Volume : 1 | Issue : 11 | November 2012 ISSN - 2250-1991

Research Paper Computer Science

Study of Genetic Approach in Estimating Reliability of Component Based Software

*Harish Rathod **Mahesh Parmar

* Dept. of CSE/IT, B. H. Gardi College of engineering & Technology Rajkot, India

** Dept. of CSE/IT, B. H. Gardi College of engineering & Technology Rajkot, India

ABSTRACT

Integration of commercial of the shelf (COTS) software components is increasing rapidly with the development of computer technology. It becomes necessary to ensure the reliability of components as well the reliability of overall system. This paper presents a new approach to analyze the reliability of component-based software, based on the reliabilities of the individual components. In order to realize the reuse of components effectively in Component Based Software Development, it is required to measure the reliability of components. In this paper, we adopt Genetic Algorithms based approach to estimate the reliability of component.

Keywords :

I. Introduction to CBSE

Component Based Software Engineering (CBSE) is a paradigm that aims at constructing and designing systems using a pre-defined set of software components explicitly created for reuse. According to Clements [1], CBSE embodies the "the 'buy, don't build'-- philosophy". He also says about CBSE that "in the same way that early subroutines liberated the programmer from thinking about details, CBSE shifts the emphasis from programming to composing software systems". Reliability: Software reliability is defined as the probability of failure –free software operation for a specified period of time in a specified environment. The reliability of a software product is usually defined to be "the probability of execution without failure for some specified interval of natural units or time" [2]. This is an operational measure that varies with how the product is used. Reliability of a component is measured in the context of how the component will be used. That context is described in an operational profile. Reliability of a piece of software may be computed or measured. If the software has not been built yet, its reliability can be computed from a structural model and the reliabilities of the individual parts that will be composed to form the software. There is error associated with this technique due to emergent behaviors, i.e. interaction effects, which arise when the components must work together. If the software is already assembled, the reliability can be measured directly. The measurement is taken by repeatedly executing the software over a range of inputs, as guided by the operational profile. The test results for the range of values are used to compute the probability of successful execution for a specific value. The error associated with this approach arises from the degree to which the actual operation deviates from the hypothesized operation assumed in the operational profile.

II. What is reliability ?

Reliability is the probability that an item will perform a required function without failure under stated conditions for a stated period of time [3]. Reliability refers to the consistency of a measure. A test is considered reliable if we get the same result repeatedly. For example, if a test is designed to measure a trait (such as introversion), then each time the test is administered to a subject, the results should be approximately the same. Unfortunately, it is impossible to calculate reliability exactly, but there several different ways to estimate reliability.

According to ANSI, Software Reliability is "the probability of failure-free software operation for a specified period of time in a specified environment". Software Reliability is also an important factor affecting system reliability [4].

III. Genetic algorithm

Genetic Algorithm

Genetic Algorithms (GAs) were developed by Prof. John Holland and his students at the University of Michigan during the 1960s and 1970s.

To use a genetic algorithm, you must represent a solution to your problem as a genome (or chromosome) [5]. The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s). The three most important aspects of using genetic algorithms are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and implementation of the genetic operators. Once these three have been defined, the generic genetic algorithm should work fairly well. Beyond that you can try many different variations to improve performance, find multiple optima (species - if they exist), or parallelize the algorithms [10].

However, genetic algorithms do not appear to suffer from local minima as badly as neural networks do. Genetic algorithms are based on the model of evolution, in which a population evolves towards overall fitness, even though individuals perish. Evolution dictates that superior individuals have a better chance of reproducing than inferior individuals, and thus are more likely to pass their superior traits on to the next generation. This "survival of the fittest" criterion was first converted to an optimization algorithm by Holland in 1975 [6], and is today a major optimization technique for complex, nonlinear problems. In a genetic algorithm, each individual of a population is one possible solution to an optimization problem, encoded as a binary string called a chromosome. A group of these individuals will be generated, and will compete for the right to reproduce or even be carried over into the next generation of the population. Competition consists of applying a fitness function to every individual in the population; the individuals with the best result are the fittest. The next generation will then be constructed by carrying over a few of the best individuals, reproduction, and mutation.

Reproduction is carried out by a "crossover" operation, similar to what happens in an animal embryo. Two chromosomes exchange portions of their code, thus forming a pair of new individuals. In the simplest form of crossover, a crossover point on the two chromosomes is selected at random, and the chromosomes exchange all data after that point, while keeping their own data up to that point. In order to introduce additional variation in the population, a mutation operator will randomly change a bit or bits in some chromosome(s). Usually, the mutation rate is kept low to permit good solutions to remain stable. According to Sultan H. Aljahadi and Mohammed E. El-Telbany [7], the two most critical elements of a genetic algorithm are the way solutions are represented, and the fitness function, both of which are problem-dependent. The coding for a solution must be designed to represent a possibly complicated idea or sequence of steps. The fitness function must not only interpret the encoding of solutions, but also must establish a ranking of different solutions. The fitness function is what will drive the entire population of solutions towards a globally best Figure 1, illustrates the basic steps in the canonical genetic algorithms.

Genetic algorithms are based on the model of evolution, in which a population evolves towards overall fitness, even though individuals perish. Evolution dictates that superior individuals have a better chance of reproducing than inferior individuals, and thus are more likely to pass their superior traits on to the next generation. This "survival of the fittest" criterion was first converted to an optimization algorithm by Holland in 1975 [9], and is today a major optimization technique for complex, nonlinear problems. In a genetic algorithm, each individual of a population is one possible solution to an optimization problem, encoded as a binary string called a chromosome. A group of these individuals will be generated, and will compete for the right to reproduce or even be carried over into the next generation of the population.

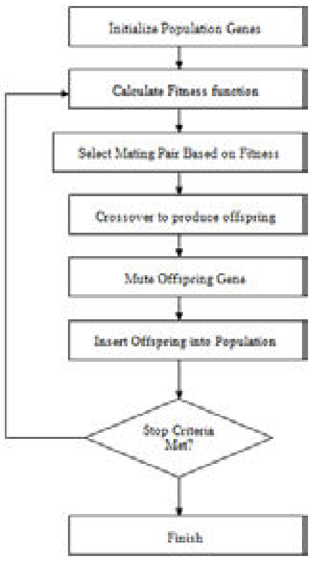
Initialize Population Genes
Calculate Fitness function
Select Mating Pair Based on Fitness
Crossover to produce offspring
Mute Offspring Gene
Insert Offspring into Population
Stop Criteria Met?
Finish

Figure 1: Genetic algorithm

IV. Parameters of genetic algorithm

The parameters of Genetic Algorithm are as follows

- o Population Size.
- o Number of Generation.
- o Crossover Rate.
- o Mutation Rate.
- o Selection Method.
- o Reproduction Rate.

V. Mapping of genetic algorithm with software reliability
GA is a powerful machine learning technique and optimization techniques to estimate the parameters of well known reliably growth models.

- **Fitness function: (for Reliability estimation).**

Fitness = (∑i=1 to m (xi – x^i)2 Dim)1/2

Where xi is the real value, x^i is the estimated value, Di is the weight of each example, and m is the size of the dataset.

- **Regression Model**

The regression model is given by Sultan H. Aljahdali and Mohammed E. El-Telbany [7] as follows.

Time series analysis deals with the problems of identification of basic characteristic features of time series, as well as with discovering - from the observation data on which the time series is built - the internal time series structure to predict time series data values which help in deciding about the subsequent actions to be taken. One of most used times series models is the auto regression model.

The AR model can be described by the following equation:

$$y_j = \omega_0 + \sum_{i=1}^{n} \omega_i y_{j-i}$$

Where is the previous observed number of faults and (i =1, 2,….,n). The value of n is referred to as the "order" of the model, and (i =1,2,…,n) are the model parameter.

VI. Steps of Genetic Algorithm

- **Steps of Genetic Algorithm**

1. Initialize population gene.
2. Calculate fitness value.
3. Select mating pair based on fitness.
4. Crossover to produce offspring.
5. Mutate offspring gene.
6. Insert offspring in to population.
7. Stop criteria met? If no go to step 2 else step 8.
8. Finish.

VII. Existing techniques
In component based software system, if a system consists of n components with reliabilities denoted by R1,…, Rn respectively, the reliability of system is an execution path, 1, 3, 2, 3, 2, 3, 4, 3, n, is given by Rs. Thus, the objective here is to estimate the reliability of a system by averaging over all path reliabilities. For this propose system consider the architecture of software as shown in Figure 2. Assume that, the application consists of n components, and has a single initial state denoted by 1, and a single absorbing or exit state denoted by n.

The expected reliability of system is defined by the following equation:

$$\mathrm{E[Rs]} = \mathrm{E}[\textstyle\prod_{i=1}^{n} \mathrm{Ri}^{\mathrm{m1,i}}$$

Where: E[Rs] is the estimated reliability of the system.

Rim1,i is the reliability of individual component.

Thus to obtain the expected reliability of the application, we need to obtain E[Rim1,i] which is the expected reliability of component i for a single execution. From the Taylor series expression of the function of a random variable [8] we have:

$$\mathrm{E}[\mathrm{R}_i^{m1,i}] = \mathrm{Ri}^{\mathrm{E[m1,i]}} + \frac{1}{2}\left(\mathrm{Ri}^{\mathrm{E[m1,i]}}\right) + (\log \mathrm{Ri})(\log \mathrm{Ri}) + \mathrm{Var[m1,i]}$$

Where: Var[m1,i] is the variance of individual component.

Since the number of visits to the absorbing state n is always 1, E[m1,n] = 1 and Var[m1,n] = 0 and hence E[Rnm1,n] = Rn. Equation (3) can be written as:

$$\mathrm{E[Rs]} = [\textstyle\prod_{i=1\,\mathrm{to}\,n-1} Ri^{E[m1,i]} + \frac{1}{2}\left(Ri^{E[m1,i]}\right) + (\log Ri)(\log Ri) + \mathrm{Var[m1,i]}]\, R_n$$

The result is shown in Table 2 according to equation (4) and (5)

p1,2 = 0.60	p1,3 = 0.20	p1,4 = 0.20	
p2,3 = 0.70	p2,5 = 0.30		

p3,5 = 1.00			
p4,5 = 0.40	p4,6 = 0.60		
p5,7 = 0.40	p5,8 = 0.60		
p6,3 = 0.30	p6,7 = 0.30	p6,8 = 0.10	p6,9 = 0.30
p7,2 = 0.50	p7,9 = 0.50		
p8,4 = 0.25	p8,10 = 0.75		
p9,8 = 0.10	p9,10 = 0.90		

Table 1: Transaction probability of components

The system reliability is 0.8267 using equation (3).

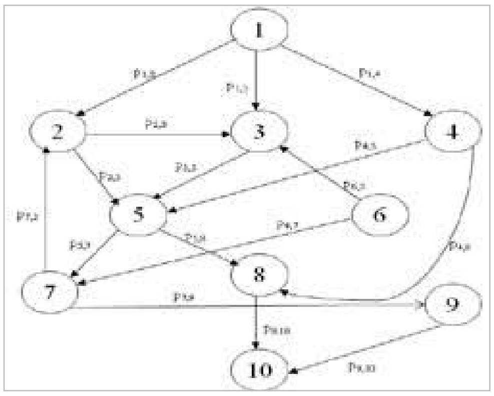

Figure 2: Architecture of Component Base Software

Component No.	Assume Reliability (Ra)	Mean (m)	Variance (v)	Estimated Reliability (Rs)
1	0.9990	1.0000	0.0000	0.9990
2	0.9800	0.9077	0.6444	0.9819
3	0.9900	0.9107	0.5499	0.9909
4	0.9700	0.4184	0.3928	0.9874
5	0.9500	1.3504	0.7185	0.9337
6	0.9950	0.2510	0.2319	0.9987
7	0.9850	0.6155	0.6261	0.9908
8	0.9500	0.8737	0.4255	0.9564
9	0.9750	0.3831	0.2462	0.9904
10	0.9850	1.0000	0.0000	0.985

Table 2: Estimated reliability using existing technique

VIII. Proposed Methodology

In this section, I propose a Genetic Algorithm based approach for estimating reliability for component based systems. It is often impossible to estimate software quality attributes directly. For example, attributes (say, reliability, etc.) are affected by many different factors, and there is no straightforward method to measure them. To estimate reliability of CBS, one needs to establish a relationship of the factors with reusability to achieve the desired goal. Following parameter has been identified, which will influence reusability of CBS:

- Assume Reliability (Ra) of component: We can assume the reliability of component according to its transaction probability from one component to other component.
- Mean (m) value of component: Denotes the number of visits to state j starting from state i before the process absorbed. Changing in mean value of component may affect the reliability of component. We can calculate the mean value using DTMC [8] method.
- Variance (v) value of component: Denotes the number of visits to state j starting from state i before the process absorbed. Changing in mean value of component may affect the reliability of component. We can calculate the variance value using DTMC [8] method.

We assume that the architecture of the application modeled using a Desecrate Time Markov Chain (DTMC) and the time spent by the application in each component per visit is a random variable with known mean and variance. We also assume that the reliability of each component per visit is known. We assume that the application consists of n components, and has a single initial stage denoted by 1, and a single absorbing state or exit state denoted by n. We also assume that the components fail independently of each other as well as successive executions.

IX. Conclusion

We can estimate the reliability of individual component of the system and according to this individual component reliability we can estimate the reliability of the whole system. We can estimate overall reliability of system considering the contribution of a component's reliability depending upon its usage time and the path propagation probability for possible paths of execution. By optimizing the assumed reliability of individual component and according to this we can optimize the reliability of individual component and whole system.

REFERENCES

[1] M. Sparling: "Lessons Learned through Six Years of Component Based Development", Communications of the ACM, 2003. | [2] Musa, John. Software Reliability Engineering, New York,NY, McGraw-Hill, 1998. | [3] J.D Musa (1987), Software Reliability measurement, prediction, application McGRAW-HILL International Edition. ISBN 0-07-100208-1 | [4] Michael R. Lyu (May 2005) Handbook of Software Reliability Engineering: Introduction. IEEE Computer Society Press and McGraw-Hill Book Company | [5] Alan wood (September 1996) TANDEM Software Reliability Growth Models. Technical report 96.1, part no 130056. | [6] Jung-Hua Lo, Chin-Yu Huang, Sy-Yen Kuo, and Michael R. Lyu (2003), Sensitivity Analysis of Software Reliability for Component-Based Software Applications. 27th Annual International Computer Software and Applications Conference, ISBN: 0-7695-2020-0. | [7] Aljahdali and Mohammed E. El-Telbany (JUNE 2008, Genetic Algorithms for Optimizing Ensemble of Models in Software Reliability Prediction", ICGST-AIML Journal, pp 5-13, Volume 8. | [8] Swapna S. Gokhale. Kishor S. Trivedi (2002), Reliability Prediction and Sensitivity Analysis Based on Software Architecture, IEEE, Software reliability, pp 64-75, ISBN: 0-7695-1763-3. | [9] Holland J., Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan, 1975. | [10] S. Rajasekaran, "Neural Networks Fuzzy Logic and Genetic Algorithms Synthesis and Application," PHI, ISBN 81-203-2186-3. 2003.