A Study in Release Time Application

Somkant Khare*

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

1. INTRODUCTION

Software Quality Characteristics

Quality is built into the software; it does not just happen to be there because the developers did a good job. Quality assurances practices are concurrent with all process activities.

Quality is defined as “the degree of excellence of excellence of something”. This implies a subjective factor; any project can be found lacking if measured against a vague notion of what high quality is.

Software quality in the context of software engineering measures how well software is designed (quality of designing), and how well the software conforms to that design (quality of conformance) [Musa et al., 1990]. Quality of designing measures how valid the design and requirements are in creating a worthwhile product [Triantafillou et al., 1995] whereas quality of conformance is concerned with implementation. Software quality is measured via a set of attributes that are characteristics of high-quality software.

Then we build into the requirements the attribute that desired in the final product. It is not always possible to measure each attribute directly, but some form of relative measurement must be made. Common among the characteristics are: completeness, correctness, dependability, efficiency, reliability, maintainability, portability, robustness (the ability to minimize the impact of external factors, such as user errors or adverse environmental conditions), testability, and usability, but the is not limited to these.

2. RELIABILITY AS A QUALITY ATTRIBUTE

Software reliability is an important facet of software quality. It is defined as “the probability of failure-free operation of a computer program in a specified environment for a specified to the successful use of computers. It is necessary that the reliability of software should be measured and evaluated, as it is in hardware. One of reliability’s distinguishing characteristics is that it is objective, measurable, and can be estimated, whereas much of software quality is subjective criteria. This distinction is especially important in the discipline of Software Quality Assurance. These measured criteria are typically called software metrics.

There are many different models for software quality, but in almost all models, reliability is one of the criteria or attributes or characteristic that is incorporated. The IEEE defines reliability as the ability of a system of component to perform its required functions under conditions for a specified period of time. To most project and software development managers, reliability is related to correctness, that is, they look to testing and the number of “bugs” found and fixed. While finding and fixing bugs discovered in testing is necessary to assure reliability, a better way is to develop a robust, high quality product through all of the stages of the software lifecycle. That is, the reliability of the delivered code is related to the quality of all the processes and products of software development; the requirements documentation the code, test plan, and testing.

3. SOFTWARE RELIABILITY GROWTH MODEL

Since computers are being used increasingly to monitor and control both safety-critical and civilian systems, there is a great demand for high quality software products. Reliability is a primary concern for both software developers and software users.

Software reliability engineering (SRE) has generated quite a bit of interest and research in the software reliability modeling.

A Software Reliability Growth Model (STGM) is a relationship between the number of faults removed from a software and the execution time/CPU time/calendar time. Several attempts have been made to represent the actual testing environment through SRGMs [Goel, 1985; Kapur and Garg, 1990; Kapur et al., 1999. Yamada et al., 1986]. These models have been used to predict the fault content, reliability and release time of a software. SRGMs have also been used to manage the testing phase. This chapter will present some of the important models that have appeared in the recent literature.
But before considering the models we’ll first provide a historical perspective of the development of this field and some needed theoretical results from reliability theory, which we’ll use in model development. We’ll then go into the models.

**With Cost and Reliability Criteria under Penalty Cost [Yamada et al., 1984]**

Here we introduce the concept of delivery time (time at which software is supposed to be released for use). If the manufacturer fails to release the software at the scheduled delivery time, he has to pay a price termed as penalty cost. Let $T_s$ (the scheduled delivery time) be a random variable (pdf) $g(t)$. If $p_c(t)$ is the penalty cost in $(0,t]$ due to delay in software release, then the expected penalty cost in $(T_s,T]$ is expressed as.

**A Software Cost Model with Warranty and Risks Costs [Pham et al., 1999]**

Here we discuss a cost model which considers the testing cost, cost of removing errors detected during testing phase, cost of removing errors detected during the warranty period, and risk cost due to software failure. A software reliability model based on NHPP is used.

The optimal release policies to minimize the expected total software cost are discussed.

This model can be used to estimate realistic total software cost for several applications, such as telecommunication, customer service, etc., and to determine the optimal testing - release policies of the software system.

**Notations**

- $R(x|T)$: reliability function of software by time $T$ for a mission time $x$.
- $T$: software testing time.
- $T'$: optimal software release time.
- $C_0$: set-up cost for software testing.
- $C_1$: software test cost per unit time.
- $C_2$: cost of removing an error per unit time during testing period.
- $C_3$: cost of removing an error per unit time during warranty period.
- $C_4$: lost due of software failure.
- $E(T)$: expected cost of software systems at time $T$.
- $\mu_y$: expected time to remove an error during testing phase.
- $W$: variable of time to remove an error during warranty period in operation phase.
- $\mu_w$: expected time to remove an error warranty period in operation phase, which is $E(W)$.
- $T_w$: period of warranty time.

**Assumptions**

1. There is a set-up cost at the beginning of the software development process.
2. The cost to do testing is a power function of testing time. In other words, at the beginning of the testing, the cost increases with a higher gradient, the growth slows down later.
3. The cost to remove errors during debugging period is proportional to the total time of removing all errors detected during this period.
4. The cost to remove errors during warranty period is proportional to the total time of removing all errors detected in the time interval $[T,T + T_w]$.
5. There is a risk cost due to the software failure after release the software.
6. It takes time to remove errors and we assume that the time to remove each error follows a truncated exponention distribution.

**Modeling Software Reliability with Multiple Failure-Typing and Imperfect Debugging [Lynch et al., 1994]**

Software developers have determined two main characteristics of the software development process: (1) no programmer is perfect, and thus when an error is removed, new errors can be introduced into the program; and (2) not all errors are created equal; that is, different errors have different implications and thus need different handling. A number of software reliability models [Kareer et al. 1990, Kapur et al. 1992 Leung et al. 1992, Yamada et al. 1984-1986, W.Kuo, 1983] incorporate one of these characteristics, but until now none has included both. When a failure occurs, the cause of the failure is identified and removed.

To ensure that the cause is perfectly fixed, the software is tested is for the same input and if a failure occurs again, the code is checked again. Two possibilities occur. The fault, which was thought to be perfectly fixed, has been imperfectly repaired and caused same type of failure again when checked on
the same input. However, it may also happen some other kind of failure occurs which might be due to the fact that the fault was perfectly removed but some other fault was generated while removing the cause of the failure. This is called error generation/introduction, which can be known only during the removal phase. On this section, we have incorporated the effect of latter type of imperfect debugging on the removal process. Here we present a model with multiple failure types and imperfect debugging for prediction of software reliability. In addition, the paper discusses cost models that can be used to determine the optimal time to be spent debugging. The software reliability model allows for three different types of errors: critical, major, and minor errors.

Critical errors are the most difficult to detect and the fairly expensive to remove. Minor errors are easy to detect and inexpensive to remove. The model also allows for the introduction of any of these types of errors during the removal of an error.

Software Reliability Model

Notations

\[ M(t) \] expected number of failures by time \( t \); \( m(t) = E[N(t)] \).

\[ N(t) \] counting process representing the cumulative number of failures detected by time \( t \).

\[ N(0) \] number of failures at time \( t = 0 \).

\( a \) expected initial error content.

\( b \) error detection rate per error at an arbitrary testing time.

\( b_j \) error detection rate per type \( j \) error, \( j = 1, 2, 3 \).

\( p_j \) content proportion of type \( j \) errors.

\[ d(t) \] error detection rate per error at testing time \( j \); \( d(t) = \frac{\lambda(t)}{[a-m(t)]} \).

\[ d_j(t) \] error detection rate per type \( j \) error at testing time \( j \).

\[ \lambda(t) \] intensity function or error detection rate; \( \lambda(t) = \frac{d[m(t)]}{dt} \).

\[ N_j(t) \] cumulative number of failures of type \( j \) error.

\[ n(t) \] the expected number of error detected plus the expected number of error remaining at time \( t \).

\[ \beta_j \] type \( j \) error introduction rate that satisfies, \( 0 \leq \beta_j < 1 \).

\[ M_j(t) \] expected number of type \( j \) errors by time \( t \).

Software Cost Model

To determine when a software package should be released for use, we must determine a cost model that accurately describes the cost incurred during the lifetime of a program.

Here a software cost model is discussed under the following assumptions:

1. The cost of debugging an error is cheaper during the development phase than during the operational phase.
2. The cost of removing a particular type of error is constant during the debugging phase.
3. The cost of removing a particular type of error is constant during the operational phase.
4. Critical error are more expensive to remove than major errors, which in turn are more expensive to remove than minor errors.
5. There is a continuous cost incurred during the entire time of the debugging period.

Cost-reliability Optimal Release Policy for Software Reliability Models Incorporating Improvements in Testing Efficiency [Huang, 2005]

Here we give a brief review of the SRGM with a generalized logistic testing effort function. Further, if the software managers wish to detect more faults that are difficult to find during regular testing, it is advisable to introduce new techniques. Also, we study the effect of introducing these new techniques...
or consultants for increasing the testing efficiency. Finally, we discuss 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 techniques during testing.

SRGM with Generalized Logistic Testing-Effort Function

Notations

\(M(t)\): expected mean number of faults detected in time \((0,t)\).

\(\lambda(t)\): failure intensity for \(m(t)\).

\(W(t)\): cumulative testing-effort consumption at time \(t\).

\(a\): expected number of initial faults.

\(r\): fault detection rate per unit testing-effort.

\(N\): total amount of testing effort eventually consumed.

\(a\): consumption rate of testing effort expenditures in the general logistic testing-effort function.

\(A\): consumption parameter in the generalized logistic testing effort function.

\(K\): structuring index.

5. CONCLUSION

This chapter incorporates fault removal efficiency into software reliability assessment.

Imperfect debugging is considered in the sense that sense all faults can be removed complete, and new faults can be introduced while removing existing ones. Both the fault removal efficiency and the fault introduced function can take a time-varying from.

Data collected from real applications [Lyu, 1996] show that proposed model provides the best fit and prediction (both the SSE and the AIC values are the lowest among all models), it also provides both, the information reliability measures, and also, some important in-process metrics including the fault removal efficiency and fault introduction rate. These metrics offer very useful information about the
development project management. With more careful data collection, more sophisticated analysis can be in this area.

We also presented a SRGM with generalized logistic TEF. It is a much more realistic model and more suitable for describing the software fault detection and removal process.

On the other hand, in practice, sometimes it is difficult for software developers to locate the faults log and test anomaly documents. Sometimes software managers may require in the developers to detect more faults due to schedule pressure. In this case, it is advisable to introduce new test techniques, which are fundamentally different from the methods in use. These test techniques can help developers get their product done quicker and more reliably. Thus, we further study the efficiency. Finally, we discussed the optimal release policy based on cost and reliability considering testing effort and efficiency. The procedure for determining the optimal release time has been discussed in detail and the optimal release time has been shown to finite.

REFERENCES


Corresponding Author

Somkant Khare* 

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