A Software Reliability Estimation Tool Using Artificial Immune Recognition System

Prof.S.Chitra M. E. Dr. M.Rajaram

***Abstract*-Software is an integral part of many critical and non-critical applications and virtually any industries dependent on computers for their basic functioning. As computer Software permeates our modern society and will continue to do so at an increasing pace in the future, the assurance of its quality becomes an issue of critical concern. Techniques to measure and ensure reliability of hardware have seen rapid advances, leaving Software as the bottleneck in achieving overall system reliability. Its evaluation includes two types of activities namely reliability estimation and defect prediction. Predicting fault-prone modules for software development project enable the companies to reach high reliable systems and minimize necessary budget, personnel and resource to be allocated to achieve their goal. This can be achieved by identifying the fault-prone modules in the software development process in the first phase. In the next phase recognize the test results & adopt suitable reliability model by applying AIRS[Artificial Immune Recognition System].**

Index terms-**AIRS, Bayesian model, Reliability models, Software lifecycle, Software reliability**

I.INTRODUCTION

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. Its evaluation includes two types of activities namely Reliability estimation and Defect prediction.

"The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component."

Software testing is often used in association with the terms verification and validation. Verification is the checking or testing of items, including software, for confirmation and consistency with an associated specification. Software testing is just a kind of verification, which also uses the techniques such as reviews, analysis, inspections and walkthroughs.

Validation is the process of checking that what has been specified is what the user actually wanted.

Validation: are we doing the right job?

Verification: are we doing the job right?

II.OBJECTIVES

The objective of the paper is to help engineers, managers and users make more precise decisions to make everyone more concretely aware of software reliability by focusing attention on it and to have high quality reliable system to reduce the overhead after delivery. Economics of the development of a system, delivery and maintenance can be scaled out. The life span of the system can be estimated and make the users to be aware of reliable system.

III.METHODOLOGY

There are two main methodologies. One is the testing methodology in which the errors are spotted and the failure rate is calculated. These values are used as input for next stage that is the reliability estimation.

There are many probabilistic and statistical approaches for modeling software reliability. Software reliability estimation is used for various purposes during development, to make the right decision. And after the software has been taken into use reliability estimation will provide idea on the basis of maintenance recommendations. Further improvement is made on the basis of the recommendation to discontinue the use of the software.

IV.THE ARTIFICIAL IMMUNE RECOGNITION SYSTEM (AIRS)

The recognition and learning capabilities of the natural immune system have been an inspiration for researchers developing algorithms for a wide range of applications. This section introduces some basic immune system concepts and provides the history and background behind the AIRS algorithm for classification.

NATURAL IMMUNE SYSTEMS

The natural immune system offers two lines of defense, the innate and adaptive immune system. The innate immune system consists of cells that can neutralize a predefined set of attackers, or ‘antigens’, without requiring previous exposure to them. The antigen can be an intruder or part of cells or molecules of the organism itself. In addition, higher animals like vertebrates possess an adaptive immune system that can learn to recognize, eliminate and remember specific new antigens. This is accomplished by a form of natural selection.

The bone marrow and thymus continuously produce lymphocytes and each of these cells can counteract a specific type of antigen. Now if for example a B-cell lymphocyte encounters an antigen it codes for, it will produce antibody molecules that neutralize the antigen and in addition a large number of cloned B-cells are produced that code for the same antigen (‘cloned expansion’ or ‘colonel selection’). The immediate reaction of the innate and adaptive immune system cells is called the primary immune response. A selection of the activated lymphocytes is turned into sleeper memory cells that can be activated again if a new intrusion occurs of the same antigen, resulting in a quicker response. This is called the secondary immune response.

ARTIFICIAL IMMUNE SYSTEMS

Natural immune systems have inspired researchers to develop algorithms that exhibit adaptivity, associative memory, self – non-self discrimination and other aspects of **A Comprehensive Benchmark of the Artificial Immune Recognition** System (AIRS) immune systems. These artificial immune system algorithms (also known as immuno computing algorithms) have been applied to a wide range of problems such as biological modeling, computer network security & virus detection, robot navigation, job shop scheduling, clustering and classification.

The Artificial Immune System algorithm (AIRS) can be applied to classification problems, which is a very common real world data-mining task. Most other artificial immune system research concerns unsupervised learning and clustering. The only other attempt to use immune systems for supervised learning is the work of Carter .The AIRS design refers to many immune system metaphors including resource competition, clonal selection, affinity maturation, memory cell retention, and so on. AIRS builds on the concept of resource limited clustering.

According to the introductory paper, AIRS seems to perform well on various classification and machine learning problems. Watkins claimed, "The performance of AIRS is comparable, and in some cases superior, to the performance of other highly-regarded supervised learning techniques for these benchmarks". Later on, Goodman, Boggess and Watkins investigated the "source of power for AIRS" and its performance on multiple-class problems. They claim "AIRS is competitive with the top five to eight classifiers out of 10-30 best classifiers on those problems", "it was surprisingly successful as a general purpose classifier" and it "performed consistently strong across large scope of classification problems".

AIRS: THE ALGORITHM

In AIRS, there are two different populations, the Artificial Recognition Balls (ARBs) and the memory cells. If a training antigen is presented, ARBs (lymphocytes) matching the antigen are activated and awarded more resources. Through this process of stimulation, mutation and selection a candidate memory cell is selected which is inserted to the memory cell pool if it contributes enough information.

This process is repeated for all training instances and finally classification takes place by performing a nearest neighbor search on the memory cell population. To describe the AIRS algorithm in detail, let us assume we have a training data set *X* containing *n* labeled instances, *d i i i ag x t* R˘Z with *xi* an input with *d* attributes and *ti* a one dimensional target class (*i=1,2…n*). The algorithm goes through the following steps

1. Initialization

First all the data items will be normalized so that the affinity of every two training instances agi and agj is in the range [0,1]. In AIRS, the affinity is usually represented by Euclidean distance over the attributes. We assume the set MC as the memory cell pool containing m memory cells: MC={mc1,mc2,…,mcm}, and set

AB as the ARB4 Lingjun Meng, Peter van der Putten, Haiyang Wang population containing r ARBs: AB={ab1,ab2,…,abr}, with { , } mc mcj j j mc x t □,(j=1,2,…,m); { , } ab ab k k k ab x t □, (k=1,2,…,r). Then the memory cells pool MC and the ARB populations AB are seeded by randomly adding training instances.

2. Memory cell identification and ARB generation

From now on, antigens (training instances) will be presented to the algorithm one by one. If an antigen agi ={xi,ti} is presented to the system, the algorithm will identify a memory cell { , } mc match mc x t □which has the same class label (mc match t =ti) and lowest distance to agi. If there is no mcmatch available at this moment, just let agi act as the mcmatch. This mcmatch will then be cloned to produce new mc clones. First the attributes of mcmatch will be mutated with a certain probability. If any mutations occurred for this particular clone, the class label will be mutated as well with the same probability

3. Competition for Resources and Development of a Candidate Memory Cell

At this moment, there are a set of ARBs including mcmatch, mutations from mcmatch, and others from previous training. AIRS mutates these memory cell clones to generate new ARBs. The number of ARBs allowed to produce is calculated by the product of the hyper clonal rate, clonal rate (both default 10), and the stimulation level (1-distance to agi). The newly generated ARBs will be combined with the existing ARBs. AIRS then employs a mechanism of survival of the fittest individuals within the ARB population. First, each ARB will be examined with respect to its stimulation level when presented to the antigen.

In AIRS, cells with high stimulation responses that are of the same class as the antigen and cells with low stimulation response that are not of the same class as the antigen are rewarded most and allocated with more resources. The losers in competing for resources will be removed from the system. Then the ARB population consists of only those ARBs that are most stimulated and are capable in competing for resources. Then the stop criterion is evaluated.

The stop criterion is reached if the average stimulation value of every class subset of AB is not less than the stimulation threshold (default 0.8).

Then the candidate memory cell mccandidate is chosen which is the most stimulated ARB of the same class as the training antigen agi. Regardless whether the stop criterion was met the algorithm proceeds by allowing the ARBs the opportunity to proliferate with more mutated offspring.

This mutation process is similar to the mutation of phase 2, with a small exception: the amount of offspring than can be to produced is calculated by the product of stimulation level and the clonal rate only. If the evaluation criterion was not met in the last test, the process will start again with the stimulation activation and resource allocation step. Otherwise the algorithm will stop.

3. Memory Cell Introduction

Now if mccandidate is more stimulated by the antigen than mcmatch, it will be added into the memory cell pool. In

addition, if the affinity value between and mccandidate and mcmatch is also less than the product of the affinity threshold (average affinity between all training items) and the affinity threshold scalar (a parameter used to provide a cut A Comprehensive Benchmark of the Artificial Immune Recognition System (AIRS) 5 off value, default 0.8), which means mccandidate is very similar to mcmatch, mccandidate will replace mcmatch in the set of memory cells.

By this mechanism, better classifying memory cells can replace existing memory cells so that the data reduction capabilities of the algorithm are improved. Training is completed now for this training instance agi. and the process is repeated from step 2 for the next instance.

4. Classification

With the training completed, the evolved memory cell population MC={mc1, mc2…,mcm} (m<n) will be used for classification using k-nearest neighbor. The classification for a test instance will be determined by the majority vote of the k most stimulated memory cells.

V.BAYESIAN MODEL

It is rather easy to incorporate evidence from many sources, e.g. experts, tests, Operational data etc.Bayesian models also work when there are no positive instances (e.g. when no Failures have been observed).

Bayesian approaches have been proposed. The advantage of Bayesian models is that various important but non-measurable factors, such as software complexity, architecture, quality of verification and validation activities, and test coverage are easily incorporated in the model.

THE PROCESS OF BAYESIAN RELIABILITY ASSESSMENT

To state the problem in Bayesian terms, Here the event $b\ e\ f\ t\ ,\ t\ ,m$ means the prediction that the software will fail $f\ m$ times in the time period starting from time $b\ t$ and lasting until time $e\ t$; if the different failure modes would be counted separately, a separate index term would be inserted to the event. Note that the reliability of the system is a special case of this formulation:

$$p(t_b, t_e, m_f \mid data) = \frac{p(data \mid t_b, t_e, m_f)\, p(t_b, t_e, m_f)}{p(data)} \quad (1)$$

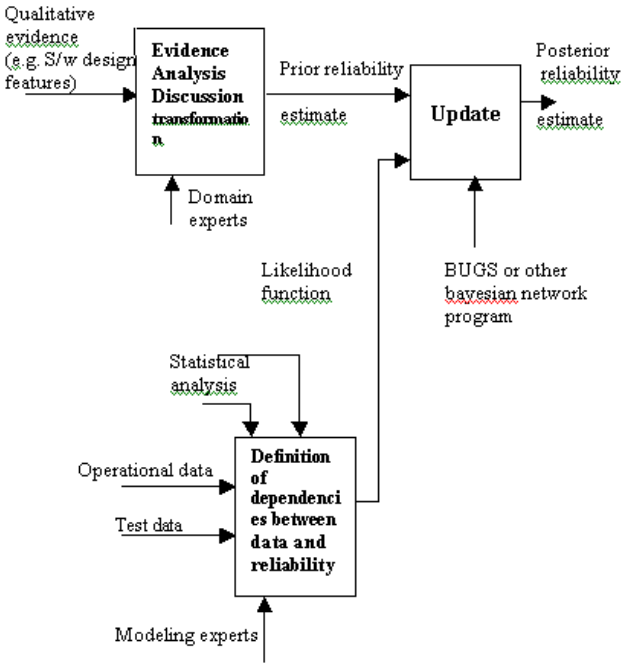

Fig 1: The process of bayesian reliability assessment

It is the probability that = 0 $f\ m$ .The event *data* means that the operational data is what it is; if the data would be in time series form, it might indicate that software failed $d\ m$ times in the time period starting at time $d\ b\ t$, and lasting until time $d\ e\ t$, ; there might of course be data for several time periods. The data might alternatively consist of failure events with timing information. The prior probability (, ,) $b\ e\ f\ p\ t\ t\ m$ is, as stated above, arrived at through expert judgment. The prior probability for the data, $p(data)$, can be thought of as a scaling factor, for discounting exceptional circumstances. This report mainly focuses on finding an estimate for the likelihood function (| , ,) $b\ e\ f\ p\ data\ t\ t\ m$. In what follows, the information about $b\ e\ f\ t$, t ,m is usually embedded in models: we try to find the likelihood of the data given that we have some model with which to predict the number of failures.

VI.SCOPE

Hardware and software reliability engineering have many concepts with unique terminology and many mathematical and statistical expressions. Basically, the approach is to apply mathematics and statistics to model past failure data to predict future behavior of a component or system. Major statistical distributions used in hardware reliability modeling include the exponential, gamma, Weibull, binomial, Poisson, normal, lognormal, Bayes, and Markov distributions.

To use these distributions, data collected from failures of systems need to be fitted with techniques like maximum likelihood or least squares estimates. The appropriateness of the models selected need to be verified by using statistical methods like Chi-squared or goodness-of-fit. Because mechanical and electrical systems tend to deteriorate over time, these reliability distributions depend on time as the variable, usually calendar time. This project is one of its kind as it combines both testing and reliability estimation techniques.

It is important to document the times and nature of bug occurrences, and their correction times, throughout the design, implementation and the formal testing phases. Plays central role in the planning and control of software development projects.

- Reliability growth model
- Unified approach

VII. UNIQUE FEATURES

- It is not a statistical method of estimation.
- It is a reliability estimation growth model, which does not consider the system as a complete entity but it estimates the reliability on each phase of the development.
- Easy to perform the reliability estimation and prediction rather than other models.
- It is reliability growth model, which gives the early alarm in the phases of development.

VIII.ALTERNATIVES

- Models that are taken into account are software architecture, software complexity, test coverage, conduct of verification and validation, and

structured expert opinion should be given priority.

- In applications requiring high dependability, software reliability models should be used only in conjunction with other methods of ensuring sufficient quality. Otherwise the amount of testing grows prohibitively large. These methods include, but are not limited to, formal methods, software inspections and reviews, static analysis of code, and systematic software testing.
- One should not rely on a single model but rather choose a set of models whose results are combined in one way or another.

IX. ILLUSTRATIONS

Table1: Test results with severity

Test Intvl	Number of Failures	Hours in a Test Interval	Severity
1	1.331250e+001	5.600000e+001	1
1	1.859375e+000	5.600000e+001	2
1	1.312500e+000	5.600000e+001	3
1	1.093750e-001	5.600000e+001	4
2	1.487500e+001	5.600000e+001	1
2	1.453125e+000	5.600000e+001	2
2	1.125000e+000	5.600000e+001	3
2	2.343750e-001	5.600000e+001	4
3	1.612500e+001	5.600000e+001	1
3	9.531250e-001	5.600000e+001	2
3	8.437500e-001	5.600000e+001	3
3	3.125000e-001	5.600000e+001	4
4	1.632813e+001	5.600000e+001	1
4	5.312500e-001	5.600000e+001	2
4	5.312500e-001	5.600000e+001	3
4	2.343750e-001	5.600000e+001	4
5	1.571875e+001	5.600000e+001	1
5	2.812500e-001	5.600000e+001	2
5	3.750000e-001	5.600000e+001	3
5	9.375000e-002	5.600000e+001	4
6	1.459375e+001	5.600000e+001	1
6	2.656250e-001	5.600000e+001	2
6	5.000000e-001	5.600000e+001	3
6	1.562500e-002	5.600000e+001	4
7	1.365625e+001	5.600000e+001	1
7	3.125000e-001	5.600000e+001	2
7	6.250000e-001	5.600000e+001	3
8	1.264063e+001	5.600000e+001	1
8	2.343750e-001	5.600000e+001	2
8	4.687500e-001	5.600000e+001	3
9	1.042188e+001	5.600000e+001	1
9	9.375000e-002	5.600000e+001	2

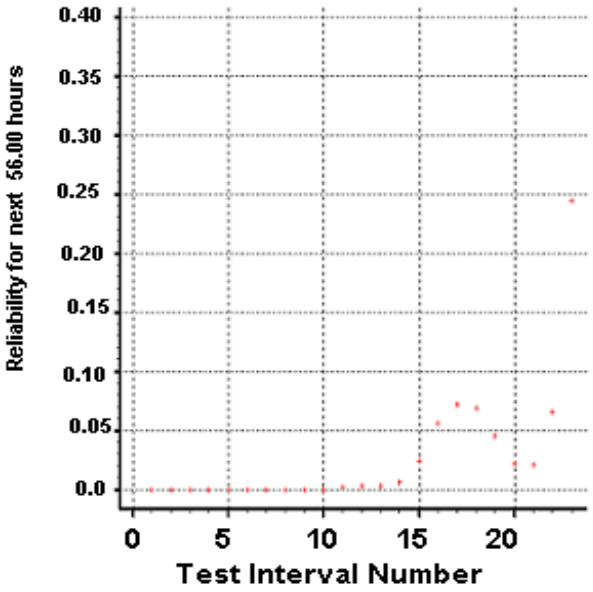

Fig 2: Reliability Plot

X. CONCLUSION

Software reliability is a key part in software quality. The study of software reliability can be categorized into three parts: modeling, measurement and improvement. Software reliability modeling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models exist which do not capture a necessary software characteristics. Assumptions and abstractions must be made to simplify the problem. There is no single model that is universal to all the situations. Software reliability measurement is naive. Development process, faults and failures found are all the factors related to software reliability.

This report is about the problem of statistically forecasting the number of software failures in a given time interval, given a history of previous failures (including the information that none have

occurred). An emphasis in the review has been put on the form of the likelihood function which represents the probability that the data is what it is, given that the model has a specific parametric form.

A multitude of models have been proposed , but each has its drawbacks, some being shared by most models. A common problem with the reviewed models is that none allow for non-existent failure data, i.e., a software usage history with a known duration of time in operational use with no detected failures. Another problem shared by the models is that they support only rather modest reliability claims. There is no solution to this problem in sight.

The AIRS algorithm to have pure data, that can be proposed to appropriate estimation model which provides the appropriate estimation on reliability for the system or module. Once the endurance of the system is fuzz , the developer can be alerted to take necessary steps to provide reliable software which will reduce the expenses after delivery of the system to end user.

XI. REFERENCES

[1.] Ebeling, Charles E., (1997), An Introduction to Reliability and Maintainability Engineering, McGraw-Hill Companies, Inc., Boston.

[2.] Denney, Richard (2005) Succeeding with Use Cases: Working Smart to Deliver Quality. Addison-Wesley Professional Publishing. ISBN . Discusses the use of software reliability engineering in use case driven software development

[3.] Kapur, K.C., and Lamberson, L.R., (1977), Reliability in Engineering Design, John Wiley & Sons, New York.

[4.] Siddhartha R. Dalal and Allen A. McIntosh, "When to Stop Testing for Large Software Systems with Changing Code", IEEE Transactions on Software Engineering

[5.] S. R. Dalal and C. L. Mallows, “When Should One Stop Testing Software”, Journal of the American Statistical Association

[6.] Willa Ehrlich, Bala Prasanna, John Stampfel, and Jar Wu, "Determining the Cost of a Stop-Test Decision”.

[7.] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software Users Guide,

[8.] Swapna S. Gokhale, Teebu Phillip, and Peter N. Marinos, “A Non-Homogeneous Markov Software Reliability Model with Imperfect Repair”

[9.] Swapna S. Gokhale, Peter N. Marinos, Michael R. Lyu, and Kishor S. Trivedi, “Effect of Repair Policies on Software Reliability”.

[10.] Chin-Yu Huang, Sy-Yen Kuo, Michael R. Lyu, and Jung-Hua Lo, “Quantitative Software Reliability Modeling from Testing to Operation”.

[11.] Daniel R. Jeski, “Estimating the Failure Rate of Evolving Software Systems” .

[12.] Ted Keller and Norman F. Schneidewind, “Successful Application of Software Reliability Engineering for the NASA Shuttle” Csenki, A.

[13.] Bayes predictive analysis of a fundamental software reliability model. IEEE Transactions on Reliability, Vol. R-

39, No. 2 (June 1990), 177.183 Cutello, V. and Nicosia, G. (2002).

[14.] Multiple learning using immune algorithms. In 4th International Conference on Recent Advances in Soft Computing (RASC-2002), pages 102–107,

[15.] Carter, J. H. The immune systems as a model for pattern recognition and classification.Journal of the American Medical Informatics Association 7(1), 28-41, 2000 Blanchard, Benjamin S. (1992),

[16.] Logistics Engineering and Management (Fourth Ed.), Prentice-Hall, Inc., Englewood Cliffs, New Jersey.