense and day to day life.

The computer revolution is fueled by an ever more rapid technological advancement.

Thoday, computer hardware and software permeates aur modern society. Computer are embedded in wristwatches, telephone, home appliances, buildings, automobiles, and aircraft. Science and technology demand high-performance hardware and high-quality software for making improvements and breakthroughs. We can look at virtually any industry-automotive, avionics, oil, telecommunications, banking, semi-conductors, pharmacals – all these industries are highly dependencies on computers increase, the possibility of cries from computer failures also increase. The impact of these failures ranges from inconvenience (eg., malfuncation of and home appliances) to economic damage (eg., interruption of banking systems) to loss of life (eg., failure of flight systems or medical software).

Needless tomsay, the reliability of computer systems has a major concern for our society.

Though high reliability of hardware part of these systems can be guaranteed, the same cannot be said for software. Therefore a lot of importance is attached to the testing phase of the software development process, where around half the development resources are used [Musa et al., 1988].

Essentially testing is a process of executing a program with the explicit intention of finding faults and it is this phase, which is amendable to mathematical modeling.

It is always desirable to remove a substantial number of faults from the software. In face the reliability of the software is directly proportional to the number of faults removed. Hence the problem of maximization of software reliability is identical to that of maximization of fault removal. At the same time testing resource are not unlimited, and they need to be judiciously used.

In focusing on error prevention for reliability, we need to identify and measure the quality attributes applicable at different life cycle phases. As discussed previously, we need to specifically focus on requirements, design, implementation, and phases.

Testing phase in Software Development Life Cycle.

Software development process is often called Software Life Cycle, because it describes the life of a software product from its conception to its implementation. Every software development process model includes system requirements as input and a delivered product as output. Many life cycle models have been proposed, based on the tasks involved in developing and maintaining software, but they all consist of the following stages and faults can be introduced during any of these stages.

2. RELEASE POLICIES UNDER PERFECT DEBUGGING

With the intrusion of computer in every walk of lives-improper functioning \failure of software can cause serious problems. As software systems have become more and more complex, the important stage of software development life cycle as it provides the measure of software reliability and assists to judge the performance, safety, fault –tolerance or security of the software. A software development life cycle consists of four phases:

Specifications, development, Testing and Implementation.

Nearly half of the resources of SDLC are used up in the testing phase. Before newly developed software is released to the user, it extensively tested for errors that may have been introduced during development.

During the testing phase one of the major concerns for the management is to determine when to stop testing and release the software to the user. Although deteted errors are removed immediately, new errors may be introduced during debugging. Software that contains errors and is released to the market incurs high failure costs. Debugging and

Release Policies Based On Different Criteria .

Notations

a : initial error content.

b : proportionality constant (failure rate per error).

m(t) : expected number of software failures by time t.

C1 : cost incurred on a perfect debugging effort before release of the software system.

C2 :cost incurred on a perfect debugging effort after release of the software system (C2 >C1).

C3 : testing cost per unit time.

x : mission time.

T' : optimal release time.

T1 : software life cycle length.

For an exponential SRGM in continuous time, the mean value function M(t) ((number of failures/faults removed) is defined as: Where a is the total expected error content in the software and b is the error detection rate per remaining error. The failure intensity function is

It may be observed that □(t) is decreasing function in t with □(0) =ab and □(∞)=0.

Thus the total software development cost incurred by the manufacturer during the software life cycle where m(T) is the total number of faults detected upto time t which are removed instantly is given by Where C1 and C2 are the cost of fixing a fault before (after) releasing the software, C3 is the testing cost per unit tine T1 is software lifecycle (>T). C2 is assumed to be greater than C1

The expected software reliability R(x|T) is defined as the probability that a software failure does not occur in tha interval (T, T+x], given that last failure occurrence time is T ≥ 0 (x ≥ 0) is defined as.

Release Policies under Cost Criteria [Okumoto et al., 1980].

A major concern in software development is the cost. It is well known that the development of a software system is time –consuming and costly. Thus the main aim of the management is always to minimize the total software development cost keeping in maind the desired reliability of the software which has to achieved. Hence software cannot be tested for an indefinite period of time as it increases tis testing cost indefinitely or it cannot be released prematurely as it increases the cost of fixing the faults in operational phase. Thus the total software development cost plays vital role in determining the optimal release time of the software. Most commonly cost models seen in literature for determaination of release time for perfect debugging NHPP models includes cost of testing, cost of removing faults during testing phase and cost of failure and removal of faults during operational phase.

3. RELEASE POLICIES UNDER IMPERFECT DEBUGGING

In conventional software growth models, it is assumed that an error or a fault is completely removed after it is detected. Thais implies that no new error or fault is introduction when an a fult is removed. This assumption significantly contributes to the simplicity of these models. In a practical project, however, it is hard to assume that no errors are introduced when an erroe is detected and removed. Almost all professional programmers have experienced cases in which they fixed one error to create another.

Sometimes new reeors are introduced seve3ral times in fixing a single error. For this reason,practical prople donot believe the results of software reliability growth analysis. They sometimes say that it is a moving target.

In rela life situations, most of the debugging processes or the fault removal efficiency is not

perfect. The fault removal team may not be able to remove the fault perfectly on the detection of a failure and the original fault may remain or replaced by another by another fault. When a failure occurs, the cause of the failure is identified and removed. To ensure occurs again, the code is checked again. Two possibilities occue.

The fault, which was thought to be perfectly fixed, has been imperfectly repaired and caused same type of failure again when checked on the fact that the faut was perfectily removed but some other fault was generated while removing the cause of the failure. This called error generation, which can be be known only during the removal phase. Imperfect fault debugging causes more faqilures as compared to removals by time infinity but the fault content temains the same. However, when a fault is genered, the munber of failures increases because the fault content has increased. Some models have been developed in literature to incorporate the. effect imperfect debugging in modeling software reliability [Kapur et al.,1996;Ohbe et al., 1989; pham etal.,1999; Zang et al.,2003].

This chapter is divided in six sections. In section-3.1,we discuss two software reliability growth models (SRGMs) under imperfect debugging based on non-homogeous poisson process (NHPP) and combining multiple failure types with impact debugging that can be used to determing the optimal 3.3 presents a software reliability growth models which incorporates the possibility of introducing secondary faults, generated through imperfect debugging of primary failures. The mean total number of failures, comprising the primary and secondary failures, is obtained. We also discussed a cost model and consider some optimal release policies based on random lift cycle as well as a penalty cost (due to delay for a scheduled delivery time).

Related optimal release policies that minimize the expected software System costs (subject to various constraints) are also discussed. Further, in section 3.5, we investigate the effect of imperfect debugging on software development cost, which, in turn, might affect the optimal software release time or operational budget. Finally, in section 3.6 we summarize the conclusion.

Exponential and Modified Exponential Reliability Growth Models [Kapur et al.,1990].

Here we discuss two software reliability growth models (SRGMs) under imperfect debugging based on-non-homogeneous poison (NHPP). Related optimal release policies are also discussed. Total cost incurred on the software until it is supported also include the cost incurred on those failures which could not be removed.

Release policies discussed have tended to minimize such a wasteful expenditure.

4. RELEASE POLICIES BASED ON COST AND RELIABILITY CRITERIA CONSIDERING TESTING EFFORT AND EFFICIENCY

During the past 30 years, a number of Software Reliability Growth Model (SRGMs) were proposed [Xie, 1991; Lyu, 1996; RAC, 1997; Pham, 2000; Grottke, 2001]. SRGMs are applicable to the late stages of testing. They can provide very useful information about how to improve reliability. Some important metrics, such as the number of initial faults, failure intensity, reliability within a specific time period, number of remaining faults, mean time between failures (MTBF), and mean time to failure (MTTD), can be easily determined through SRGMs. Issues such as imperfect debugging and the learning phenomenon of developers have been considered in these models.

Most SRGMs assume that faults detected during tests will eventually be removed.

Consideration of fault removal efficiency in the existing models is limited. Fault removal efficiency is a useful metric in software development practice and it helps developers to evaluate the debugging effectiveness and estimate the additional workload. In practice, fault removal efficiency is usually imperfect. Although some software reliability studies addressed the imperfect debugging phenomenon, most of them only considered possibility of adding new fault while removing the existing ones. However, imperfect debugging also means that detected faults are removed withimperfect removal efficiency other than 100%.

It is not unusual for the software development team to find that a software fault has been reported multiple times befor it is finally removed. Some faults can only be encountered in the customer field trails. Therefore fault removal efficiency is an important factor for software quality, reliability estimation and software project management.

Some reliability models are very successful in predicting the faults during phases of the development. However, choosing a good model that can be used to explain the current and past failure behavior most adequately is very important. From our studies, we find that many authors considered a Non-homogeneous poisson process (NHPP) as a stochastic process to describe the fault process. Recently, [Huang et al., 1998, 1999a,b; kuo et al.,2001; Huang and kuo, 2002; Huang et al., 2003 Huang, in press] proposed a SRGM that

incorporates the concept of logistic testing-effort function (TEF) into an NHPP model to get a better description on the software fault phenomenon. The logistic TEF has the advantage of relating the work profile more directly to the natural structure of software development. It can be used to pertinently describe the resource consumption during the software development process and get a conspicuous improvement in modeling the distribution of testing – effort ecpenditures.

The proposed model has a fairly accurate prediction capability.

In addition to modeling the software fault detection process, we also address the problem faced by most software managers, namely, how to decide when to stop testing and release software. This is a problem of decision-making under uncertainly and involves a tradeoff between realistic and cost. Here we propose a new software cost model that can be used to formulate realistic total software cost projects discuss the optimal release policy based on cost and reliability considering testing-effort and efficiency. The cost modal includes the testing cost, the debugging cost during testing phase, and the extra cost due to introduce new test techniques, etc.

This chapter is organized as follows: In section 4.1, we discuss a methodology to integrate fault removal efficiency into SRGMs. Section 4.1.1 presents the formulation of NHPP model addressing fault removal efficiency and fault introduction rate. Moreover, the exlicit solution of the mean value function (MVF) for the proposed model is also discussed in this section. In section 4.2, we frist review a SRGM with generalized logistic testing-effort function (TEF) in sevtion 4.2.1. In section 4.2.2, we introduce the concept of testing efficiency improvement obtained by new test techniques during testing. The optimal software release time problem based on minimizing cost subject to achieving a given level of reliability considering the extra cost of introducing new tehniquse during testing is discussed in section 4.2.3. Section 4.3 concludes the chapter.

Bicriterion Release policy for Continuous Software Reliability Growth Model under Imperfect Debugging [Kapur et al.,1994].

Notation

a : initial error content.

b : proportionality constant (fault removal

rete per remaining fault).

m(t): mean value function in the NHPP model,m(0)=0.

C1(C2) : cost of fixing an error during testing (operation) (C2 > C1)

C3 : testing cost per allocated for the software.

CB : total budget allocated for the software.

Assumptions

1. Software system is subject to failures during execution caused by faults remaining in the software.

2. Failure rate of the software is equally affected by faults remaining in the software.

3. The expected number of failures per test occasion is proportional to the current cantent of the current fault content of the software system.

4. On a failure, instantaneous repair effort starts and the following may occur:

a) fault content is reduced by one witty probability p ;

b) fault content is unchanged with probability 1- p .

It is assumed that p > 1- p.

5. Software life cycle length is more than the optimal number of test occasions the release of the release of the software.

REFERENCES

Akaike H. (1974). A new look at statistical model identification, IEEE Trans. Automat. Cont., vol.Ac-19,pp. 716-723.

Bittant S., Bolzern P., pedrotti E., pozzi M.and Scatto;ini R.(1998). "A falxible modeling approach for software reliability growth, in software reliability modeling and identification, Ed.S Bittani, Springer-verlag, Berlin.

Caspi P.a. and Kouka E.F (1984). Stopping rules rules foe a debugging process Based on different software reliability models proc. Int. Conf on Fault-Tolerant Computing, pp. 114-119.

Doob J.L.(1953). Stochastic process, john wiley.

Ehrlich W., Prasanna B., Stampfel J. and Wu J. (1993), " Determining the Cost of a Stop-Testing Decision,'' IEEE Software, pp. 33-42.

Forman E.H and Singpurwalla N.D (1979). "An empirical stopping rule for debugging and

testing computer software" ,Jour. Amer. Stut. Asso. 72., pp. 750-757.

Gillies A.C. (1992). "Software Quality, Theory and management", Chapman Hall computing Series, London, UK.

Goal A.L. and Okumoto, K. (1979). "Time dependent error-detection rate modal for software reliability and performance measures". IEEE Trans. Reliability R-28(3), pp. 206-211.

Goel A.L. (1985). "Software Reliability Models: Assumptions, limitations and applicability", IEEE Trans. On software engineering, SE-11, pp. 1411-1423.

Hossnin S.A. and Dahiya R.C. (1993). "Estimating the Parameters of a Non- Homogeneous Poisson process Model for Software Reliability", IEEE Trans. Reliability, vol. 42 pp. 604-612.

Corresponding Author

Somkant Khare*

Research Scholar, Department of Mathematics, SVN University Sagar (MP)