IJRST
International Journal of Research in Science & Technology
Website: www.ijrst.weebly.com (Volume 1, Issue 1, February 2014)

Neural Network Methodology for Embedded System Testing

P. Banu Priya[1], M. Vinusha Mani[2], D. Divya[3]

[1]*PG Scholar, Embedded Systems Technologies, Vel Tech Multi Tech Dr. RR Dr. SR Engineering College, Chennai, India, Priya.priyabe.banu@gmail.com*

[2]*PG Scholar, Embedded Systems Technologies, Vel Tech Multi Tech Dr. RR Dr.SR Engineering College, Chennai, India,vinelshaddai@gmail.com*

[3]*PG Scholar, Embedded Systems Technologies, Vel Tech Multi Tech Dr. RR Dr.SR Engineering College, Chennai, India,ddivya18@gmail.com*

***Abstract*—This paper describes testing framework that is capable of testing heterogeneous embedded systems. Here, we present a new concept of using an artificial neural network as an automated oracle for a tested software system. A neural network is trained by the back propagation algorithm on a set of test cases applied to the original version of the system. The network training is based on the "black-box" approach, since only inputs and outputs of the system are presented to the algorithm. A set of different experiments in the domain of testing of embedded system is presented. The trained network can be used as an artificial oracle for evaluating the correctness of the output produced by new and possibly faulty versions of the software. We present experimental results of using a two-layer neural network to detect faults within mutated code of a small credit approval application. This approach is capable to testing of host based and target based embedded systems.**

***Keywords*—Black-box, DID-Discovery Interface Device, HBT-Host based Embedded System Testing, neural network and TBT-Target based embedded system testing.**

I. Introduction

Testing is most commonly used method for determining quality of software. In the embedded world testing is a great challenge. In test plan sole characteristics of embedded systems must be reflected as they are application specific systems. It gives embedded systems testing exclusive and distinct flavor. There are two important classes of embedded systems, safety critical embedded systems and technical scientific algorithm based embedded systems. Host based embedded devices and target based embedded devices are sub classification for embedded systems. This paper brings together new approach and methodology to construct a frame work for heterogeneous and extremely complex embedded testing. Toward this end we have constructed a discovery interface device which provides classification and identification of embedded system as well as responsible for selection of testing methodology, this discovery interface device is the new concept in the embedded world. In the traditional testing approaches decision has to be made that whether to test hardware or software, but instead of finding heterogeneous testing techniques focus is on developing a mechanism which will assemble together all techniques of testing, This approach is capable to test host based and target based embedded system testing.

The main objective of testing is to determine how well an evaluated application conforms to its specification. Two common approaches to software testing are black-box and white-box testing. While the white-box approach uses the actual code of the tested program to perform its analysis, the black-box approach checks the program output against the input without taking into account its inner workings. Software testing is divided into three stages: generation of test data, application of the data to the software being tested, and evaluation of the results.

Traditionally, software testing was done manually by a human tester who chose the test cases and analyzed the results. However, due to the increase in the number and size of the programs being tested in present day, the burden of the human tester is increased, and alternative, automated software testing methods are needed. While automated methods appear to take over the role of the human tester, the issues of reliability and the capability of the software testing methods still need to be resolved. Thus, testing is an important aspect in the design of a software product. One of the limitations of the white-box testing approach is that it is not capable of analyzing certain faults, one of which is testing for missing code. The main problem associated with the black-box approach is to generate test cases that are more likely to detect faults. "Fault-based testing" is the term used to refer to methods that base the selection of test data on the detection of specific faults, and is a type of white-box approach as it uses the code of the tested program. Mutation analysis is a fault-based technique that generates mutant versions of the program that is being tested. A test set is applied to every mutant program and is evaluated to determine whether the test set is able to distinguish between the original and mutant versions.

Artificial neural networks (ANNs) have been used in the past to handle several aspects of software testing. Experiments have been conducted to evaluate the effectiveness of generating test cases capable of exposing faults, to use principle components analysis to find faults in a system, to compare the capabilities of neural networks to other fault-exposing techniques, and to find faults in failure data.

II. LITERATURE SURVEY

Embedded system testing is not a simple task, it varies application to application. Embedded systems can be tested manually as well as automatically. Some of the embedded system testing uses simulation based verification when new multi core application arises, so in order to improve the quality of software the test generation frame work is used static analysis, mutation testing and the model checking is used for the message passing programming libraries for distributed embedded system. The verification simulation testing model checking could be used in order to check the equivalency between the original program and the program with inserting fault depend on the result .The fault is detected by model checker used in order to generate the test cases. No time out were used for C bounded model checker, no scalability will explored in order to test. Unique test cases are used MCAPI which has no longer benchmark. [01]

Due to diverse architecture of software between the embedded systems, make it difficult to reuse, portability and dependability. Middleware is software between operating system and an application to solve any beginning problem some middleware consists of API module service manager and content manager module. API module is used with interface to communicate with lower and upper layer the application of touch screen for the embedded, middleware system is used in order to verify usability. Functionality and the integrated test of embedded middleware used communication between system to system, operating system to libraries ,libraries to application were the issues related to the diverse architecture embedded system such issues were solved somewhere in some extend.[02] Some embedded software testers use supporting tool for embedded software testing.

Embedded system requires hardware for testing purposes having very high development cost. Supporting tool can automatically generate test cases and test drivers, and supports coverage test and unit test which are based cross testing technology and multiple round mechanism results can be shown graphically by using constructed testing environment. The (ATEMES) Automatic testing environment for multi core embedded software is composed of four parts pre-processing module (PRPM), Host–side auto testing module (HSATM), Target-side auto-testing module (TSATM) and the last is post processing module. [6]

The construction and application of electronic port gate system, which is vivid application scenario of internet of vehicles, will benefits the entry and exit of vehicles for quick clearance guidance. It is also a great help for the monitoring of comprehensive import the efficiency of entry and exit port is a significant improvement on administrative work. [11]

Traditional PLC software about programming and monitoring all were designed in serial port debugging mode based on local. So the remote monitoring and controlling function couldn’t be realized in the mode. In the article the virtual serial port (VSP) technology was imported into the system and it turned the PLC serial port software from local control function to remote one. [12]

III. NEURAL NETWORKS

The most frequent example for biologically inspired computing is that of neural networks (NNs). A NN consists of interconnected neurons, each with a set of input and output connections. In principle, a neuron contains a simple add-and-compare mechanism that sums up the input signals and generates an output signal (i. e., the neuron “fires”) if a particular threshold has been exceeded. While the concept of such a neuron cell is very simple, a whole NN shows emergent properties such as learning and reasoning (for example, think of the abilities of the human brain, a NN with about 1010 neurons). NNs are extreme versatile. According to the theorem of Hecht-Nilsson [3], *any* given function can be expressed by a three-layer NN with an appropriate number of neurons. An impressive feature of a NN is the ability to *learn*, which enables such systems to adapt to changing conditions [4]. NNs support supervised and unsupervised learning.

In supervised learning, back-propagation NNs are used. During a training phase, the parameters of the NN are adapted until the system performs the desired function. The trained system is then able to perform the programmed function with a high robustness against errors.

An example for such an application in the embedded domain is given in [5], where an artificial NN is used to filter out errors from infrared distance sensors. Unsupervised learning algorithms try to extract common sets of features from the input data [4]. An example for an unsupervised learning artificial NN is Kohonen's self organizing map [6]. Unsupervised learning algorithms are used for automatic classification, modeling, and data compression systems.

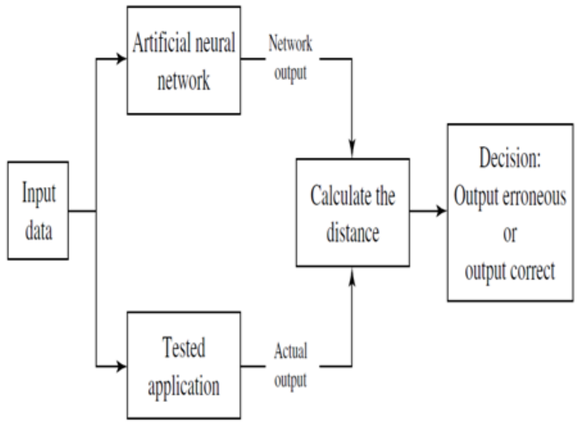

Figure1. Method overview

IV. PROPOSED METHODOLOGY

In this paper, we present a new application of neural networks as an automated "oracle" for a tested system. A multi-layer neural network is trained on the original software application by using randomly generated test data that conform to the specification. The neural network can be trained within a reasonable accuracy of the original program, though it may be unable to classify the test data 100 percent correctly. In effect, the trained neural network becomes a simulated model of the software application. When new versions of the original application are created and "regression testing" is required, the tested code is executed on the test data to yield outputs that are compared with those of the neural network. We assume here that the new versions do not change the existing functions, which means that the application is supposed to produce the same output for the same inputs. A comparison tool then makes the decision whether the output of the tested application is incorrect or correct based on the network activation functions. Figure 1 presents the overview of the proposed testing methodology.

TABLE I
OVERVIEW ON PRESENTED METHOD

Method	**Characteristics**	**Sample embedded applications**
Artificial Neural Networks	unsupervised or supervised learning, black-box data processing structure	Signal calibration, pattern recognition, classification.

Using an ANN-based model of the software, rather than running the original version of the program, may be advantageous for a variety of reasons. First, the original version may become unusable, due to a change in the hardware platform or the OS environment. Another usability problem may be associated with a third-party application having an expired license or other restrictions. Second, most inputs and outputs of the original application may be non-critical at a given stage of the testing process, and, thus, using a neural network for an automated modeling of the original application may save a significant amount of computer resources. Third, saving an exhaustive set of test cases with the outputs of the original version may be infeasible for real-world applications. Finally, the original version is never guaranteed to be fault-free, and comparing its output to the output of a new version may overlook the cases where both versions do not function properly. Neural networks provide an additional parameter associated with every output, the activation function, which, as we shown below, can be used to evaluate the reliability of the tested output.

V. DESCRIPTION OF THE TESTING METHODOLOGY

Our testing methodology can be viewed as a black-box approach. In the training, the input vector (test case) for each training example is generated randomly, subject to the specifications of the tested program. Each input vector is then provided as an input to the original version of the tested program, which subsequently generates a corresponding output vector. The input and output vectors of the program are used for training the neural network. The trained neural network can perform the function of an automated "oracle"12 for testing the subsequent versions of the program (a process known as "regression testing").

In the evaluation phase, each test case is provided as an input vector to a *new version* of the tested program and to the trained ANN. For each input vector, the comparison tool calculates the absolute distance between the winning ANN output and the corresponding value of the application output. All the outputs are numbers between zero and one. The computed distance is then placed into one of three intervals defined by two threshold values in the [0, 1] range. If the network prediction and the application output are identical, and the computed distance falls into the interval between 0.0 and the low threshold (the distance is very small), this means that the network prediction matches the program output and both outputs are likely to be correct. In this case, the comparison tool reports that the program output is correct. On the other hand, if the network prediction is different from the application output and the distance between outputs is very large (falls into the interval between the high threshold and 1.0), a program fault (incorrect output) is reported by the comparison tool. In both cases, the network output is likely to be correct and it is reliable enough to evaluate the correctness of the application output. However, if the distance is placed into the interval between the low threshold and high threshold, the network output is considered unreliable. In the last case, the comparison tool reports a program fault only if the network prediction is identical to the application output.

The comparison tool is employed as an independent method of comparing the results from the neural network and the results of the tested versions of the credit approval application. An objective automated approach is required to ensure that the results have not been affected by external factors. This in effect replaces the human tester, who may be biased by having prior knowledge of the original application.

TABLE III
EACH OUTPUT HAS A DEFINED CATEGORY

ANN output	Tested application output(correct)	Tested application output(wrong)
correct	1, True positive	2, True negative
wrong	4, False negative	3, False positive

The tool uses the output of a neural network and the output of the tested application. The distance between the outputs is taken as the absolute difference between the value of the winning node for each output and the corresponding value in the application. Since a sigmoid activation function is used to provide the network outputs, the activation value of the winning output nodes is a number between 0.0 and 1.0. The corresponding value of the application output is equal to 1.0 if the predicted and actual outputs are identical. Otherwise, it is equal to 0.0. Thus, the distance covers a range between 0.0 and 1.0, and we use this value to determine whether the faulty application has generated an invalid or correct result. Table II displays the four possible categories where each output can be placed.

Since the ANN is only an approximation of the actual system, some of its outputs may be incorrect. On the other hand, the tested application itself may produce errors, which is the main reason for the testing process. If the ANN output is correct while the output of the tested application is wrong, the evaluation of the comparison tool is classified as being a true negative or a category of 2, i.e., the determination that the output of the application is an actual error. Similarly, the remaining three classifications represent the other possibilities for the output categorization. Each output arising from the neural network and the tested program is evaluated in this fashion. Although we are mainly interested in finding the wrong outputs (categories 2 and 3), we also note that there is no visible difference when the network output is the same as the output of the tested program (categories 1 and 3). Categories 2 and 4 are also similar in that regard, as either the network output is correct or the tested program output is correct, with the former being more likely. The ANN is trained to simulate the original application; however it is not capable of classifying the original data 100% correctly due to the problem of error convergence discussed in Section 2. In the next section, we show that despite this seeming disadvantage, the percentage of incorrectly classified outputs is minor compared to the overall data set size. Thus, we are interested in the cases where the tested application output is wrong: categories 2 and 3, using the notation of Table II. When the outputs are compared with one another, they are either the same or different. Consequently, categories 1 and 3 have to be distinguished from one another by the comparison tool; a similar separation is required for categories 2 and 4. Thus, the need for calculating the distance is justified.

The two types of outputs are handled differently by the comparison tool, as there are four possible situations for a binary output and five for a continuous output. The analysis of a continuous output differs in one part with respect to a binary output. When both the ANN output and the faulty application are different, the ANN output may be correct, the faulty application output may be correct, or in the case of a continuous output, they both may be wrong, whereas only two situations are possible for the binary output (See Table III).Here, the values of both the low and the high threshold for each type of output were obtained experimentally to yield the best overall fault detection results.

TABLE IIIII
POSSIBILITIES OF FAULT TYPES

Type of output	Same	Different
Binary	• Both correct • Both wrong	• ANN correct • Faulty application correct
Continuous	• Both correct • Both wrong	• ANN correct • Faulty application correct • Both wrong

VI. DESCRIPTION OF THE PROCESS

The design of the experiment is separated into three parts, one for the sample application, one for the neural network, and the last for the comparison tool that calculates the distance between the two outputs (See Figure 1). The specification and algorithm for a credit card approval application are used to provide the necessary format of the input and output attributes and the placement of the injected faults.

The design of the neural network is also partially dependent on the input/output format of the application; however, the tuning knobs for training the neural network were manually determined by trial and error. Figures 2 and 3 present the flow of the process followed in the experiment. In the training phase, the input data for the training of the neural network and the tested program are generated randomly.

The input data are fed singularly to the tested program as an input vector that generates an output vector for each training example. Thus, the set of input vectors and output vectors are passed as input to the two-layer neural network, which is trained using the back propagation algorithm. In the evaluation phase, a mutated version of the tested program is created by making one change at a time to the original program. The testing data are then passed to both the neural network and the tested program. The outputs from both the tested program and the trained network are then processed by the comparison tool that makes a decision on whether the output of the tested program is erroneous or correct.

The process that the experiment follows begins with the generation of test cases. The input attributes are created using the specification of the program that is being tested, while the outputs are generated by executing the tested program. The data undergo a preprocessing procedure in which all continuous input attributes are normalized (the range is determined by finding the maximum and minimum values for each attribute), and the binary inputs and outputs are either assigned a value of 0 or 1. The continuous output is treated in a different manner, and the output of each example is placed into the correct interval specified by the range of possible values and the number of intervals used.

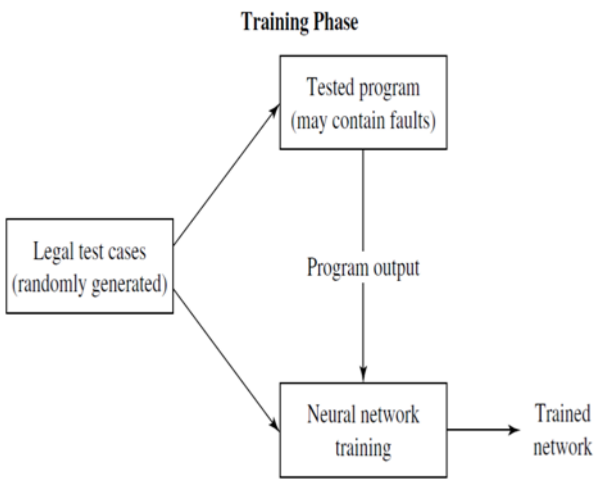
Training Phase
Tested program (may contain faults)
Legal test cases (randomly generated)
Program output
Neural network training
Trained network

Figure2. Overview of the Training Phase

The processed data are used as the data set for training the neural network. The network parameters are determined before the training algorithm begins. The training of the network includes presenting the entire data set for one epoch, and the number of epochs for training is also specified. The back propagation training algorithm concludes when the maximum number of epochs has been reached or the minimum error rate has been achieved. The network is then used as an "oracle" to predict the correct outputs for the subsequent regression tests. The comparison tool uses both the data from the oracle and the tested program to evaluate whether the output of the latter is wrong or correct. The back propagation neural network used in this experiment is capable of generalizing the training data; however the network cannot be applied to the raw data, as the attributes do not have uniform presentation. Some input attributes are not numeric, and in order for the neural network to be trained on the test data, the raw data have to be preprocessed, and, as the values and types differ for each attribute, it becomes necessary that the input data are normalized.

The values of a continuous attribute vary over a range, while there are only two possible values for a binary attribute (0 or 1). Thus for the network to be able to process the data uniformly, the continuous input attributes have to be normalized to a number between 0 and 1. The data are preprocessed by normalizing the values for all the input attributes according to the maximum and minimum possible values for each attribute. The output attributes were processed in a different manner. If the output attribute was binary, the two possible network outputs are 0 and 1. On the other hand, continuous outputs cannot be treated in the same way, since they can take an unlimited number of values. To overcome this problem, we have divided the range (found by using the maximum and minimum possible values) of the continuous output into ten equalized intervals and placed the output of each training example into the corresponding interval (a value between 0 and 9).

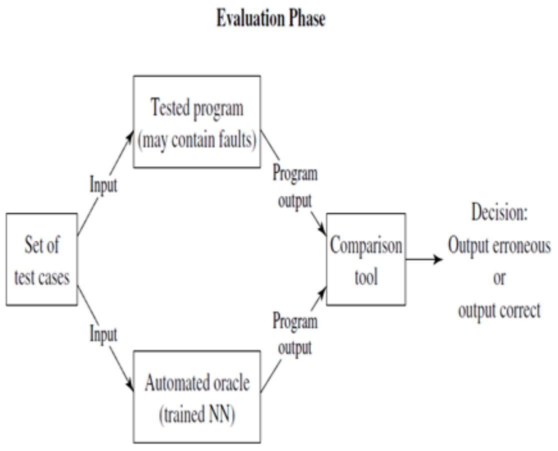
Evaluation Phase
Tested program (may contain faults)
Input
Program output
Set of test cases
Comparison tool
Decision: Output erroneous or output correct
Input
Program output
Automated oracle (trained NN)

Figure3. Overview of the Evaluation Phase

The architecture of the neural network is also dependent on the data. In this experiment, we used eight non-computational input units for the eight relevant input attributes (the first is not used, as it is a descriptor for the example), and twelve output computational units for the output attributes. The first two output units are used for the binary output. For the purposes of training, the unit with the higher output value is considered to be the "winner." Similarly, the remaining ten units are used for the continuous output. The initial synaptic weights of the neural network were obtained randomly and covered a range between –0.5 and 0.5. Experimenting with the neural network and the training data, we concluded that one hidden layer with twenty-four units was sufficient for the neural network to approximate the original application to within a reasonable accuracy. A learning rate of 0.5 was used, and the network required 1,500 epochs to produce a 0.2 percent misclassification rate on the binary output and 5.4 percent for the continuous output. Figure 4 shows the number of epochs versus the convergence of the error.

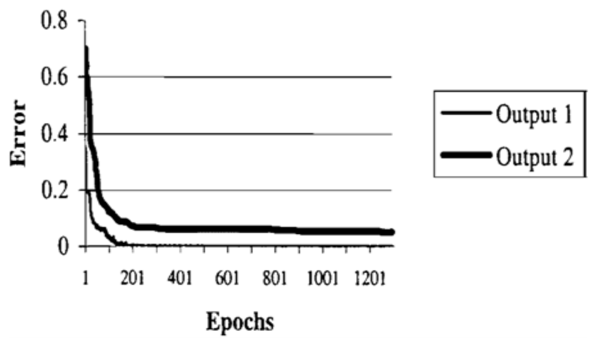

Figure4. Error Convergence

VII. EXPERIMENTAL RESULTS

The results for the error rate as a function of the threshold values are, the minimum error rate for the binary output is (low threshold=0.26, high threshold=0.86). The results include the injected fault number, the number of correct outputs and incorrect outputs as determined by the "oracle," and the percentages for the correct outputs classified as being incorrect and incorrect outputs classified as being correct. The percentages were obtained by comparing the classification of the "oracle" with that of the original version of the application. The original version is assumed to be fault-free, and is used as a control to evaluate the results of the comparison tool. The best thresholds were chosen to minimize the overall average of two error rates.

The binary output was affected by an injected fault only in five faulty versions out of 21 due to the structure of the application: the first output is only affected by the first two conditions in the program. The average error rates were obtained by varying the threshold values for the classification of the binary output. The lowest error rate obtained as a function of the thresholds. Comparing the overall average error rates to the minimum value is done; there is not much of a difference between them. However, in comparing the values for every set of threshold values, the individual error percentages vary for each fault.

The results for the continuous output are found. Due to the increased complexity involved in evaluating the

continuous output, there is a significant change in the capability of the neural network to distinguish between the correct and the faulty test cases: the minimum average error of 8.31 achieved for the binary output versus the minimum average error of 20.79 for the continuous output. An attempt to vary the threshold values also did not result in an evident change to the overall average percentage of error for the continuous output.

VIII. CONCLUSION

In this paper, we have used a neural network as an automated "oracle" for testing a real application, and applied mutation testing to generate faulty versions of the original program. We then used a comparison tool to evaluate the correctness of the obtained results based on the absolute difference between the two outputs. The neural network is shown to be a promising method of testing a software application provided that the training data have a good coverage of the input range.

The back propagation method of training the neural network is a relatively rigorous method capable of generalization, and one of its properties ensures that the network can be updated by learning new data. As the software that the network is trained to simulate is updated, so too can the trained neural network learn to classify the new data. Thus, the neural network is capable of learning new versions of evolving software.

The benefits and limitations of the approach presented in this paper need to be fully studied on additional software systems involving a larger number of inputs and outputs. However, as most of the methodology introduced in this paper has been developed from other known techniques in artificial intelligence, it can be used as a solid basis for future experimentation. One possible application can include generation of test cases that are more likely to cause faults. The heuristic used by the comparison tool may be modified by using more than two thresholds or an overlap of thresholds by fuzzification. The method can be further evaluated by introducing more types of faults into a tested application.

BIOGRAPHY

P. Banu Priya was born in Salem, Tamil Nadu on 03.02.1991. She received the B.E degree in Electronics and Communication Engineering from Alpha College of Engineering, Anna University, in 2012. She is studying Embedded Systems Technologies, Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engineering College of Anna University, Chennai for the M.E degree. Her interests are focused in the fields of Design of Embedded Systems, Networks.

M. Vinusha Mani was born in born in Nagercoil, Tamil Nadu on 13.02.1991. She received the B.E degree in Electronics and Communication Engineering from Kings Engineering College, Anna University, in 2012. She is studying Embedded Systems Technologies in Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engineering College of Anna University, Chennai for the M.E degree. Her interests are focused in the fields of Design of Embedded Testing.

D. Divya was born in born in Tiruttani, Tamil Nadu on 18.01.1991. She received the B.E degree in Electronics and Communication Engineering from St. Peter's College of Engineering and Technology, Anna University, in 2012. She is studying Embedded Systems Technologies in Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engineering College of Anna University, Chennai for the M.E degree. Her interests are focused in the fields of Signals and communication field.

References

[1] Alper Sen and etem Deniz, "Verification test for multicore communication API (MCAPI)", 2011-12th international workshop on microprocessor test and verification, 2011.

[2] Yang-hassin fan and jan-ouwu, "Middleware software for embedded system", 2012 26th international conference advance information networking and application workshop, 2012.

[3] Swapnili P. Karmore, "Universal Methodology for Embedded System Testing".

[4] C. Reichmann; P. Graf; K.D. Müller-Glaser: "GeneralStore – A CASETool Integration Platform Enabling Model Level Coupling of Heterogeneous Designs for Embedded Electronic Systems", 11th IEEE Conference on the Engineering of Computer Based Systems 2004, Brno,Czech Republic, May 2004.

[5] Justyna Zander-Nowicka1, Zhen Ru Dai1, "Model Driven Testing of Real-Time Embedded Systems- From Object Oriented towards Function Oriented Development", IFIP 17th Intern. Conf. on Testing Communicating Systems – Test Com 2005, Montreal, Canada, ISBN: 3- 540-26054-4, Mai 2005.

[6] Hua-ming Qian; Chun ZhengA, "Embedded Software Testing Process Model Computational Intelligence and Software engineering", 2009. CiSE, International Conference on 2009.

[7] Alexander Krupp, Wolfgang Muller Paderborn University, "A Systematic Approach to the Test of Combined HW/SW Systems",

IEE Conference on the testing and automation of embedded systems, Nov 2010.

[8] Huang Bo, Dong Hui, Wang Dafang and Zhao Guifan School, “Basic Concepts on AUTOSAR Development”, School of Automobile Engineering, Harbin Institute of technology at wihai, Shandon Province, China, Princeton University, IEE International Conference on Intelligent Computation Technology and Automation, 2010.

[9] Johannes Kloos, Tanvir Hussain, Robert Eschbach, “Risk based Testing of Safety-Critical Embedded Systems Driven by Fault Tree Analysis Fourth International Conference on Software Testing”.

[10] Mark Baker and Hong kong, “Design and Implementation of The Java Embedded Micro-kernel Software Architecture The Distributed Systems”, Group University of Portsmouth, PO1 2EG, UKMark.Baker@computer.org and Hong.Ong@port.ac.uk 2011.

[11] Wenxuan Yin_*yz*, Xiang Gao*yz*, Xiaojing Zhu*yz*, Deyuan Guo*x*, “An Efficient Shared Memory Based Virtual Communication System for Embedded SMP Cluster”, 2011 Sixth IEEE International Conference on Networking, Architecture, and Storage.

[12] Li Deng, Hai Jin, Song Wu, Xuanhua Shi, Jiangfu Zhou ,“Fast Saving and Restoring Virtual Machines with Page Compression”, International Conference on Cloud and Service Computing, 2011.

[13] Anderson C, von Mayrhauser A, Mraz R, on the use of neural networks to guide software testing activities. In: Proceedings of ITC’95, the International Test Conference; October 21–26, 1995.

[14] Choi J, Choi B. Test agent system design. In: 1999 IEEE International Fuzzy Systems Conference Proceedings; August 22–25, 1999.

[15] DeMillo RA, Offutt AJ, Constraint-based automatic test data generation. IEEE Transactions on Software Engineering 1991; SE-17(9):900–910.

[16] Kirkland LV,Wright RG, Using neural networks to solve testing problems. IEEE Aerospace and Electronics Systems Magazine 1997; 12(8):36–40.

[17] Mitchell T. Machine Learning, McGraw-Hill; 1997.

[18] Sherer SA, Software fault prediction. Journal of Systems and Software 1995; 29(2):97–105.

[19] Voas JM, McGraw G. Software Fault Injection; 1998.