# Quality Measurement Challenges for Artificial Intelligence Software

Zafar Ali
*Ms150200227, MSCS Student*
*Virtual University of Pakistan, Lahore, Pakistan*

***Abstract***— *In this paper, various metrics of software measurements are explained with examples. The standards developed for software measurement is described. The goals of these standards are also explained. The challenges of quality measurement for AI software are discussed here. AI software is different from the common software in two ways: it generally solves different kind of problems and it solves the problems in a very different way.*

*AI software generally comes with a vague or a quickly changing requirement list. Quality measurement becomes meaningless as how far the objective has been achieved cannot be decided when the objective itself is not clear. Rapid prototyping and freezing of requirements is a short term measure for this issue. The other option is to identify the modules that can be implemented by conventional software and to integrate the measurement plan of all these modules with suitable modifications.*

*Another issue is; AI software is often implemented in a heuristic manner that gives a poor readability and measurement problems. Further a conflict between the prejudiced human being as a tester and tolerant expert system may take place. Ultimate goal of development of expert system is to provide solutions to problems impossible for normal human beings. Future research trends, both long and short term, are briefed here.*

***Keywords***—*artificial-intelligence, software, quality, measurement.*

## I. INTRODUCTION

Software quality measurement encompasses varieties of techniques [1]. As new types of software are emerging newer measurement techniques might be essential. Initial attempt was to adopt the well established techniques from hardware area. Measurement techniques are mainly based on measurement of defects or faults. In software area this is commonly called as bugs. Due to defects or bugs failures may take place. To estimate defects failure rates can be studied. Some of the metrics related to failure are:

Failure Probability F(t): It is the probability that the software will fail prior to time t.

Reliability R(t): It is the probability that the software will work satisfactorily till time t. Clearly, $R(t)=1 – F(t)$.

These metrics are extremely useful for hardware components and many other metrics are defined and analyzed. But estimation of these metrics for software does not seem to be a practical idea. Probability is estimated from large number of outcomes. For hardware components a batch produces a huge number of components. Estimation of various probabilities is quite feasible. Further the observed time $t_{ob}$ can be scaled down from the predicted time t by application of temperature, bias, electric field or magnetic field stress.

For software there is no established concept of stressing. The tester may need to wait for indefinite time to get the software failure. There might be no relation between software bugs and time t. Bugs usually show up for a particular input set. Whenever a bug is detected it should be removed. A bug removed at a later stage always proves to be very expensive for the software development. A bug in the requirement analysis phase proves costliest if detected by customer while using the software. This is in contradiction to hardware component testing situation where defects can be removed only in the next batch of manufacturing. For software, probability estimations are inconvenient due to lack of large number of outcomes. As soon as a bug is detected it is removed. After the bug removal this becomes a different product and the test results from different products cannot be clubbed.

For software, thorough testing and immediate bug removal is the standard strategy. Testing can be classified into two categories: static and dynamic. In static testing thorough scrutiny is done without the actual execution. In dynamic testing some type of execution is necessary. There are various types of dynamic testing. Random testing is done for randomly generated inputs within the specified range. There is no guarantee that software will be bug free after random testing. Suppose software involves lots of divisions and division by zero is not taken care. Random inputs generated in a very large range of integers may not include a zero. The particular bug remains undetected. An exhaustive test with all input sets is very time consuming. Further the sequence of input application might be important for bug detection. This is true for some Artificial Intelligence (AI) software.

Testing should be of regression type. In this type of testing after removal of a bug all the previously done tests are repeated. This is needed as a bug removal may or may not be successful and the removal process may introduce additional bugs.

There are different views about what type of software can be put in the AI category. W. J. Rapaport made quite an exhaustive compilation of definitions of AI software [2]. It is surprising to note that most of the definitions indicate that, this is new and emerging type of software with not enough quality measurement metrics, control and assurance. One such definition is: “AI is a collective name for problems which we do not yet know how to solve properly by computer. Once we do know how to solve them, they are no longer AI”. If we accept this definition quality measurement is definitely a challenge or might be impossible in some cases. However, to focus on a particular type of software we follow the first definition listed in this document: “The goal of work in artificial intelligence is to build machines that perform tasks normally requiring human intelligence”. We follow the dictionary meaning of intelligence as: “The ability to acquire and apply knowledge and skills”.

Newborns are usually very intelligent. Though they do not have any knowledge or skill they have the capability to acquire and apply these. AI software just after implementation can be compared to a newborn’s brain.

In the learning phase it may learn face recognition and becomes very knowledgeable. In the testing phase it applies its knowledge to recognize a face and take a suitable decision e.g. opening a door to a authenticated person. Prior to learning it did not have any knowledge but had the ability to learn and become knowledgeable. So it was intelligent.

## II. RELATED WORK

Software measurement metrics are directly taken from the hardware component measurement. But the two situations are different. N. Fenton et al. emphasized the need for causality modeling [3]. Though the bugs are responsible for failure there is no relation between the detected bugs and subsequent failures. A detected bug is immediately removed. It is the residual bugs that create the subsequent failures. There is no relation between detected and the residual bugs. This can be seen in the measured data in fig. 1.

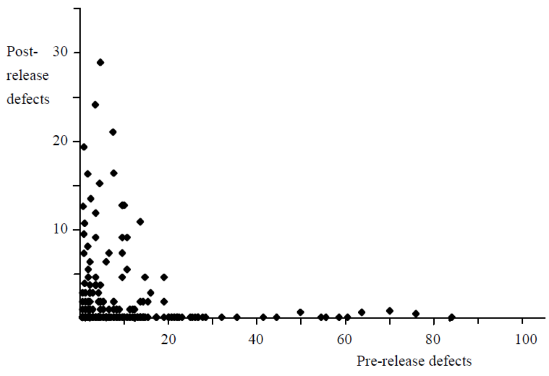
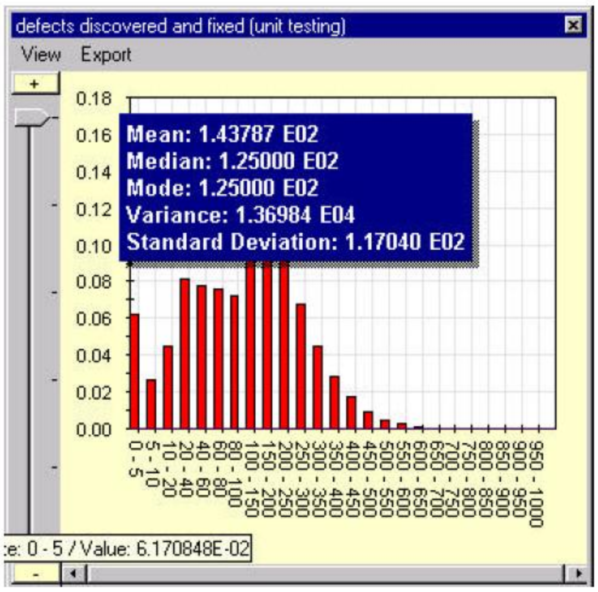


**Figure 1: Measured data for pre- and post-release defects for different modules [3].**

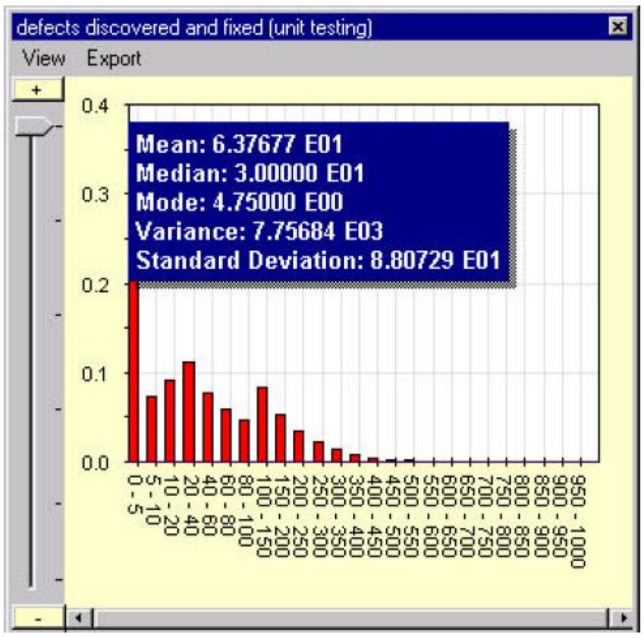
Authors developed a tool named AID based on Bayesian network that can predict residual defects distribution. Two modules were used one very simple and the other very complex. From figures 2 and 3 it can be observed that, detected defect distribution is more in the simple module as compared to complex module.



Mean: 1.43787 E02
Median: 1.25000 E02
Mode: 1.25000 E02
Variance: 1.36984 E04
Standard Deviation: 1.17040 E02

e: 0 - 5 / Value: 6.170848E-02

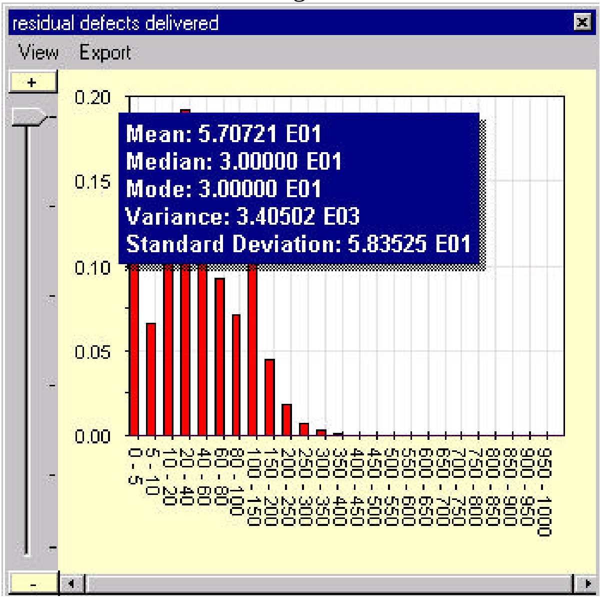
**Figure 2: Distribution of defects detected in unit tests for a very simple module [3].**



Mean: 6.37677 E01
Median: 3.00000 E01
Mode: 4.75000 E00
Variance: 7.75684 E03
Standard Deviation: 8.80729 E01

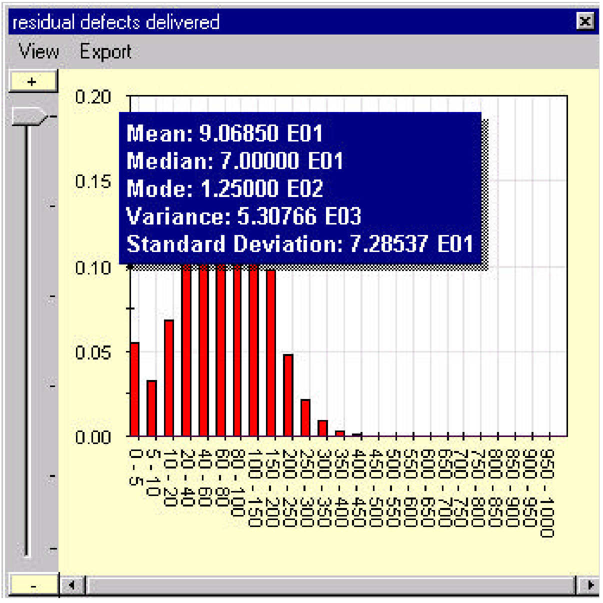
**Figure 3: Distribution of defects detected in unit tests for a very complex module [3].**

More number of detected defects does not necessarily imply more defects in the software. Though number of detected defects is much higher in simple module is does not mean a simple module has more defects than a complex module. From our common sense we can say that, it is just the reverse. The complex module has less detected defects because such module can hide the defects. This becomes clear when residual defects are compared for the two modules from figures 4 and 5.



Mean: 5.70721 E01
Median: 3.00000 E01
Mode: 3.00000 E01
Variance: 3.40502 E03
Standard Deviation: 5.83525 E01

**Figure 4: Distribution for residual defects delivered for a very simple module [3].**



Mean: 9.06850 E01
Median: 7.00000 E01
Mode: 1.25000 E02
Variance: 5.30766 E03
Standard Deviation: 7.28537 E01

**Figure 5: Distribution for residual defects delivered for a very complex module [3].**

M. Khraiwesh introduced the concept of Goal Question Metrics (GQM) [4]. This is integrated in the existing Capability Maturity Model Integration (CMMI). This model is developed in Carnegie Melon University and is practiced world-wide. The model considers product as well as process level measurements. Fig. 1 illustrates two types of goals.

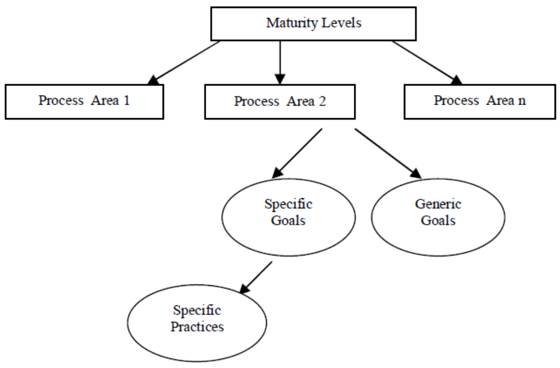


**Figure 6: Example of specific and generic goals [4].**

J. Mylopoulos et al. did a study on non-functional requirements based on goals [5]

L. Prechelt did an extensive study on quality measurement status of Artificial Neural Network (ANN) software [6]. The author collected papers on ANN in the years 1993 – 94 from 5 journals. Total 190 papers were studied. For ANN the quality is very much dependent on inputs. An algorithm should be evaluated experimentally for as many input sets as possible. It should be evaluated at-least for 2 input sets so that comparisons can be done between the two. Fig. 7 shows the % of reported algorithms vs. the no. of input sets.

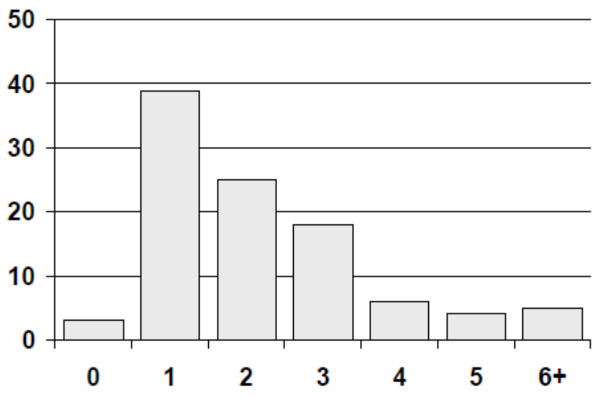


**Figure 7: % of reported algorithms vs. no. of input sets [6].**

Above figure shows that, only 33% of algorithms use 2 or more input sets. Often adequate input data of a particular set is not available or accessible. This problem is managed with artificial data inputs. These artificial data are generated from suitable models. Fig. 8 shows the % of reported algorithms vs. the no. of artificial input sets.

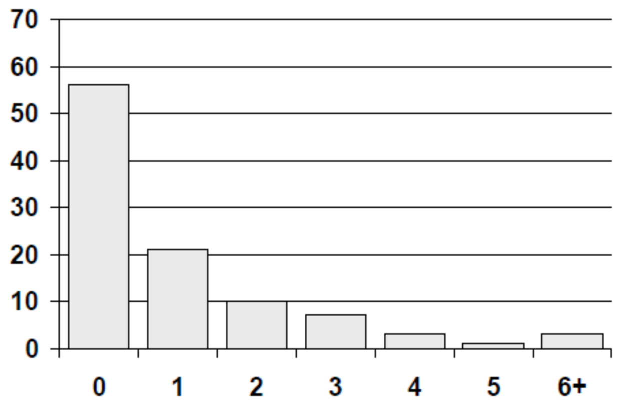


**Figure 8: % of reported algorithms vs. no. of artificial input sets [6].**

Above figure shows no. of tested algorithms for 2 or more input sets is further reduced. So non-availability of real data is not the issue. Fig. 9 shows the distribution for real data. Distribution remains the same as artificial data for input data-sets below 6.

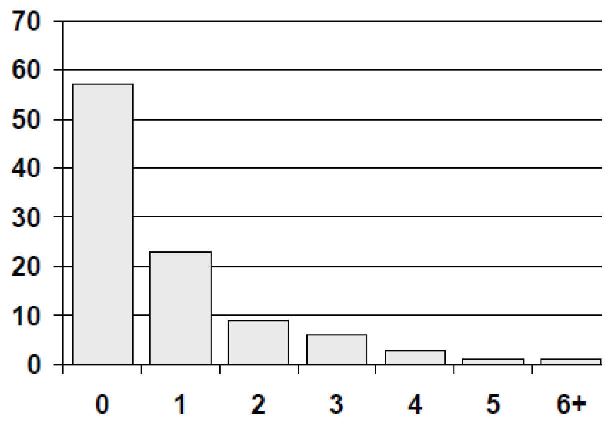


**Figure 9: % of reported algorithms vs. no. of real input sets [6].**

Many reasons were given for inadequate testing. Following gives the list of reasons along-with this paper author's comments.

A. *An algorithm proposed for a very specific application domain does not allow use of general pupose test data.*

Even if varieties of data sets are not applicable a number of data sets can be created by selection of data from the original data set and quality can be compared. Use of noisy data-sets by introducing artificial noise is useful in some cases.

B. *Non-availability of comparable algorithms for the very specific application domain.*

But the algorithm can be compared with a general-purpose algorithm.

C. *Algorithms solving a problem for which no solution was given earlier so it cannot be compared to others.*

This is true. But such algorithms were not found in the reported study.

D. *Totally new approaches to a problem do not allow for comparison.*

Software measurement is independent of the approach.

E. *Often a thorough evaluation is too much work.*

But this needs to be done.

F. *Erroneous data.*

If only erroneous data are available the algorithm should be able to eliminate the effect of errors on the output.

The study clearly shows quality measurement in this area is quite inadequate. Lots of work needs to be done. The effort takes a backseat as there are urgent needs for many practical applications using ANN algorithms. One of the biggest problems for quality measurement of ANN is the long execution time. Training phase may take a week depending on the nature of the problem. Whether the training is successful or not cannot be known during this phase. It can be known only in the next phase, the testing phase. For unsuccessful training again the very long training phase is to be repeated. Efforts are on to reduce the training phase time by developing novel architecture. D. F. Specht proposed General Regression Neural Network (GRNN) with clustering where training time is greatly reduced [7].

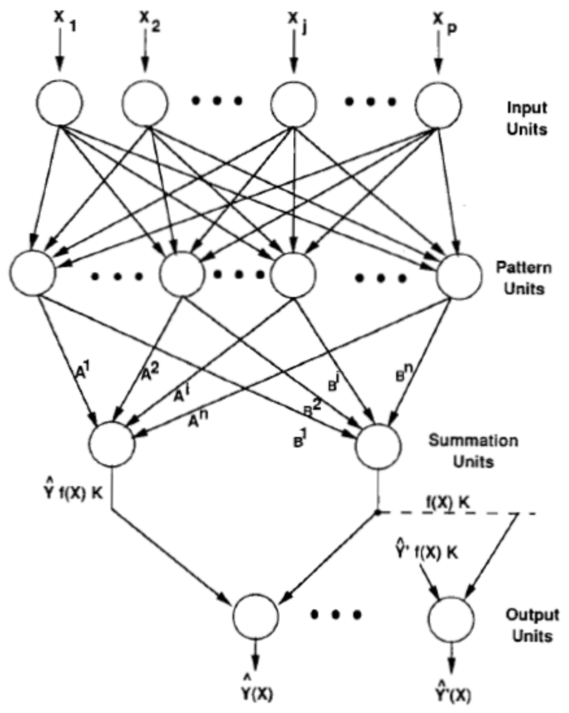


**Figure 10: GRNN block diagram [7].**

Here neurons arranged in four layers are shown by circles, In the layer prior to output layer, clustering is done that is very useful for regression analysis. It has been demonstrated that clustering greatly improves the execution time. The study has been done for simulation of a simple plant with characteristics as given in fig. 11.

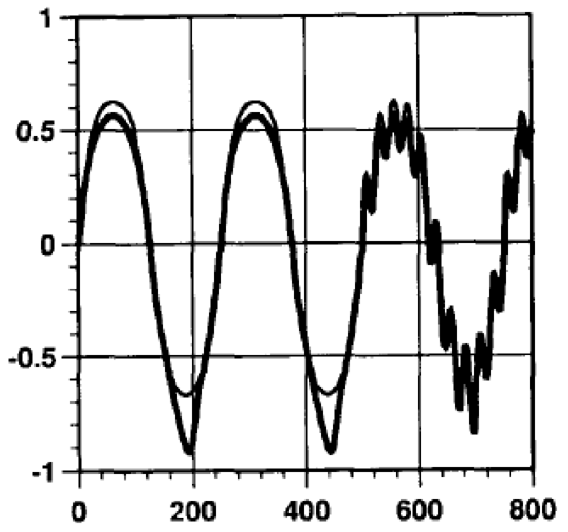


**Figure 11: Outputs in time steps of plant (dark line) and GRNN model (lighter line) after training with 1000 patterns [7].**

GRNN model works in a satisfactory way for reduced number of patterns. This can be understood if figures 11 and 12 are compared.

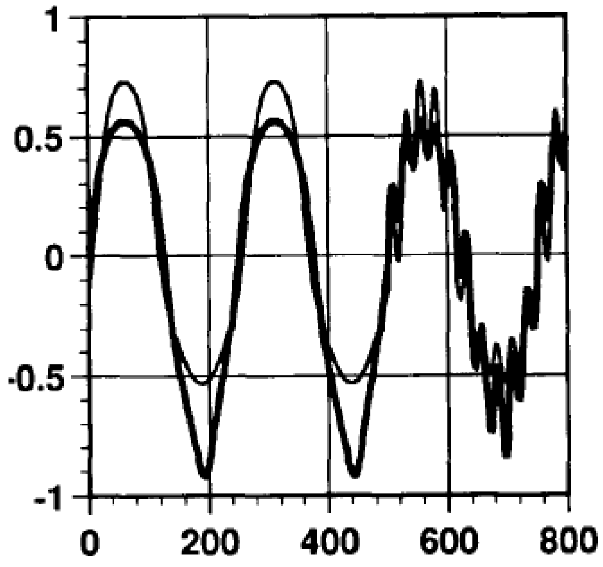


**Figure 12: Output in time steps of plant (dark line) and GRNN model (lighter line) for only 10 input patterns [7].**

I. F. B. Tronto *et al.* compared traditional regression analysis and ANN [8]. Results in Table I show ANN performs better.

**Table 1**

THE PREDICTIVE ACCURACY

| | Regress. Eq. | R- square | MMRE |
|---|---|---|---|
| ANN | -1,68+1,676*x | 0,85 | 420 |
| Regression | -1,71+1,623*x | 0,83 | 462 |

R. S. Behara *et al.* did a detailed study on Service Quality Measurement abbreviated as SERVQUAL [9]. It is the difference in customer expectations and perceptions scores. In this reported work, service quality is measured along the five conceptually distinct yet interrelated dimensions: tangibles, reliability, responsiveness, assurance and empathy. A systematic diagram of SERVQUAL for auto-service named as reverse SERVQUAL is shown partly in fig. 13.

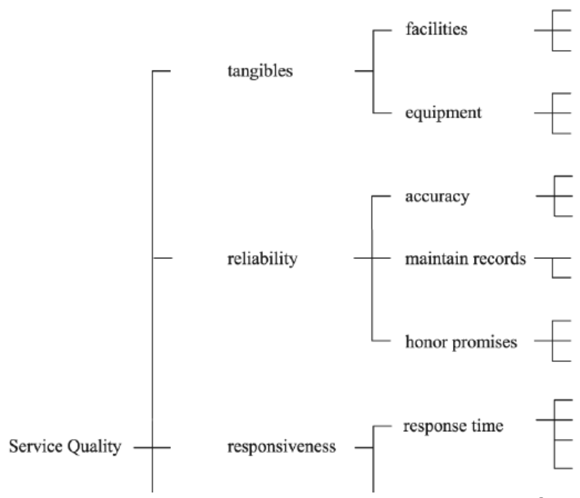
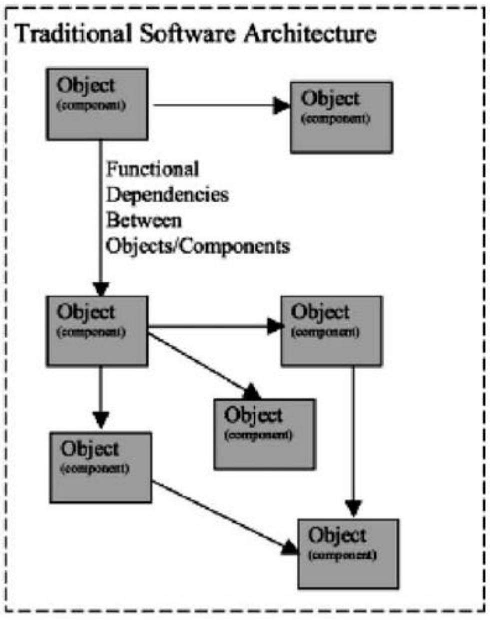


**Figure 13: Reverse SERVQUAL, systematic diagram of service quality [9].**

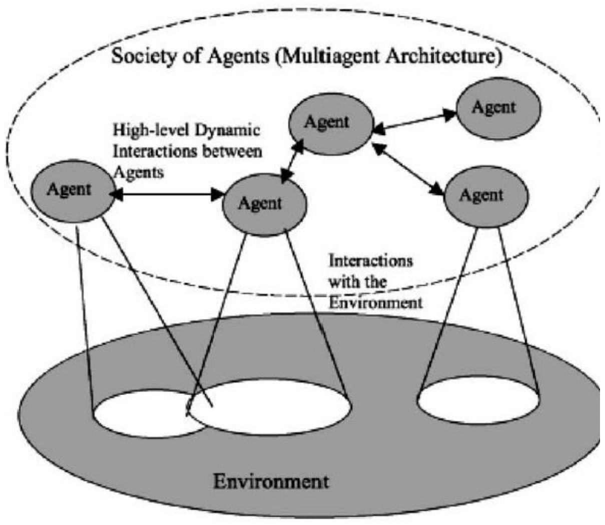
Instead of perception minus expectation model a perception only model is claimed to be more accurate. However, adequate experimental results are not available in this paper to support this claim.

M. R. Genesereth and S. P. Ketchpel applied the recently developed concept of agent based computing to AI [10]. Agents are separate modules that might be developed in different languages, architectures and platforms. Agents based computing might be thought of at higher integration level than Object Oriented Programming (OOP). In OOP objects consist of data and process in the same language, architecture and platform. Figures 14 and 15 illustrate the differences between OOP and Multi Agent System (MAS).



Traditional Software Architecture
Object (component)
Functional Dependencies Between Objects/Components

**Figure 14: Traditional software architecture [11]**



Society of Agents (Multiagent Architecture)
High-level Dynamic Interactions between Agents
Agent
Interactions with the Environment
Environment

**Figure 15: Architecture of Multi Agent Systems (MAS) [11].**

With use of agents the AI software becomes distributed and is called Distributed Artificial Intelligence (DAI). F. Zambonelli and A. Omicini [11] discussed the measurement issues with MAS. Existing tools work only for a particular language and platform. A new tool should view the interaction between agents at macro level. But at macro level quality measurement becomes an issue. If the interactions are studied at micro level it becomes too complex. Authors defined a new concept of "Meso" level of interaction to ease quality measurement studies.

M. Harman and B. F. Jones [12] applied Genetic Algorithm (GA) for information search an interesting application in the area of AI. Software measurement is an issue for such meta-heuristic algorithm.

## III. ANALYSIS

- Initially quality measurement metrics were directly taken from the established metrics for hardware components. But software quality measurement is different from hardware quality measurement.
- Software is flexible hardware is rigid. Sometimes the flexibility creates a problem for measurement.
- For AI software dynamic testing is an issue. Software usually has long execution time with learning and testing phase. Failures can be detected only at the end phase or testing phase.
- AI algorithms are of heuristic nature; this poses a problem for static testing.
- AI software work with large number of input sets. Exhaustive test for all combination of inputs seems to be impractical.
- For AI software testing at the learning phase sequence of inputs are important.
- AI software has varieties of real life applications. For many emerging applications quality measurement demands a revisit.

## IV. CONCLUSION

Software quality measurement is one of the most important tasks in software engineering. Initially the measurement metrics were taken from the existing hardware components. That creates a problem as two situations are different. Artificial intelligence software has several issues for quality measurement. Considering the market demands this area should be considered as thrust area for research.

## REFERENCES

[1] J. Rushby, "Quality measures and assurance for AI software, Technical Report CSL-88-7R, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, also available as NASA contractor report 4187, pp. 1 – 134, September 1988. Available: http://www.csl.sri.com/papers/csl-88-7/csl-88-7r.pdf
Viewed: 9th July 2015.

[2] W. J. Rapaport, "Some definitions of artificial intellgence". State University of New York at Buffalo, Buffalo, NY 14260-2000, September 2012. Available:

http://www.cse.buffalo.edu/~rapaport/572/S02/aidefs.html Viewed: 6th July, 2015.

[3] N. Fenton, P. Krause, and M. Neil, "Software measurement: uncertainty and causal modelling", IEEE Software, vol. 19, no. 4, pp. 116 - 122, July 2002.

[4] M. Khraiwesh, "Process and product quality assurance measures in CMMI", International Journal of Computer Science and Engineering Survey, IJCES, vol. 5, no. 3, pp. 1 - 15, June 2014.

[5] J. Mylopoulos, L. Chung, and B. Nixon. "Representing and using nonfunctional requirements: A process oriented approach", IEEE Trans. Software Engineering, vol. 18, no. 6, pp. 483 – 497, June 1992.

[6] L. Prechelt, "A quantitative study of experimental evaluations of neural network learning algorithms: current research practice", Neural Networks, vol. 9. pp. 1 – 7, 1995.

[7] D. F. Specht, "**A** general regression neural network", IEEE Trans. Neural Networks, vol. 2, no. 2, pp. 568 – 576, November 1991.

[8] I. F. B. Tronto, J. D. S. Silva, N. S. Anna, "Comparison of artificial neural network and regression models in software effort estimation", Proc. International Joint Conference on Neural Networks, Orlando, Florida, USA, August 2007, pp. 1 – 6.

[9] R. S. Behara, W. W. Fisher, and J. G. A. M. Lemmink, "Modelling and evaluating service quality measurement using neural networks", International Journals of Operations and Production Management, IJOPM, vol. 22, no. 2, pp. 1162 – 1185, 2002.

[10] M. R. Genesereth and S. P. Ketchpel, "Software agents", Center for Integrated Facilty Engineering, CIFE, Stanford University, CA 94305-4020, no.32, pp. 1 – 12, April 1994.

[11] F. Zambonelli and A. Omicini, "Challenges and research directions in agent-oriented software engineering", Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, vol. 9, pp. 253 – 283, 2004.

[12] M. Harman and B. F. Jones, "Search based software engineering", Information and Software Technology, vol. 43, Elsevier, pp. 833 - 839, 2001.