Assessing Software Reliability of Goel-Okumoto model

G.Lavanya
Student of 4/4 B.Tech,
Dept of IT, VRSE College
JNTUK, Vijayawada,
Andhra Pradesh

M.Rojalakshmi
Student of 4/4 B.Tech,
Dept of IT, VRSE College
JNTUK, Vijayawada,
Andhra Pradesh

M.Vasundhara
Student of 4/4 B.Tech,
Dept of IT, VRSE College
JNTUK, Vijayawada,
Andhra Pradesh

Dr.Y.Sangeetha
Assistant Professor,
Dept of IT, VRSE College
JNTUK, Vijayawada,
Andhra Pradesh

ABSTRACT:
Statistical Process Control (SPC) is the best choice to monitor software reliability process. It assists the software development team to identify and actions to be taken during software failure process and hence, assures better software reliability. In this project we propose a control mechanism based on the cumulative observations of failures which is ungrouped data using an infinite failure mean value function G-O model, which is Non-Homogeneous Poisson Process (NHPP) based. Maximum likelihood Estimation method (MLE) approach is used to estimate the unknown parameters of the model.

Keywords: Statistical Process Control, Software Reliability, Non-Homogeneous Poisson Process (NHPP)

1. INTRODUCTION
SOFTWARE RELIABILITY:
Software Development organizations have a challenging task of meeting two requirements simultaneously. The first one being able to predict and meet Business Growth opportunities and the second one is being providing software with minimum fault percentage. For Quantitative control of software testing process and also to measure the reliability of the software, SRGMs are used. Many models have been developed in the past by estimating initial fault number and their effect on software operations and also to predict software reliability. Software Reliability Model is categorized into two, one is static model and the other one is dynamic model. Dynamic models observe the temporary behaviour of debugging process during testing phase. In Static Models, modelling and analysis of program logic is done on the same code. A Model which describes about error detection in software Reliability is called Software Reliability Growth Model. We assume that the software system is subject to failure randomly due to software errors. Whenever, there is a software failure, it is removed and assumed, that new errors are not introduced. It is a probability that software component will produce an incorrect output. Software does not wear out, software can continue to operate after a bad result. System reliability is measured by counting the number of operational failures and relating these to demands made on the system at the time of failure. Software Reliability plays an important role in software developments due to large scale applications of system software’s. Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. It is the ability of a computer program to perform its intended functions and operations in a system's environment, without experiencing failure (system crash).

1.1 NON-HOMOGENEOUS POISSON PROCESS (NHPP):
It provides an analytical framework for describing the software failure phenomenon during testing. It is used to estimate the mean value function of the cumulative number of failures experienced up to certain point in time. It is similar to an ordinary Poisson process, except that the average rate of arrivals is allowed to vary with time. Many applications that generate random points in time are modelled more faithfully with such non-homogeneous processes. The mathematical cost of this generalization, however, is that we lose the property of stationary increments.

1.2 STATISTICAL PROCESS CONTROL (SPC):
It is the application of statistical method to monitor and control a process to ensure that it operates as it full potential to produce conorming product. It is an analytical decision making tool which allows you to see when a process is working correctly and when it is not. Variation is present in any process, deciding when the
variation is natural and when it needs correction is the key to quality control. Quality data is in the form of product or process measurements are obtained in real time during manufacturing. This data is then plotted on graph with pre-determined by the capability of the process, whereas specifications limits determined by client’s needs.

2. LITERATURE SURVEY/CASE STUDY:
Software reliability modelling has, surprisingly to many, been around since the early of 1970’s The basic approach is to model past failure data to predict future behaviour. This approach employs either the observed number of failures discovered per time period or the observed time between failures of the software. Previously they used one of the method Particle Swarm Optimization (PSO). PSO is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions (particles) and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position and is also guided towards the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm towards the best solutions. But the drawback in this PSO method that easily suffers from the partial optimism, which causes less extract at the reputation of its speed and direction. Then the method cannot work out the problem of scattering and optimization. Also PSO cannot work without a problem such as the solution to the energy field and the moving rules of the particles in energy field in a non coordinate system. We considered one of the previous software reliability model Weibull Model one of the most widely used models for hardware reliability modelling is the Weibull distribution. It can accommodate increasing, decreasing, or constant failure rates because of the great flexibility to the infinite failures category and is of binomial type using the Musa Classification. But we considered G-O Model to calculate estimation accuracy. When compared to other models G-O Model got first rank. In this paper we propose a control mechanism based on the cumulative observations of failures which is ungrouped data using an infinite failure mean value function G-O model, which is Non-Homogenous Poisson Process (NHPP) based. Maximum Likelihood Estimation method (MLE) approach is used to estimate the unknown parameters of the model.

3. NHPP Exponential Models
3.1 Goel-Okumoto Model:
This is a continuous time – independent and identical error behaviour model. This is extension of J-M Model with the possibility of imperfect debugging. It represents the number of errors X (t) in the software at time t by a Markov Process whose transition probabilities are governed by the proud of imperfect debugging. Time between the transactions of x (t) is taken to be exponentially distributed with rates dependent on the current fault content of the system. The Prediction of the model can be given as \[ m(t) = a \left(1 - e^{-bt}\right). \]

The Goel-Okumoto model is based on the following assumptions:
1. All faults in a program are mutually independent from the failure detection point of view.
2. The number of failures detected at any time is proportional to the current number of faults in a program. This mean that the probability of the failures for faults actually occurring, i.e., detected, is constant.
3. The isolated faults are removed prior to future test occasions.
4. Each time a software failure occurs, the software error which caused it is immediately removed, and no new errors are introduced.

This is shown in the following differential equation:

\[ \frac{\partial m(t)}{\partial t} = b[a - m(t)] \]

Where \( a \) is the expected total number of faults that exists in the software before testing and \( b \) is the failure detection rate or the failure intensity of a fault. The mean value of this differential equation is given by
This model is known as the Goel-Okumoto model.

3.2 Maximum Likelihood Estimation Method (MLE):

In statistics, MLE is a method of estimating the parameters of a statistical model. When applied to a data set and given a statistical model, maximum-likelihood estimation provides estimates for the model's parameters. The method of maximum likelihood corresponds to many well-known estimation methods in statistics. MLE would accomplish the estimates by taking the mean and variance as parameters and finding particular parametric values that make the observed results the most probable.

The estimate of parameters \(a\) and \(b\) using MLE method can be obtained by solving the following equations:

\[
\begin{align*}
\frac{n}{b} &= \sum_{i=1}^{n} s_i + \frac{n s_n}{(1 - e^{-bs_n})} e^{-bs_n} \\
\frac{n}{a} &= n (1 - e^{-bs_n})
\end{align*}
\]

Let \(\hat{a}\) and \(\hat{b}\) be the MLE of parameters \(a\) and \(b\) respectively. We can then obtain the MLE of the mean value function (MVF) and the reliability function as follows:

\[
\begin{align*}
m(t) &= a[1 - e^{-bt}] \\
R(x|t) &= e^{-[a(1 - e^{-bt}) + e^{-bt} - e^{-(bt + x)}]}
\end{align*}
\]

It is of the interest to determine the variability of the number of failures at time \(t\), \(N(t)\). One can approximately obtain the confidence intervals for \(N(t)\) based on the Poisson distribution as

\[
m(t) - z_{\alpha/2} \sqrt{m(t)} \leq N(t) \leq m(t) + z_{\alpha/2} \sqrt{m(t)}
\]

Where \(z_{\alpha/2}\) is \(100(1+\alpha)/2\) percentile of the standard normal distribution, i.e., \(N(0,1)\).

Table 1: Real-time Command and Control Data (in an one-hour interval)

<table>
<thead>
<tr>
<th>Hour</th>
<th>Number of failures</th>
<th>Cum Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

It should be noted that this data set belongs to a concave class, therefore, it seems reasonable to use the Goel-Okumoto NHPP model to describe the failure process of the software system. From the failure data, the two unknown parameters, \(a\) and \(b\), can be obtained using the above equation and the estimated values for the two parameters are

\[
a = 142.3153 \quad b = 0.1246
\]

Recall that \(a\) is an estimate of the expected total number of failures to be eventually detected and \(b\) represents the number of faults detected per fault per unit time (hour). The estimated mean value function and software reliability function, respectively, are

\[
m(t) = 142.3153[1 - e^{-0.1246t}] \\
R(x|t) = e^{-(142.3153) e^{0.1246(t-x)} - e^{-(0.1246)(t+x)}}
\]

The above two functions can be used to determine when to release the software system or the additional testing effort required when the system is ready for the release. Let us assume that failure data from only 1
hours of testing are available, a total of 6 failures are observed from the above table. Based on this data and using MLE method, the estimated values for the two parameters are 

\[ a=138.3779 \text{ and } b=0.1334 \]

and the estimated mean value function becomes

\[ m(t)=138.3779(1-e^{-0.1334t}) \]

The reliability of the software system is

\[ R(x|t)=e^{-138.3779[1-0.1334t+e^{-0.1334t}]} \]

As estimate number of remaining errors after 1 hours of testing is 16.38 with a 90% confidence interval of (4.64,28.1). Similarly the estimated current software reliability for the next hour is 0.121 and the corresponding 90% confidence interval is (0.019, 0.31).

Next, suppose the problem of interest is to know how much additional testing is needed in order to achieve an acceptable number of remaining errors so that the software can be released for operational use. For example, we would want to release the software if the expected number of remaining errors is less than or equal to 10. In the above analysis, we learned that the best estimate of the remaining errors in the software after 16 hours of testing is about 17. Therefore, testing has to continue in the hope that additional faults can be detected. If we were to carry on a similar task after each additional hour of testing, we can expect to obtain another seven additional errors during the next 4 hours. In other words, the expected number of remaining errors after 20 hours would be 9.8 so that the above objective would be met.

<table>
<thead>
<tr>
<th>Testing time T</th>
<th>a</th>
<th>b</th>
<th>Remaining errors</th>
<th>Reliability R(0.1/T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>138.3779</td>
<td>0.1333</td>
<td>16.4</td>
<td>0.8049</td>
</tr>
<tr>
<td>2</td>
<td>133.7050</td>
<td>0.1432</td>
<td>11.7</td>
<td>0.8466</td>
</tr>
<tr>
<td>3</td>
<td>141.2543</td>
<td>0.1274</td>
<td>14.3</td>
<td>0.8349</td>
</tr>
<tr>
<td>4</td>
<td>139.7190</td>
<td>0.1304</td>
<td>11.7</td>
<td>0.8591</td>
</tr>
<tr>
<td>5</td>
<td>138.8495</td>
<td>0.1323</td>
<td>9.8</td>
<td>0.8786</td>
</tr>
<tr>
<td>6</td>
<td>140.3408</td>
<td>0.1290</td>
<td>9.3</td>
<td>0.8871</td>
</tr>
<tr>
<td>7</td>
<td>140.1002</td>
<td>0.1296</td>
<td>8.1</td>
<td>0.9010</td>
</tr>
<tr>
<td>8</td>
<td>141.9104</td>
<td>0.1255</td>
<td>7.9</td>
<td>0.9060</td>
</tr>
<tr>
<td>9</td>
<td>142.0264</td>
<td>0.1252</td>
<td>7.0</td>
<td>0.9162</td>
</tr>
<tr>
<td>10</td>
<td>142.3153</td>
<td>0.1246</td>
<td>6.3</td>
<td>0.9248</td>
</tr>
</tbody>
</table>

4. RESULT OBTAINED IN THE FORM OF GRAPH:

![Failure rate Comparison graph for Software reliability performance](image)
5. CONCLUSION:
From the analysis of the data it is observed that we are able to come up with an early conclusion about the reliability or unreliability of a software product. The results of the graph describes that reliability is maintained at a constant level but whereas errors in the beginning are high and gradually decreases. Therefore, we conclude that assessing software reliability of Goel-Okumoto is performing well when compared to others models.

6. REFERENCES: