http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (6): 835-842, 2007
ISSN 1812-5638

Software Quality in Artificial Intelligence System

B. Vinayagasundaram and S.K. Srivatsa
Computer Center, MIT Campus Anna University,
Chromepet Chennai-600044, Tamilnadu, India

Abstract: The main objective of the study is to define the metrics to measure the quality of software in the architecture for an artificial intelligence system. The proposed architecture for measurement consists of four components; Task specification layer, problem solver layer, domain layer and an adapter layer. These four components are hierarchically organized in a layered fashion. In this architecture, the overall structure is decomposed into sub components, in a layered way such that a new layer can be added to the existing layer that can change the behavior of the system. The quality of components in the architecture are measured with metrics such as source code, depth of inheritance, number of paths, complexity level etc., These metrics are related to software quality characteristics suggested by ISO. This study is organized in the following way; Firstly, the study addresses the significance of software architecture in a software intensive AI system, the importance of quality of the software in the architecture and a layered architecture for artificial intelligence system. The secondly, the study addresses the relation ship between the quality characteristics and the metrics used for measuring the quality. The performance of the system with respect to functional requirement and non-functional requirements are measured and discussed.

Key words: Artificial intelligence, software architecture, quality, software engineering

INTRODUCTION

Quality is a multi-dimensional construct comprising of various parameters, which are reflected in quality modeling. Although, several quality attributes to measure the software quality are specified in the literature, like FURPS, McCall, Boehm, they are related to various aspects of the software such as product revision, transition and operation. Hence it is necessary to select appropriate attributes that are relevant to the software pertaining to Artificial Intelligence systems. In this study, the quality attributes, which are relevant to the software of Artificial Intelligence systems, are identified as testability usability readability and self-descriptiveness. Metrics are related to these attributes and measured for the software applicable to Artificial Intelligence System. The attributes selected for measurement in this study is based on ISO 9126 model.

The definition for Intelligence as per Hideyuki Nakshima (1999) is:

Intelligence is the ability to process information in a properly complex environment in a partial way; nevertheless this partial processing mechanism yields maximally successful behavior. Even though AI is a sub field of information processing, there are some important differences, in information processing complete processing is a must, whereas in Artificial intelligence processing in most cases complete processing is not possible. The illustrations for the above is, in information processing, the algorithm for searching a data must be complete, but in AI processing complex heuristics are used to reduce the search and in some cases the best options may be missed occasionally. Hence the AI systems should be architected in an evolutionary way not only to fulfill the functional requirements or services that the system has to accomplish but also to take in to account the quality of the service provided which are non-functional requirements such as portability, efficiency etc., Several constraints like type of the platform, kind of network are also non-functional requirements, which will have the impact on overall quality of the system.

Software architecture and its relation to quality: Software architecture of a system describes the structure, organization of components/modules and their interactions not only to satisfy the systems' functional and non-functional requirements but also provide conceptual integrity to the overall system structure. Software architecture concerns with the structure of large

Corresponding Author: B. Vinayagasundaram, Computer Center, MIT Campus Anna University, Chrome pet Chennai-600044, Tamilnadu, India Tel: +91 44 22230850/22237276

software intensive systems (Garlan, 2000). The architectural view is an abstract view that separates the details of implementation, algorithm and data representation and concentrates on behavioral aspects and interaction among the various components. In other words, the software architecture of the system provides an abstract description of the system by exposing certain properties and hiding others (Land, 2002). Hence the software architecture plays an important function with respect to following aspects in the development of large software intensive systems.

- **Understandability:** It helps to understand the large system by the appropriate level of abstraction. It also exposes the high-level design constraints, thereby providing way for making architectural decision.
- **Reusability:** Architectural designs support the reuse of large components and provide frameworks into which components can be integrated.
- **Construction:** An architectural description provides a blue print for the development of the system indicating the major components and the relationship among them.
- **Evolution:** The architectural description of the system separates the functionality from implementation, thereby permitting to manage the concerns regarding performance, reusability and prototyping in an easy way.
- **Analysis:** The architectural description provides a new attribute for analyzing the system with respect to quality, performance, dependency etc. Moreover analysis of architectures built with different styles can also be made to arrive at good architectural design decisions.
- **Management:** Successful development of software, addressing specific application depends on critical selection, analysis and evaluation of software architecture.

Artificial Intelligence systems are large and complex. The more powerful way of structuring the complexity lies in architecting the system. Hence an efficient method is needed to structure and handle the complexity of these systems. A good architecture of the system will not only satisfy the functional requirements but also satisfies the key non-functional requirements of the system such as performance, reliability, portability, maintainability etc. Hence software architecture is the very first step in the development of large software intensive system in which non-functional i.e., quality requirements are addressed (Reza and Grant, 2005).

SIGNIFICANCE OF QUALITY

Many authors have defined quality. Some definitions are given below:

- Fit for use
- Compliance with specified requirements
- Free from defects, imperfection or contaminations
- Degree of excellence
- Customer satisfaction
- Delighting the customers

ISO 8402: Definition: The totality of characteristic of an entity, that bears on its ability to satisfy the stated and implied needs. Quality can also be expressed by the simple formula Q = P/E, where P = Performance or result, E = customer expectations. When Q = 1, the customer expectations are complied with and full customer satisfaction is attained, which is an ideal one. The concept of quality according to ISO is meeting customers' requirements. A product or service has a quality when it satisfies the user needs both stated and implied. Ultimate survival of any product or service depends on meeting the customer requirement. Hence quality forms the core task of a product development, which is not an option but essential characteristic of a process and a product.

REVIEW OF VARIOUS QUALITY MODELS

McCall's quality model: In the literature several quality models are proposed for the measurement of software quality viz., McCall's Quality factors in 1977, Boehm's Quality model in 1978, FURPS in 1987 and ISO 9126 in 1991 and Dromey model in 1996. McCall's Quality model is based on product's operations, revisions and transitions. This model combines eleven factors viz. Correctness, reliability, efficiency, integrity, usability, maintainability, portability, reusability and interoperability. The main idea behind McCall's model is to measure the relationship between external quality factors and internal quality of the product. McCall's model is used in vary large software intensive systems. Even though this model has some drawbacks the major contribution it created is a baseline relationship between the quality characteristics and metrics.

Boehm's quality model: Boehm added some more characteristics to McCall's model stressing on the maintainability. This model is based on wider range of characteristics that incorporates 19 factors. Boehm's model also includes characteristics related to hardware performance, which are not included in McCall's Model.

This model aims at utility from various dimensions, considering the types of user expected to be working in the system. General utility is refined into portability, utility and maintainability. Utility is further refined into reliability efficiency and human engineering. Similarly, maintainability is refined in to testability, understandability and modifiability. The limitation in Boehm model is it does not suggest about measuring the quality characteristics.

FURPS model: The FURPS model proposed by Robert Grady and Hewlett Packard Co. decomposes the requirements in to two different categories viz., Functional requirements defined by the input and the expected output and Non-functional requirement defined by usability, reliability, performance and supportability. FURPS takes into account the five characteristics that constitutes its name viz., functionality, usability, reliability, performance and supportability. The limitation in this model is it does not account for one of the important characteristic that is portability. This model is based on requirement analysis and modeling.

ISO quality model: In order to standardize the evaluation of software and software products ISO proposed a standard, which specifies six independent high-level quality characteristics. These quality characteristics are uses as targets for measuring external quality (validation) and internal quality (verification) (Vinayagasundaram and Srivatsa, 2004) These characteristics are related to measurable code metrics in this study. The quality characteristics of ISO 9126.1 standard for software quality measurement are shown in Table 1. One of the advantages of ISO 9126 model is that it identifies the internal and external quality characteristics of a software product (Chirinos, 2004).

Dromey's quality model: Dromey propsed a model consisting of eight high level quality characteristics, namely six from ISO 9126 and Reusability and Process maturity. This model suggested a more dynamic idea for modeling the process on three prototypes concerning quality; (I) Implementation quality model (ii) Requirements quality model (iii) Design quality model. Dromey model provides the relationship between characteristics and sub characteristics of quality. It also attempts to pinpoint the properties of the software which affects the attributes of quality.

Hybrid layered architecture for quality measurement: Most AI systems display a rigid separation between the standard computational components of data, operation and control. That is, if these systems are described at an appropriate level one can identify three important components viz., knowledge database that is manipulated by well-defined operations all under the control of global control mechanism. Even though at machine code level, any neat separation in to distinct components can disappear; it is important to specify them at appropriate level of description. One difficulty in using conventional software systems with hierarchically organized programs for AI applications is that modifications to knowledge base might require extensive changes to various existing programs, data structures and subroutine organization. In the proposed hybrid architecture for Artificial Intelligence system, components are defined in a modular way. The system design is more modular and changes to any of the components can be made relatively independently. Apart from describing the system with distinct components, the critical issue in the construction of AI system is its architecture; that is its gross organization as a collection of interacting components. A good architecture can ensure that a system will satisfy not only the key functional requirements but also ensures the non functional requirements such as reliability, portability, modifiability etc., In the proposed hybrid architecture, four distinct components are defined at appropriate level as shown in Fig. 1. (Vinayagasundaram and Srivatsa, 2007). A task definition layer defines the problem that should be solved by the system. The second component viz., Problem solver layer defines the method for reasoning and a domain model layer describes the domain knowledge of the AI system. These three components are described independently to enable the reusability. The fourth component Adapter layer adjusts the three other components to each other and to the specific problem.

Table 1: Features of ISO 9126.1 quality model

Characteristic	Description
Functionality	The capability of the software system to provide functions that meet stated and implicit needs when the system is used under specified conditions.
Reliability	The capability of the software system to maintain its level of performance under stated conditions for stated period of time.
Usability	The capability of the software system to be understood, used, learned and attractive to the user when used for specified conditions
Efficiency	The capability of the software system to provide appropriate performance relative to the amount of resources used under stated conditions
Maintainability	The capability of the software product to be modified. Modifications may include corrections, improvements or adaptations of the system to change in environment and in the requirements and functional specifications
Portability	The capability of the software product to be transferred from one environment to another. The environment may include organizational, hardware and software

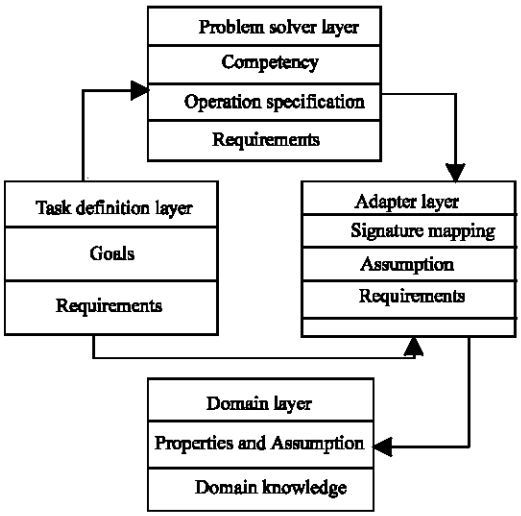

Fig. 1: Hybrid layered architecture for artificial intelligence system

FORMAL SPECIFICATIONS OF COMPONENTS

Task description layer: The task description layer consists of two elements viz., the goals that should be achieved to solve a given problem. The second element of the task description layer is the definition of the requirements on the domain knowledge. Usually, axioms are used to define the requirements. The task definitions are done by algebraic specifications that provide a signature consisting of types constants functions predicates that defines the property of this component.

Problem solver layer: The concept of problem solver layer is present in many AI frameworks. The problem solver describes the reasoning steps and which types of knowledge are needed to solve the given problem. Even though, there are some differences in the approaches, the following features are common in almost all problem solvers:

- The problem solver should decompose the entire reasoning process into primary set of elementary inferences.
- The problem solver should define the type of knowledge that is required by the inference steps to be done.
- The problem solver should define the control and flow of knowledge between the inferences.

The description of problem solver layer consists of three elements viz., Competency description, operational specification and requirements on domain knowledge. The competency of a problem solver is a logical theory that characterizes the solution process. It is similar to the specification of functionality in software engineering. Selection of a solution method for the given problem and the verification of whether the problem solver fulfils the requirement for the solution process can be done independently from the details of the internal reasoning behavior of the method by proving that the Problem solver has some competency. The operational specification defines the dynamic reasoning of the problem solver, which explains how the competency is achieved. The third element introduces the requirements on domain knowledge. All the inference steps and competency description of the problem solver requires the specific types of domain knowledge. These requirements on domain knowledge distinguish a problem solver from conventional software. Competency description specifies the actual functionality of the AI system where as the task description specifies the problem that should be solved by the AI system. Distinction between the competency description and task description is made because of the following reasons.

- The problem solver introduces requirements on domain knowledge in addition to the task description. Even though this knowledge is not necessary to define the problem it is required to describe the solution process.
- It is not always assumed that the functionality of the AI is strong enough to completely solve the problem. Hence the problem solver should introduce some important assumptions to reduce the problem size.

This is similar to the distinction between functional specification and the design- implementation of software in software engineering. The functional specification deals with what and design specification deals with how. This separation is often not practically possible even in the domain of software engineering, would not possible in the development of artificial intelligent system because a large amount of problem solving knowledge is required, that is knowledge about how to meet the solution requirements is not a question of algorithm and data structures, it exists as a result of experience over the years and heuristics. Even though, some problems are completely specifiable but it is not necessarily possible to derive efficient algorithms from such specifications. Hence it is not sufficient to have knowledge about what is the solution to the problem, but also have knowledge about how to derive such a solution in an efficient way.

The domain layer: The description of the domain layer introduces the domain knowledge required by the problem solver and the task description layer. To represent domain

knowledge ontologies are proposed so that the knowledge can be reused. In the domain layer of the architecture three elements are defined viz., properties, assumptions and the domain knowledge itself. The domain knowledge is necessary to define the task in the given application domain to carry out the inference steps of the problem solver. Properties are derived from the domain knowledge and assumptions are the attributes that have to be assumed to be true. Hence, the properties and assumptions are both used to characterize the domain knowledge.

The adapter layer: Adapters are of general importance in component-based software. In the hybrid architecture, adapter layer is introduced to relate the competency of the problem solver to the functionality given by the task description layer. Further, the adapter layer introduces new requirements and assumptions because, the most problems tackled with AI systems are in general complex and intractable. Hence, a problem solver can solve such tasks with reasonable computation by the way of partial processing by introducing assumptions that restrict the complexity of the problem or by strengthening the requirements on the domain knowledge. The other three layers viz., Task description, problem solver and domain layer can be described independently since the adapter layer combines these layers in such a way that meets the solution requirements of the specific application. The consistency in the relation and the adaptations to the specific aspects of the problem makes it reusable.

METRICS USED FOR MEASURING QUALITY

Conventional software development projects based on life cycle models are composed of several stages. A series of products are generated apart from the source code. The design and development process determines the systems' final behavior and hence the measurement cannot be restricted to a single product. With regard to conventional software even though the methodologies may be different the products generated are common, like functional specifications, high-level design, low-level design has necessitated the continuous evaluation of these products through out the development process. This is evident from the several standards published for different stages of the system development life cycle. While this is applicable for software engineering, it is not applicable for the development of artificial intelligent system because:

- Firstly, it is not feasible to accept AI system as conventional software products, because it is not possible to apply conventional software methodologies for AI system development, since these systems have broader capabilities and functionalities.
- Secondly, AI development methodologies vary enormously as to their depth and orientation. Therefore, there is no direct means of building a measurement environment valid for AI system developed according to different methodologies.

In the proposed architecture there are four layers each consists of several components. We limit our analysis to components in the problem solver layer. The components are measured using the following metrics. (Fenton and Pfleeger, 1997)

The accepted range for each of the metric is shown in the parenthesis.

- **Number of lines (LOC):** Counts the number of executable statements per component {1-200}
- **Cyclomatic Complexity:** (McCabe, 1976) (VG): As defined by Tom McCabe. It is a metric based on graph theory, which measures the logical complexity of a program. It is considered as an indicator of the effort needed to understand and test the component {1-15}.
- **Maximum levels of nesting (MAX_NST):** Measures the maximum number of nesting in the control structure of a component. Excessive nesting reduces the readability and testability of the component {1-15}.
- **Number of paths (NO_PATHS):** Counts the number of non-cyclic paths per component. It is another indicator of number of test cases needed to test a component {1-80}.
- **Unconditional jumps (UNCOD_GOTO):** Counts the number of occurrences of GOTO. Normally in structured programming this statement should be avoided {0}.
- **Ratio of Comment statements (COM_R):** Defined as the proportion of comment lines to number of executable statements {0.2-1}.
- **Vocabulary frequency (VOC_F):** Defined by Halstead as the sum of the number of unique operands and operators that are necessary for the definition of the component {1-4}.
- **Program length (PR_L):** Measure the program length as the sum of the number of occurrences of the unique operands and operators {3-500}.
- **Average size (AVG_SIZE):** Measures the average statement size of the component and is equal to PR_L/LOC {3-7}.

Table 2: Metrics related to quality attributes

Attribute	Metric Used
Testability	VG,MAX_NST,NO_IO
Usability	VG,LOC,AVG_SIZE
Readability	VG,PR_L,MAX_NST,AVG-SIZE
Self descriptive ness	COM_R

- **Number of input/outputs (NO_IO):** counts the number of input and exit points of the component {2}.

The above metrics are related to the software quality attributes as shown in Table 2.

In order to assess the testability the conformance to the predefined ranges for the metrics viz., VG, MAX_NST, NO_IO are examined. From the table it is seen that only seven metrics are used for assessing the quality of the component. The metrics are used according the empirical formulae as shown below:

Testability = 0.4*BVG+0.4*BMAX_NST+0.2*NO_IO
Usability = 0.4*BVG+0.3*BLOC+0.3*BAVG_SIZE
Readability = 0.4*BVG+0.2*BPR_L+0.2BMAX_NST+ 0.2*BAVG_SIZE
Maintainability = 0.3*BVG+0.3*BMAX_NST+0.2* NO_IO+0.2*BCOM_R

All the variables shown in the above formulae are Boolean variables, which assumes the value 1, when the corresponding metric conforms to the range and 0 otherwise. For example, if the VG of the component is in the acceptable range then BVG is 1 and 0 otherwise. In the testability measure the weight factor of 0.4 is assigned to VG and 0.4 is assigned to MAX_NST and 0.2 to NO_IO. Similarly, all the other quality attributes are assigned with weight factors.

PROPOSED ARCHITECTURE APPLIED TO TEXT CLASSIFICATION

To evaluate the functional and non-functional performance of the proposed architecture, the task of text categorization is used. The objective of the text classification is to assign a number of appropriate categories based on the content. To carry out this task manually, a large amount of human resources are needed. Several features of text classification task make it different from other artificial intelligence problems. Firstly, the problem spaces of text classification involve a high dimensional space (Dolores Del Castillo and Jose Ignacio Serrano). Secondly, even though the problem space is high the each document contains only a small number of features, which are sparse. Some of the standard methods for text classifications are K-nearest neighbor algorithm, Bayesian algorithm and back propagation algorithm. The main disadvantage of K-nearest neighbor algorithm is it makes use of training examples as instances for computing similarity. To overcome this k-nearest neighbor is enhanced such that it makes use of generalized instances for computing the similarity, which is called as Generalized Instance Set algorithm. The main idea of this enhanced version is to construct the generalized instances to replace the original training examples. Given a particular category, it can be observed that the regularity among the positive examples is usually more than that of the negative examples. The classification knowledge induced from a pool of similar examples is relatively accurate. But, on the other hand negative examples close to such a pool are likely incorrect negative instances or noise. Based on this idea the Generalized Instance Set algorithm focuses on refining the original instances and constructs a set of generalized instances. First it selects a positive instance and conducts a generalization process using k nearest neighbors(Wai Lam and Yiqiu Han, 2003). After a generalized instance is formed it is used as a new starting point and the process is repeated based on nearest neighbors. This search is repeated until no positive instance remains.

RESULTS AND DISCUSSION

The performance of K-nn, GIS and Bayesian algorithms are implemented in the problem solver layer and functional performances and non-functional performances are measured and the results obtained have been compared. The functional performance measurement for text classification is usually done by macro averaged recall/precision break-even point measure (F measure). In this scheme precision and recall are two important parameters used in calculating the F measure. For a category I, the precision is defined as the ratio of number of documents classified according category I by the classifier. Recall is defined as the ratio between the numbers of documents correctly classified to category I to the total number of documents actually belonging to I.

$$\text{Macro averaged } F = \Sigma F(I)/m$$

Where m is the total number of categories and

$$F(I) = (2 * \text{precision}(I) * \text{recall}(I))/(\text{recall}(I) + \text{precision}(I))$$

The Fig. 2 shows the performance of the problem solver implemented with Generalized Instance set (enhanced K-nearest neighbor) and K-nearest neighbor algorithms. The recall level at which K-nearest neighbor starts is lower than the recall level of Generalized Instance set. For the recall level between 0.5 and 1 it is seen that

Table 3: Macro averaged F measure for various methods (Functional performance measure of three methods)

Category	Generalized instance set method	K-nearest neighbor method	Bayesian method
Corn	0.4736842105263158	0.4909090909090909	0.1544715447154472
Crude	0.6824324324324325	0.6590909090909091	0.4353741496598639
Earn	0.9089099054255849	0.9374130737134908	0.7619047619047620
Ship	0.6585365853658537	0.4979919678714860	0.4353312302839116
Interest	0.5148514851485150	0.4162679425837320	0.3584229390681003
Trade	0.7326732673267328	0.6249999999999999	0.4256756756756756
Acq	0.8706838185511171	0.7193277310924370	0.4645030425963488
Wheat	0.7260273972602739	0.5714285714285715	0.2321428571428571
Money-fx	0.4020100502512563	0.4090909090909091	0.3953488372093023
$\sum F(I)/m$	0.6633121280320090	0.5918355773089584	0.4070194486951410

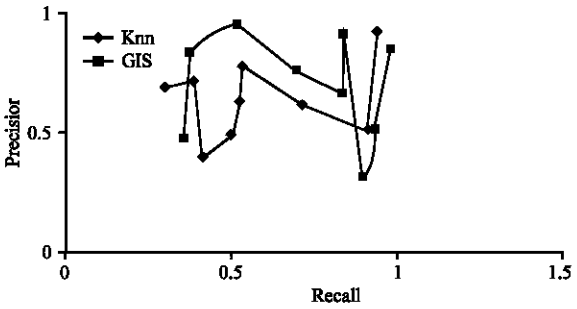

Fig. 2: Relative performances of GIS and K-Nearest Neighbor Algorithm

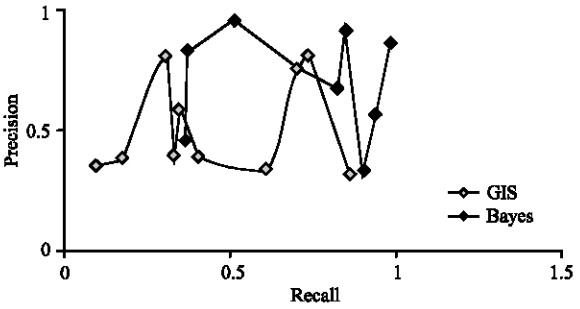

Fig. 3: Relative performances of GIS and Bayesian Algorithm

the precision level of Generalized Instance set method is less than k- nearest neighbor. This is because k nearest neighbor method operates well on this data without noise. Hence, there is a slight increase in the performance for this recall level. But when the macro averaged precision F is compared it is inferred that the average precision of GIS is more than that of K-nearest neighbor and also the macro averaged break-even point measure for GIS is better than K nearest neighbor. To over come this problem feature set pertaining to each category can be extracted and generalized instance can then be formed. When this is done GIS method will show better performance for all recall levels.

The Fig. 3 shows the performance of the problem solver implemented with enhanced K-nearest neighbor method and Bayesian method. Except for recall level at 0.4, the precision is more for enhanced K-nearest neighbor method than that of Bayesian. The precision is more for all other recall levels. The macro-averaged break-even point measure is also higher for GIS method.

Table 4: Quality measure of three methods (Non-functional performance of three methods)

Method	Testability	Usability	Readability	Maintainability
GIS	0.8	0.7	0.8	0.8
K-nn	0.6	0.6	0.6	0.4
Bayes	0.6	0.6	0.6	0.4

The Table 3 gives the macro averaged F measure for various categories in the data set for Generalized Instance set, k-nearest neighbor and Bayesian classification method. From the Table 3, it is inferred that the macro-averaged precision is also higher for Generalized Instance Set (enhanced K-nearest neighbor). In this study, Bayesian method-nearest neighbor method and the enhanced version of it were implemented in the problem solver layer of the hybrid layered software architecture in a reusable way. It is also found that the F measure for enhanced version is 12% higher than primitive version.

Apart from the measurement of functional performance, the metrics discussed has been used for the measurement of the non-functional performances namely, the quality requirements. In this architecture, the problem solver layer is reused with various methods. The measure of various quality attributes with respect to different methods of the text classification is shown in the Table 4. By specifying suitable metrics, the quality of the software in the AI system architecture is measured. It is found that GIS method shows better quality in terms of various quality attributes. The K-nearest neighbor method scores second in terms of quality. The Bayesian method implementation also provides the quality to acceptable level. In order to further improve the quality of the software, the software in the problem solver layer can be optimized so that the metrics are in the acceptable range.

CONCLUSIONS

Software architecture for artificial intelligence system has been developed applying conceptual and formal framework based on reusable components. In this

architecture, the overall structure is decomposed into sub components, in a layered way such that adding new layer without modifying the existing layers can change the behavior of the system as a whole. Hence, an AI system can be built in an evolutionary way by combining and adapting several reusable components. The performance of the architecture with respect of functional requirement and non-functional requirement has been measured and discussed. Future work may be focused on the development of a semantic search layer, which can be augmented to the existing system without changing the architecture that can change the behavior of the system. The consistency checker in the adapter layer cannot make changes directly to the knowledge base in the domain layer. Hence the present work can be extended to perform this task by suitably modifying the task description layer. The measurement of quality discussed above can be extended to the proposed addition to the system.

REFERENCES

Chirinos, L.F., 2004. ISO Quality standards for measuring Architectures J. Sys. Software, 72: 209-233.

Dolores del castillo jose ignacio serrano, a multistrategy approach for digital text categorization from imbalanced documents sigkdd explorations. Vol. 6, pp: 70-77.

Fenton, N.E., S.L. Pfleeger, 1997. Software Metrics: A Rigorous and Practical Approach II edition PWS Publishing company, Int. Thomson Computer Press London.

Garlan, 2000. Software Architecture: A Roadmap ACM press.

Lam, W. and Y. Han, 2003. Automatic textual document categorization based on generalized instance sets and metamodel. IEEE Trans. Pattern Analy. Mach. Intelli., 25- 628-633.

Land, R., 2002. A brief survey of software architecture MRTC report. Malardalen University Sweden.

McCabe, T., 1976. A complexity measure IEEE Trans. Software Eng., pp: 308-320.

Nakshima, H., 1999. AI as Complex processing Minds Mach., pp: 57-80.

Reza, H. and E. Grant, 2005. Quality-oriented Software architecture. In: The proceedings of the international conference on Information Technology ITCC 2005.

Vinayagasundaram and Srivatsa, 2004. Software architecture quality in artificial intelligence systems. In: The proceedings of international conference on emerging technology ICET-2004 pp: 307-311.

Vinayagasundaram and Srivatsa, 2007. Implementation of Hybrid software architecture for Artificial Intelligence System IJCSNS Jan2007. pp: 35-41.