Optimum Software Reliability: A Literature Review

Gunjan Sethi\(^1\) and Poonam Rana\(^2\)

\(^1\)M.Tech. Research Scholar and \(^2\)Senior Lecturer,
Department of Computer Science and Engineering,
Ambala College of Engineering and Applied Research,
Devsthali, Ambala, Haryana, India
E-mail: gunjansethi87@gmail.com, rana.poonam1@gmail.com

Abstract

A number of analytical models have been proposed during the past 15 years for assessing the reliability of a software system. In this paper, we proposed a model to get the optimum software reliability & optimum cost subject to time. Now a days industry needs a software reliability product with optimum cost.

Keywords: Software reliability, Two dimensional software reliability growth model, Goodness of fit, optimal release problem.

Introduction

A metric which reflects the degree of program correctness and which can be used in planning and controlling additional resources needed for enhancing software quality. One such quantifiable metric of quality that is commonly used in software engineering practice is software reliability. A commonly used approach for measuring software reliability is via an analytical model whose parameters are generally estimated from available data on software failures. Reliability and other relevant measures are then computed from the fitted model. Software reliability is a probabilistic measure and can be defined as the probability that software faults do not cause a failure during a specified exposure period in a specified use environment [1]. Assessing software reliability in a testing phase of a software development process is one of the important issue to develop a highly reliable software system. Software reliability models are used for the prediction and estimation of software reliability [2]. A software reliability growth model is known as one of the fundamental technologies for quantitative software reliability assessment and playing an important role in software project management for producing a highly reliable software system [3]. Nonhomogeneous process (abbreviated as NHPP) model is to determine an
appropriate mean value function to denote the expected no of failures experienced up

appropriate mean value function to denote the expected no of failures experienced up
to a certain time. NHPP can be classified by the fault –detection time distribution [4].Software developing managers have a great interest in how to develop a reliable software product economically and when to release the software to the customers.

**Literature Review**

Software reliability models are used for the prediction and estimation of software reliability [2]. Jintao Zeng Jinzhong Li, Xiahoui Zeng, Wenlang Luo. In A Prototype System of Software Reliability Prediction and Estimation have proposed an approach for software reliability model selection based on experiences from history software projects [2]. In this paper it is considered that experiences from model selection of history projects which can be used to serve model selection of the current project. By using this approach two questions be solved how to measure the similarity of software projects and how to make full use of history project experiences. Shinji Inoue and Shigeru Yamada in Two-Dimensional Technologies describes a software reliability growth process depending on two-types of software reliability growth factor : Testing –time and Testing–effort factors[3].Two-dimensional software reliability measurement technologies enable us to conduct more feasible software reliability assessment than the one-dimensional (conventional) software reliability measurement approach . In this approach software reliability growth process depends only on testing-time. Two-dimensional software reliability growth modeling approaches for feasible software reliability assessment, and conduct goodness-of –fitness comparisons of our models with one-dimensional software reliability growth models. Shinji Inoue in Generalized Discrete Software Reliability Modeling With Effect of Program Size have proposed an approach for cost-optimal and cost-reliability-optimal software release policies[4],[5],[6].Cost-optimal software release policies based on generalized discrete binomial process model. Optimal software release problems which take both total software cost and reliability criteria into consideration simultaneously.

**Proposed Approach**

Optimization of software reliability is considered as objective with respect to cost as constraint. The expected software cost $E(T)$ and the software reliability $R(x/t)$, we have to determine the optimum release time that minimizes the expected software cost subject to attaining a desired reliability level.

\[
\text{Minimize } E(T) \\
\text{Subject to } R(x|T) > R_0
\]

The Software reliability $R(x/T)$ and the expected software cost $E(T)$.The Optimization problem can be formulated as

\{Maximize $R(x|T)$

\{Subject to $E(T) < C$
E (T) can be calculated as
\[
E(T) = 0 \int_{[C_3 t + \sum_{i=1}^{3} C_{i1} m_i(t)]} g(t) dt + T \int_{\infty}^{\infty} [C_3 T + \sum_{i=1}^{3} C_{i1} m_i(T)] + \sum_{i=1}^{3} C_{i2} (m_i(t) - m_i(T))] g(t) dt
\]

The Function E(T) represents testing costs per unit time and of fixing errors during testing incurred if the determination of the software life cycle is less than or equal to the software release time. On the other hand if the determination of the software lifecycle is greater than the software release time, then an additional cost factor should be involved i.e. the cost of fixing errors during the operation phase.

**Mathematical Model**

Let E (T) be the expected software cost.
- R (X|T) is the software reliability.
- Optimize the software reliability taking cost as a constraint.

So, the problem is formulated as:
- Optimize (E (T),
- Subject to R (X|T)

**Example**

To optimize the software reliability taking cost as a constraint let us assume that C3, C_R, C_{i1} and C_{i2} for i=1,2,3 be the cost and the optimal value of T say T_{rel}, that maximizes R(X|T) subject to the cost constraint is determined from

If E (T*) > C_R Then there is No solution

Else T_{rel} = \{T \geq T*: T = E^{-1} (C_R)\}; End if.

This shows that if E (T*) > CR, the minimum software system cost required to develop and debug the program exceeds the maximum amount allowable. Therefore, it is impossible to produce the software under these conditions. Similarly , if E(T*) \leq CR, and as the reliability of software continually improves with testing and debugging time, then the program should be debugged until the cost constraint is binding, implying that additional debugging will violate the constraint.

The software reliability function can be formulated as

\[
R (x|t) = e^{-\sum_{i=1}^{3} (a p_i \beta_i (1-\beta_i) (e^{(1-\beta_i)b_i t} + e^{-(1-\beta_i)b_i x}))}
\]

In this equation:

- An expected number of software errors to be eventually detected
- b_i error detection rate per type i error i = 1, 2, 3; 0 < b_1 < b_2 < b_3 < 1
- p_i content proportion of type i errors
- \beta_i type i error introduction rate that satisfies, 0 \leq \beta_i < 1
Figure 1. Flowchart representing proposed technique.

Table 1. Failure Data

<table>
<thead>
<tr>
<th>Week</th>
<th>Hours</th>
<th>Critical</th>
<th>Major</th>
<th>Minor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>62.5</td>
<td>0</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>68</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>62</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>66</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>73</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>73.5</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>92</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>71.4</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>64.5</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>64.7</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>36</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>54</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>39.5</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>68</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>61</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>62.6</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>98.7</td>
<td>0</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Conclusions
In today’s life we are dependent on some of the products that are highly critical in nature so such kind of products need a high level of reliability. To achieve high level of reliability a high cost is required which is not always possible. So, the objective of the work is to maximize the reliability with respect to software cost using heuristic based approach such as genetic algorithm.

References