A Software Reliability GEP Model Based on Usage Profile

S.J. Sun¹, J. Xiao²
¹Hebei University of Engineering
HanDan Guangming S. Main Street No. 199, 0310 8579377
²Shijiazhuang Non-commissioned Officer School of Chinese People's Armed Police Force
Shijiazhuang Xuefu Road No. 81
*corresponding author, e-mail: sunshengjuan@163.com, xiaojing8785@163.com

Abstract

Abstract: In this paper, software reliability measurement is studied from the perspective of usage profile. Reliability parameters on each usage profile are expressed with unascertained rational numbers. Since failure process of each usage profile can be affected by multiple factors such as software attributes and users imports etc., GEP (gene expression programming) is introduced into software reliability modeling, to study comprehensive force among the factors. Finally, a software reliability GEP model based on usage profile is constructed. With the new model, users can confirm software reliability according to its usage, thus enabling software reliability to a more objective assessment. The new model is experimented to measure reliability of SYS in Musa data, and proved with a satisfactory performance.

Keywords: software reliability, usage profile, unascertained characteristic, GEP

Copyright © 2012 Universitas Ahmad Dahlan. All rights reserved.

1. Introduction

The rapid development of information technology enables software to be applied to various social fields, and software quality becomes the focus accordingly. Software with poor quality not only increases significantly the maintenance cost of developers and the use cost of users, but also brings other responsibilities risks. In some crucial applications, such as aviation reservation system, bank settlement system, military defense and nuclear power plant safety control system, using software with quality problems may result in disastrous consequences.

Software quality can be estimated by historical failure data and development data. Software reliability model is such a measurement tool. It can quantitative evaluate and predict software reliability in different stages of software life cycle. A good reliability model can predict software reliability action accurately, and this is of great importance to software resource allocation and software market decision–making [1]. Since 1970s, research on software reliability model domain gains greatly development, a great many models have been thrown into practice, and software reliability has already stepped into engineering stage from conceptual stage [2]. However, facing increasing complication of software and their development course, reliability models still appears inherent shortcomings.

2. Analysis on Software Failure Mechanism

Suppose software input is composed of m values which can be char, numeric, boolean or struct, and it is denoted as a m-dimension vector \( (i_1, i_2, \ldots, i_m) \). All input vectors construct an m-dimension input space. Similarly, suppose software output is composed of n values, \( o_1, o_2, \ldots, o_n \), and all output vectors construct n-dimension output space. Given an input vector, an output vector can be calculated by the software. So the software is essentially a mapping transformation:

\[
O = P(I), I^m \rightarrow O^n
\]

While bugs exist in software, some data in the input space can trigger codes with error, and then map to an unexpected output, which would result in a software failure phenomenon [3]. This mechanism is shown as Fig. 1.
Noting locations of error codes in the program are fixed, the input modules which cause failures can be confirmed. Thus, for a program itself, its failures are objective certain. However, software failure process could represent uncertainty, since test process can be affected by multiple factors such as software attributes, skill of test person and test environment etc.. So in order to describe software failure process accurately, it is necessary to consider comprehensive force among the multiple factors.

![Software failure mechanism](image)

Software processing to inputs means a software operation to users. The assembly of these operations and their usage frequencies is called a usage profile. That is to say, given software, its reliability can be only confirmed by usage profiles. As usage profiles change, software reliability changes accordingly. So it is dishonest to discuss software reliability setting usage profiles aside. But so many classic models are quite so. Failure rate is supposed subjectively to a certain statistical distribution, and this is equivalent to restricting one usage profile to make the actual operation meet the assumption. Software reliability can not be evaluated accurately in such way. Therefore, analysis software failure characteristics from the perspective of usage profiles would be more objective and accurate.

3. Software Reliability Model Based on Usage Profile

Software usage profiles are completely certain themselves. That means, given software, its reliability is certain. But subjective uncertainty to software reliability actual exists because test engineers can not select test case from almost endless input space in software test stage due to various reasons. This subjective and cognitional uncertainty is called as unascertained characteristic.

3.1. Related Knowledge about Unascertained Theory

There are some questions such as “how much does this building weigh? Suppose us to visit sb., he isn’t in, but where is he?” Objectively, the answers are certain. But we do not know. So they are uncertain to us, also to decision makers. This kind of uncertainty is different from uncertainty of fuzzy information, stochastic information and grey information. It is a kind of subjective and cognitional uncertainty, and is called unascertained characteristic. Information with unascertained characteristic is called unascertained information [4].

Unascertained mathematics theory mainly studies how to express and deal with unascertained information. Among all the means, unascertained rational number is the simplest and the most extensive way. The mathematics definition of unascertained rational number is described as below.

For a random interval \([a,b]\), \(a = x_1 < x_2 < \cdots < x_n = b\), if function \(\varphi(x)\) satisfies:
\[
\varphi(x) = \begin{cases} 
\alpha_i, & x = x_i \ (i = 1,2,\ldots,n) \\
0, & \text{others} 
\end{cases}
\]

and

\[
\sum_{i=1}^{n} \alpha_i = \alpha, 0 < \alpha \leq 1.
\]

\([a,b]\) and \(\varphi(x)\) can constitute a \(n\)-step unascertained rational number, denoted as \([[a,b], \varphi(x)]\), in which \(\alpha\) is the total confidence degree, \([a,b]\) is the value interval, and \(\varphi(x)\) is the density function of confidence degree [4].

Let \(A = [[a,b], \varphi(x)]\) is the unascertained rational number of above expression. Its 1-step unascertained rational number is the expectation of \(A\), which is denoted as:

\[
E(A) = \left[ \frac{1}{\alpha} \sum_{i=1}^{n} x_i \alpha_i, \frac{1}{\alpha} \sum_{i=1}^{n} x_i \alpha_i \right], \varphi(x)
\]  

(1)

where,

\[
\varphi(x) = \begin{cases} 
\alpha, & x = \frac{1}{\alpha} \sum_{i=1}^{n} x_i \alpha_i \\
0, & \text{others}
\end{cases}
\]

\(E(A)\) is also called unascertained expectation, and expectation or mean for short [1].

3.2. Software Reliability Unascertained Description Based on Usage Profile

Reliability of a software with \(n\) usage profiles can be denoted as:

\[
R = [[R_{\min}, R_{\max}], \varphi(R)]
\]  

(2)

where,

\[
R_{\min} = \min(R_j), \\
R_{\max} = \max(R_j), \\
j = 1,2,\ldots,n,
\]

\(R_j\) is software reliability on the \(j\) th usage profile,

\[
\varphi(R) = \begin{cases} 
\alpha_j, & R = R_j \\
0, & \text{others}
\end{cases}
\]

\(\alpha_j\) is usage frequency on the \(j\) th usage profile.

4. A New Diversity Strategy for Gene Expression Programming to Describe Failure Characteristics of Each Usage Profile

On analysis of software failure mechanisms, it is necessary to consider the comprehensive force among multiple factors, to describe software failure characteristics of each usage profile. GEP (Gene Expression Programming) is an evolutionary algorithm inheriting the advantages of the traditional genetic algorithm (GA) and genetic programming (GP). It is an efficiency data mining technique which has been widely applied to symbolic regression, function finding, classification rule mining and other fields [5]. It can be adopted to express comprehensive force among multiple factors, and is very suitable for mining software failure characteristics [6-8]. Therefore, adopting GEP to construct software reliability models based on usage profiles, we can obtain software reliability on each usage profile.

4.1. GEP Fundamental

The implementation techniques of GEP include encoding, fitness function selection, genetic operators, transposition operators, recombination operators, and numerical variables [9-12]. Now we just introduce the parts that will be improved in this paper.
4.1.1. Fitness Function Selection

Individuals that represent problem solutions need to be evaluated in all evolutionary algorithms. In GEP the solution is a computer program, or more exactly an expression. So the evaluation is to be completed by the fitting degree of data calculated by the expression and the training data. The following three ways are usually adopted.

\[ f_i = \sum_{j=1}^{C_i} \left( M - \left| C_{i,j} - T_j \right| \right) \]  \hspace{1cm} (3)

\[ f_i = \sum_{j=1}^{C_i} \left( M - \left| \frac{C_{i,j} - T_j}{T_j} \times 100 \right| \right) \]  \hspace{1cm} (4)

\[ if \ n \geq \frac{1}{2} C_i, then \ f_i = n, else \ f_i = 1 \]  \hspace{1cm} (5)

Where M is the range of selection, and C(i,j) is the value returned by the individual program i for fitness case j (out of Ct fitness cases), and Tj is the target value for fitness case j, and n is the number of correct cases. Note that formula (3) and (4) can be used to solve any symbolic regression problem, but formula (5) to logic problems. In the design of fitness function, the goal is very clear that to make the evolutionary direction of the system in accordance with requirements.

4.1.2. Mutation Operator

According to Candida’s experiments [5], we know that the mutation operator is the most basic and most efficient operator among all genetic operators. Mutation operator can adjust parts of gene values of the individual encoding string, to make GEP search the local space and improve the local search ability. Besides, mutation operator can change encoding structure, to maintain the population diversity, and prevent or reduce premature and jump out of local optimal solution.

Mutation operator acts on a single chromosome, and tests randomly on each code of the chromosome. When the mutation probability Pm meets a certain value (typically is 0.044), the code is re-generated. To ensure the same organizational structure, the code can be varied to any symbol of the function set and terminal set if mutation occurred in the head. Conversely, the code could be symbol of terminal set when in tail. It is can be predicted the structure of new individual generated through mutation is always correct.

4.2. A New Diversity Strategy for Gene Expression Programming

4.2.1. Block Strategy

Genetic operators play an important role in the evolutionary results quality. If they are designed unreasonably, some extraordinary individuals generated in the early evolutionary could multiply rapidly and fill the population positions after several generations. So the local optimal solution, also called premature phenomenon is coming. Another way, the algorithm is close to convergence in the later stage of evolutionary, and the fitness difference between individuals is smaller. So the potential of optimization reduced, and the result is tend to purely random selection and hardly a global optimal solution. In this paper, we adopt blocking population to make sure the population diversity of each generation. The scheme is as follows.

Step 1, suppose \( f_j, i = 1, 2, \cdots, n \) is the fitness of individual \( x_i \), order individuals by \( f_j \), a block of 20, the population is divided into \( m \) blocks \( B_j, j = 1, 2, \cdots, m \) (number of \( B_m \) is permitted less than 20). \( f_{j-\text{max}} \) (the fitness maximum of \( B_j \)) is less than the fitness minimum of \( B_{j+1} \) \( \left( f_{(j+1)-\text{min}} \right) \), that is \( f_{j-\text{max}} < f_{(j+1)-\text{min}} \).

Step 2, as in the individual fitness of each block are very close, linear or power function transformation method is adopted for scaling the fitness function, and then individuals are selected to genetic operations follow the roulette wheel or tournament method.

Step 3, since the individuals’ goodness differences in the blocks, mutation operator is reset respectively to each other block, like a smaller mutation probability set to individuals in the
block with a high goodness and larger to low goodness, in order to ensure high population diversity.

In view of this scheme, we need to redesign fitness function and improve mutation operator.

(1) Fitness Function
On GEP-based symbolic regression problems, the two evaluation models proposed by Candida own their inherent shortcomings. In statistics, it is more usually to employ \( R^2 \) (Coefficient of Determination) to evaluate the fit degree of two sets of data. The calculation formula is as below.

\[
R^2 = 1 - \frac{SSE}{SST}
\]  

(6)

in which \( SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \), \( SST = \sum_{i=1}^{n} (y_i - \bar{y})^2 \). \( y_i \) is the real observed value, and \( \bar{y} \) is the average one of observed values, and \( \hat{y}_i \) is the regressed value. SSE is residual sum of squares of the observed values and the regressed values, and summation of SSE and SSR(regression sum of squares \( \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 \)). So, we design the fitness function like this:

\[
f = n \times 100 \times R^2 \quad (n \text{ is the sample size})
\]  

(7)

\( \because \; SSE < SST \Rightarrow 0 < R^2 < 1 \). It can be known the range of \( f \) is \((0, n \times 100)\). When the individual fitness of each block are very close, fitness of the next generation can hardly be improved obviously, which would lower evolutionary efficiency. So we make fitness linear amplified by multiplying the factor \( n \times 100 \) (n is the sample size).

(2) Mutation Operator
We set dynamic mutation probability in this paper, in order to make mutation operator self-adaptive. Mutation probability function is designed as follows.

\[
P_{m} = P_{M} \times e^{-\frac{\bar{f}_i + \bar{f}}{c} - c}
\]  

(8)

where \( P_{m} \) is mutation probability of the current block, and \( P_{M} \) is a constant set before evolutionary with a range of \((0, 0.15)\), and \( \bar{f}_i \) is average fitness of the current block and its maximum is \( \bar{f}_i \) \( \text{max} \), while \( c = n \times 100 \) is the maximal fitness.

It can be easily learned from formula (8) that \( P_{m} \) of each block is in inverse ratio to the average fitness, also to generations (or \( \bar{f}_i \) \( \text{max} \)). The value range of \( P_{m} \) is \( \left[ 0, \frac{1}{e} P_{M} \right] \).

4.2.2. BS-GEP Description
Every individual mutates on a fixed probability in the classic GEP algorithm, which affect population diversity seriously. We brought out a scheme based on block strategy to the mutation operator. BS-GEP algorithm structure is shown in Fig. 2.

5. Experimental Analysis
We adopt software reliability unascertained model based on usage profile to measure software SYS from Musa. SYS has three major usage profiles: SYS1, SYS2 and SYS3. Their usage frequency is 0.15, 0.80 and 0.05 respectively [6]. Failure data SYS1, SYS2 and SYS3 are collected from testing cases of the three usage profiles. We adopt GEP to model on their cumulative time of failures \( T \) (also means the next failure time). Parameters of the evolutionary algorithm in the test are as shown in Table 1. From Fig.2, it is apparently that the new algorithm adds the mutation rate reset in every generation contrast to the classic GEP.
Run the evolutionary program in the mixed environment of VC++ and Mathematica. After 1000 generations of evolution, we get preferable adaptive models and their structures expression are as:

\[
T_{\text{GEP, SYS1}} = 3109.99 - \frac{3574.13}{x^2} - 178.833x + 5.4358x^2 + 656.112\sin\left(\frac{x}{5}\right)
\]  

\[
T_{\text{GEP, SYS2}} = 56.0347 - 17.0796e^{11} + 26690.19e^{-\frac{2}{x}} - \frac{91779.57}{x^3} + \frac{90505.97}{x^5} - 1337.31x + 65.787x^2 - 1.0034x^3 + 0.0071x^4 + 870.194\sin\left(\frac{x}{3}\right) + 2945.56\sin(2x)
\]  

\[
T_{\text{GEP, SYS3}} = 537939+1.5022e^{11} - \frac{518199}{x^5} + 0.6358e^2 - 0.0019x^7 + 88.558\sin x + 7.4718\sin(2x) - 1.0562\sin(1x)
\]  

The three figures following give out cumulative time simulation results of the new models on usage profile SYS1, SYS2 and SYS3 respectively.

---

**Table 1 Parameters Sets of GEP**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>60</td>
<td>Maximum of Generations</td>
<td>1000</td>
</tr>
<tr>
<td>Gene Number</td>
<td>5</td>
<td>Head Length</td>
<td>6</td>
</tr>
<tr>
<td>Function Set</td>
<td>+, -, ×, /, V</td>
<td>Terminal Set</td>
<td>{1, 0, 1, · · · , 9}</td>
</tr>
<tr>
<td>Select Operator</td>
<td>Roulette Wheel</td>
<td>Mutation Operator</td>
<td>0.044</td>
</tr>
<tr>
<td>Transposition</td>
<td>0.1</td>
<td>Recombination Operator</td>
<td>0.3</td>
</tr>
<tr>
<td>Fitness Function</td>
<td>( f_i = \sum_{j=1}^{n} M - \frac{C_{ij} - T_s}{T_s} * 100 ) (M=100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terminal condition</td>
<td>Maximum of Generations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 3. Simulation Result of Model for SYS1

Figure 4. Simulation Result of Model for SYS2

Figure 5. Simulation Result of Model for SYS3

So, we can calculate the 66th MTBF (mean time between failures) on usage profile SYS1, $MTBF_{SYS1} = 527.9855$, the 76th MTBF on usage profile SYS2, $MTBF_{SYS2} = 2720.9866$, the 172th MTBF on usage profile SYS3 $MTBF_{SYS3} = 48.5621$. In actual measurement of the
three usage profiles, the next software failures intervals is 543, 2716 and 46 respectively. It is thus clear that the new models has quite high prediction accuracy. Moreover, Fig.3-5 above show that it is very clear that GEP model has a high prediction accuracy and can fit well with real failure data (Fitness represents error between the predictive value and the real one.) .

MTBF of SYS can be expressed as such an unascertained rational number:

\[
MTBF_{SYS} = [(48.5621, 2720.9866), \varphi(MTBF)]
\]  

where,

\[
\varphi(MTBF) = \begin{cases} 
0.15, MTBF_{SYS} = 537.9855 \\
0.80, MTBF_{SYS} = 2720.9866 \\
0.05, MTBF_{SYS} = 48.5621 
\end{cases}
\]  

It means that the next failure interval is predicted as 537.9855 when SYS is on the usage profile SYS1. Because SYS1 runs on a probability of 0.15, the software failure time interval values 537.9855 on the probability of 0.15. In the same way, the next failure interval on usage profile SYS2 can be valued 2720.9866 on the probability of 0.8, and the next failure interval on usage profile SYS3 can be valued 48.5621 on the probability of 0.05. Substituting them into the basic software reliability formula, we can be obtained software failure rate, reliability function and other parameters which can represent software reliability on each usage profile. These parameters can also be expressed in unascertained rational number. In this way, software reliability can be confirmed according to its usage, even expressed as an unascertained expectation using unascertained mathematics theory when the whole software reliability parameters need to be evaluated comprehensively.

6. Conclusion

In this paper, software failure mechanism is analyzed firstly. On this foundation, software reliability measurement is studied from the perspective of usage profiles and reliability parameters of each usage profile are expressed with unascertained rational numbers. Gene expression programming is introduced to dig out software failure characteristics. Finally, a software reliability GEP model based on usage profiles is constructed. The new model is experimented to measure reliability of SYS in Musa data, and proved with a satisfactory performance. In conclusion, analysis of software failure characteristics from the perspective of usage profiles is more objective and accurate, and unascertained theory provides mathematics basis for its parameters expressions. It is worth further study.

Acknowledgments

The authors wish to thank the Natural Science Foundation of Hebei Province for contract F2010001040 under which the present work was possible.

References


