Evaluating Quality of AI-Based Systems

Satvir Kaur Toor and Parvinder Singh Sandhu

Guru Nanak Dev Engineering College, Gill Park, Ludhiana, India.

Summary
The main objective of the work is to provide a general setting for quantitative quality measures of Knowledge-Based System behavior. It includes 'Metrics Suite studies': an analysis of all the metrics available related to AI (Artificial Intelligence) based systems which includes both qualitative metrics and quantitative metrics and it is shown that how system quality changes as a function of values of the design descriptors of the AI programs. To show the feasibility of this approach, we have applied it in Prolog Language. AI is the part of computer science concerned with designing intelligent computer systems, that is, computer systems that exhibit the characteristics we associate with intelligence in human behavior-understanding language, learning, reasoning and solving problems. It is the ability of these new electronic machines to store large amounts of information and process it at very high speeds that gave researchers the vision of building systems which could emulate some human abilities. AI is the branch of science concerned with the study and creation of computers systems that exhibit some form of intelligence: systems that learn new concepts and tasks, system that can reason and draw useful conclusions about the world around us, systems that can understand a natural language or perceive and comprehend, a visual scene, and systems that perform other types of feats that requires human types intelligence.

Key words:
Artificial Intelligence, Knowledge Base, Metrics, Quality, Prolog

1. Introduction

There is lack of effective methods for ensuring the quality and reliability of AI system, means these metrics lack in measuring all the basic mechanisms of AI systems paradigm and moreover there is no correlation revealed between the basic mechanisms and the quality attributes. One impediment to the growing use of AI-based software is the lack of metrics suite for determining various characteristics as well as effective methods for ensuring the quality and reliability of this type of system. In particular, the nature of AI applications often confounds conventional software requirements specification techniques, while the characteristic architectures of AI-based systems confound conventional verification and testing techniques. This has lead to a great deal of interest in studying the metrics suite for AI-based systems and then with further refinement of metrics takes place in order to extract its design factors from the design of AI-based software. So the need to study the Artificial Intelligence lies in the idea of developing such metrics based on the design descriptors of AI, so that these metrics gives an idea about the aspects of AI programs. Measurement of different aspects related to the program code here refers to measurement of coverage, granularity, extent, scope, correctness, consistency, robustness, effectiveness, efficiency, completeness and validness. From the refinement of metrics suite, the main work is how to evaluate the quality of the AI programs which are already considered as complex programs and it turns to be more difficult when there were no criteria on the basis of which one can evaluate quality of these systems. It is clear from the survey of the past literature that till now there was no strategy based on metrics which can evaluate the quality in an either way. It is also written that as AI programs have large complexity, they are usually degraded programs in relation to quality. So it's very difficult to measure quality traits of such AI programs. The work tends to evaluate the quality of software by means of metrics which are suited to quantify design characteristics in the different stages of software development. Results of empirical validation studies are reported to show the analytical and predictive power of metrics.

2. Evaluating Quality based on Metrics Suite

In order to evaluate the quality of the AI systems based on the Metrics Suite, the need is to extract the design descriptors from the design of AI based systems This has lead to a great deal of interest in studying the metrics suite for AI-based systems and then with further refinement of metrics takes place in order to extract its quality factors from the design of AI-based software.

To study AI systems, all the metrics related to these systems directly or indirectly are collected and then thoroughly studied and revised. These metrics suite include both qualitative and quantitative aspects, means AI systems qualitative and quantitative behavior. With the help from the Metrics Suite, a set of Prolog files are analyzed and its design descriptors are determined. Design descriptors are the facets that contain the manifestation of Artificial Intelligence systems. These traits characterize these systems in all ways: - Quantitatively and Qualitatively. Prolog file is read by the framework engine designed and upon analyzing the file, its various facets are

Manuscript received August 5, 2007
Manuscript revised August 20, 2007

determined. The purpose of the work is to study the metrics suite for the AI based systems in order to refine the metrics suite which adds to the fact of designing the framework engine to extract the various characteristics of these Knowledge Based Systems. From the study of the past literature, it is clear that there exists no particular metrics suite for the AI based systems specially Prolog Based Systems. And in the last, a quality model is provided which describes the different descriptors which can be considered to measure the performance and quality of these knowledge based systems.

So the first step is to study the metrics suite of AI systems and then extracting the metrics from which we can further determine the design descriptors in order to reveal the design of AI systems, especially prolog systems. A set of rules is specified on some contrary basis which provides the extent of quality for these AI based systems. Depending on the value of these design facets and the rules specified, the quality of AI systems can be determined. But this metrics suite and rule set cannot be compared as there was no strategy or framework exists earlier which can be used to evaluate the quality of these prolog systems.

2.1 Metrics Suite for AI-based Systems

Metrics suite for the AI systems includes both qualitative and quantitative behavior. All the metrics collected over and studied in order to analyze the prolog systems are given below with their important aspects.

Metrics for Appropriateness Measurement: Appropriateness measurement attempts to use answer patterns to recognize a typical examinees. These procedures are the statistical tests for choosing between a null hypothesis of normal test-taking behavior and an alternative hypothesis of a typical test-taking behavior.

Coverage Metrics: Coverage analysis is the process of finding areas of a program not exercised by a set of test cases, creating additional test cases to increase coverage, and determining a quantitative measure of code coverage, which is an indirect measure of quality and identifying redundant test cases that do not increase coverage.

Data Flow Testing: The Data Flow Testing method selects test paths of a program according to the locations of the definitions and uses of the variables in the program. If the program user wants to test the data flow in program, one can do it by discovering the minimum number of paths [1]. A challenge of path coverage is to discover the input values that will cause a particular path to be executed. Using data flow graphs to create DEF USE paths for variables used in the program [2] makes it easier to discover how a particular input value affects program flow. Another set of constructs, based on dataflow, traces values from their definition point to their subsequent usage [3].

Granularity Metrics: It is a measure of the size or descriptions of components, which make up a system. Systems of large components are called coarse-grained, and systems of small components are called fine-grained.

Checking Prolog Policies: The Prolog policies specify constraints on the classes and instances of entities allowed for performing certain kinds of tasks. An example of a class-level constraint is that no Audio Presentation application component should be used to notify a user in case he is in a meeting. This rule is expressed as:

$$disallow(Presentation, notify, User) :- subclass(Presentation, audioPresentation), activity(User, meeting).$$

Policies specify which parameter values may be preferred depending on the state of different entities, the context and state of the environment, the semantic similarity of the class of the value to the developer-specified class and the end-user performing the task and assigns numerical values to the utility of different entities in different contexts [4].

Time Complexity Metrics: To measure the complexity in terms of time, the main task is to determine the similar classes of entities from the ontology hierarchy. The average time to discover all these classes is $O(n/i)$, where the number of concepts in the ontology hierarchy is n and i is the length of the path from the root of the hierarchy to the fourth level ancestor. The step is to checking class level constraints. The time taken to check the constraints is given in equation (1):

$$time = O(\#semantically\ similar\ leaf\ classes) \times O(\#class\ constraints) \quad (1)$$

The next step is querying the Prolog Knowledge Base and other repositories to get instances of the similar classes that satisfy instance level constraints. Ref. [5] shows the number of queries sent to the Prolog Knowledge Base is $O(n)$, in the worst case, where n is the number of classes as given in equation (2) as

$$number\ of\ instances = O(n) \times O(\#instances\ per\ class) \quad (2)$$

The final step is ranking the different instances to choose the best ones. So, the net complexity of the whole process is given as equation (3) where the number of concepts in the ontology hierarchy is n.

$$O(n)+O(n)\times O(\#class\_constructors)+O(n)\times O(\#instances\_per\_class)\times O(\#instance\_constructor)+ \\ O(n)\times O(\#instances\_per\_class) \quad (3)$$

Extent Metrics: the graphic token based metrics and the volume based metrics are used to measure the extent of prolog systems.

Graphics Token Based Metrics: Approach is to extend the Graphic Token Count of Nickerson [6] to produce operator and operand counts. The standard Software Science notation [7] includes as η_1 is the Number of unique operators, N_1 is the total number of operators, η_2 is the Number of unique operands, N_2 is the total number of operands, Vocabulary and N is the Software Science "size". Number of unique operators as given in equation (4),

$$\eta_1 = \begin{pmatrix} number\_of\_edge\_types + number\_of\_label\_types + enclosure\_present + \\ adjoinment\_present \end{pmatrix} \quad (4)$$

And, Number of unique operands as given in equation (5),

$$\eta_2 = (number\_of\_node\_types + number\_of\_textual\_token\_types) \quad (5)$$

Correctness Metrics: Correctness of a program is asserted when it is said that the program is correct with respect to a specification. Functional correctness refers to the input-output behavior of the program.

Measuring Function Point from Specifications: A knowledge based approach for the automated measurement of the Function Point Metric [8] starting from the specifications of a software system expressed in the form of an Entity Relationship Diagram plus a Data Flow Diagram. An integration of the two diagrams, as illustrated in [9], ER-DFD, in which the data stores of DFD are substituted by the entities and relationships of the ER. A knowledge based system automatically counts Function Point by analyzing the graph [10].

Scope Metrics:

Interface Size: Procedure Count (np) is the total count of procedures that are publicly declared by an interface in the program. Argument Count (na) is the total count of arguments of the publicly declared procedures. Procedure Count and Argument Count allow measurement of component size using only the interface. Interface size has an effect on component reusability. It is believed that interfaces that have fewer procedures and arguments tend to be easier to understand. This implies that smaller component interfaces will have better reusability [11].

Arguments per Procedure (APP) measure the mean size of procedure declarations of an interface, and given as equation (6):

$$APP = na/np \quad (6)$$

Metrics for Adequateness Measurement: Adequateness means that the program has sufficient ability to satisfy a requirement or meet a need. Quality of being able to meet a need satisfactorily is adequacy.

Robustness Metrics: A dynamic metric is robust if a "small" change in program behavior results in a correspondingly small change in the measured value. It is difficult to precisely determine what a "small" change is; it is also difficult to achieve robustness when quantifying program behavior. As illustrated in [12], basic problems encountered when trying to accurately and reasonably measure dynamic properties of a program are discussed.

Complexity Measure: The complexity of a data set relative to a theory is defined as the length of the shortest program necessary to reconstruct it from that theory. The total information required to represent the data $I(D)$, is the amount of information necessary to specify the theory $I(T)$, plus the information necessary to specify the data given the theory $I(D/T)$ can be written as equation (7).

This formulation encapsulates a trade off between a complex, over-fitted, theory where $I(T)$ is large and

$$I(D) = I(T) + I(D/T) \quad (7)$$

$I(D/T)$ small, and a simple, over-general, where $I(T)$ is small and $I(D/T)$ large. Aim is to compare different theories on the same data [13].

Naturalness and Friendliness Metrics: The assumption of naturalness means that unless a more detailed explanation exists, all conceivable terms in the effective action that preserve the required symmetries should appear in this effective action with natural coefficients. The friendliness feature makes the software program easy to understand or easy for use. Semantic complexity can be defined using a new concept of meaning automata [14].

Metrics for Content Measurement: Content refers to the subject matter of a written work. Content can be measured by measuring consistency, correctness, validity

as well as the completeness of a program. All these facets are measured by the following metrics [15].

Consistency Metrics: The relative number of 'contradict relationships' out of all relationships can indicate the degree of inconsistency as defined in equation (8). RCC is the set of relationships that connect concepts in RS.

$$Consistency = \frac{\left|\left\{ x \mid x \in RCC \wedge negation(contradict(x)) \right\}\right|}{\left| RCC \right|} \qquad (8)$$

Completeness Metrics: Missing requirements cause additional cost. All in ontology should be mentioned in RS ideally. Completeness can be given as equation (9) where Con is the concept and Rel is the relations.

$$Completeness = \frac{\left|\left\{ x \mid x \in Con \cup \mathrm{Rel} \wedge exists y : \mathrm{Re}qItem.\, x \in F_{\mathrm{int}}(y) \right\}\right|}{\left| Con \cup \mathrm{Rel} \right|} \qquad (9)$$

Correctness Metrics: Domain ontology is a guideline to decide requirement's naturally required, so all statements in RS should correspond to elements in the ontology in domain. Correctness can be defined as given in equation (10).

$$correctness = \frac{\left|\left\{ x \mid x \in \mathrm{Re}\, qItem \wedge F_{\mathrm{int}}(x) \neq \Phi \right\}\right|}{\left| \mathrm{Re}\, qItem \right|} \qquad (10)$$

Validity Metrics: When a requirement statement is mapped onto several elements that are not semantically related, the statement is regarded as an ambiguous one. Validity can be defined as given in equation (11).

$$validity = \frac{\left|\left\{ x \mid x \in \mathrm{Re}\ qItem \wedge F_{\mathrm{int}}(x) \subseteq Clo \right\}\right|}{\left| \mathrm{Re}\ qItem \right|} \qquad (11)$$

Where 'Clo' is the transitive closure of relationships except contradict and antonym.

Metrics for Coupling and Cohesion Measurement: In an empirical study [16], it is shown that metrics values indicating lower modularity were in fact associated with lower productivity, greater rework effort and greater design effort in the context of object-oriented systems.

Metrics for Frames and Rules: These metrics are quantitative models of the fundamental concepts coupling and cohesion, which represent important aspects of the modularity of software systems [17]. In this sense, we show the theoretical soundness of our coupling and cohesion metrics. Doing so, we also show how to assess the theoretical soundness of newly proposed metrics. Coupling and cohesion can be explained in terms of elements (like modules, classes or frames) that are linked in some way [18, 19]. The metric $DC_pF(f)$ (Degree of Coupling of Frame) is defined as the number of edges from the vertices corresponding to the slots of the frame to the vertices corresponding to the slots of other frames. Then the metric for cohesion, ChF (Cohesion of Frame) is defined as the number of actual edges divided by the number of possible edges in G_f as shown in [20] and given by the equation (12):

$$ChF(f) = \begin{cases} \dfrac{|E_f|}{2\binom{s}{2}} = \dfrac{|E_f|}{s(s-1)} & if\ s > 1\ and\ 1\ if\ s \leq 1. \end{cases} \qquad (12)$$

Metrics for Size Measurement: Size can be defined as character, value, or status with reference to relative importance or the capacity to meet given requirements or the actual state of affairs.

Metrics for Estimating Size: A model to size Prolog programs is developed using the concepts of an "operator" and an "operand" from software science. By separating the operator counts into counts of Prolog predicates and non-predicate operators, and the operand count into counts of instantiated and uninstantiated variables [21], it is possible to deduce a model which is as accurate as any model produced for sizing programs. By banding the model's parameters, it is also shown that the complexity of the model can be reduced whilst retaining its high level of accuracy. Dividing system size by PM effort gives a measure of productivity and a useful indicator of the feasibility of developing a system and reliability of this type of system means these metrics lack in measuring all the basic mechanisms of AI systems paradigm and moreover there is no correlation revealed between the basic mechanisms and the quality attributes. One impediment to the growing use of AI-based software is the lack of metrics suite for determining various characteristics as well as effective methods for ensuring the quality and reliability of this type of system. In particular, the nature of AI applications often confounds conventional software requirements specification techniques, while the characteristic architectures of AI-based systems confound conventional verification and testing techniques. This has lead to a great deal of interest in studying the metrics suite for AI-based systems and then with further refinement of metrics takes place in order to extract its design factors from the design of AI-based software. From the refinement of metrics suite, the main work is how to evaluate the quality of the AI programs which are already considered as complex programs and it turns to be more difficult when there were no criteria on the basis of which one can evaluate quality of these systems. It is clear from the survey of the past

literature that till now there was no strategy based on metrics which can evaluate the quality in an either way. It is also written that as AI programs have large complexity, they are usually degraded programs in relation to quality. So it's very difficult to measure quality traits of such AI programs. The work tends to evaluate the quality of software by means of metrics which are suited to quantify design characteristics in the different stages of software development. Results of empirical validation studies are reported to show the analytical and predictive power of metrics [22].

After reviewing the different metrics under design descriptor category, it is clear that metrics can be used to evaluate the quality of AI based programs (prolog programs) in both ways:

- Qualitatively
- Quantitatively

Qualitative research is one of the two major approaches to research methodology in social sciences. Qualitative research involves an in depth understanding of human behavior and the reasons that govern human behavior. Unlike quantitative research, qualitative research relies on reasons behind various aspects of behavior. Simply put, it investigates the why and how of decision making, as compared to what, where, and when of quantitative research. Hence, the need is for smaller but focused samples rather than large random samples, which qualitative research categorizes data into patterns as the primary basis for organizing and reporting results. We can measure quality both using qualitative as well as quantitative metrics. So firstly, the need is to extract the design descriptors from the design of AI based systems. In particular, the nature of AI applications often confounds conventional software requirements specification techniques, while the characteristic architectures of AI-based systems confound conventional verification and testing techniques. This has lead to a great deal of interest in studying the metrics suite for AI-based systems and then with further refinement of metrics takes place in order to extract its quality factors from the design of AI-based software. From the literature survey, we can categorize these metrics as shown.

Qualitative Metrics: - These metrics make use of characteristics in order to evaluate the quality traits of the programs taken under consideration. These metrics considers those facets of the programs which governs the human behavior like the friendliness and naturalness of the program. These metrics used to measure the extent of the interaction between the program and the human behavior.

The qualitative metrics include the following metrics: -

- Coverage Metrics: - Data Flow Testing, Program Fault Testing, Syntax Testing
- Granularity Metrics: - Checking Prolog Policies
- Extent Metrics: - Volume Based Metrics
- Correctness Metrics: - Using Complementary Code to resolve External Dependencies
- Robustness Metrics
- Naturalness and Friendliness Metrics
- Effectiveness Metrics: - Complexity Measurement
- Efficiency Metrics: - Dialogue Structure
- Metrics for Content Measurement: - Consistency Metrics, Completeness Metrics, Correctness Metrics, Validity Metrics

These all are the qualitative metrics based on the traits which determine the human response towards the program execution. These metrics can be used to evaluate the quality of AI programs in terms of the human behavior towards the program behavior and upon their interaction with each other.

Quantitative Metrics: - Software systems and in particular also Artificial Intelligence-based systems become increasingly large and complex to understand. In response to this challenge, software engineering has a long tradition of advocating modularity. For measuring certain important aspects of modularity, coupling and cohesion metrics have been developed. For AI based systems, it presents the core of the first metrics suite, its coupling and cohesion metrics. These metrics measure modularity in terms of the relations induced between facts through their common references in rules. Study will show the soundness of these metrics according to theory and report on their usefulness in practice. As a consequence, we propose using our metrics in order to improve Quality of AI- based systems development, and developing other important metrics and assessing their theoretical soundness along these lines. So the quality of something depends on the criteria being applied to it. Something might be good because it is useful, because it is beautiful, or simply because it exists.

Quantitative Metrics can be categorized as:

- Granularity Metrics: - Time Complexity Metrics
- Extent Metrics: - Graphic Token Based Metrics
- Correctness Metrics: - Measuring Function Points from Specifications
- Scope Metrics: - Interface Size Metrics
- Metrics for Coupling and Cohesion Measurement: - Metrics for Frames and Rules

- Metrics for Size Measurement: - Metrics for estimating Size

These metrics are used to measure the quantifiable descriptors which then are used to evaluate quality of the programs taken under consideration. Depending on the values of these metrics, quality can be evaluated. One thing is that we cannot measure the exact value or approximate value of the quality. But this implemented strategy yet evaluates the quality based on the metrics values only. There was no existing strategy in the past for evaluating the quality based on Metrics Suite or any other criteria

2.2 Rule Specification

It is clear from the survey reports that the AI programs are generally of low quality. All the design facets cannot agree with each other for the quality of the program to be high. These rules are used to specify the value for each metrics and for the combination of these metrics values define the value for the quality measure. If there are values for the metrics for which no rule is specified, then they are considered to be laying near the boundary cases just above and just below. The design descriptors that are determined using the Metrics Suite Framework Engine are: -

- Number of operators and operands
- Program length (Halstead measurements)
- Degree of coupling and cohesion
- Granularity
- Lines of code (size)

Table 1: Rule Specification for AI systems

Granular-ity	**Coupli-ng**	**Cohesio--n**	**LOC**	**Halstead Mmts**	**Quality**
Low	High /Low	Very Low	Low	Very Low	Nil
Low	High	Low	Medium	Medium	Very Low
High	High	Low	Medium	Medium	Low
High	Low	High	Medium	Medium	High
High	Low	Very High	Medium	High	Very High
High	Very Low	Very High	Medium	High	Excelle-nt

2.3 Quality Model for AI Systems

To evaluate the software quality more quantitatively and objectively, software metrics appears to be powerful and effective technology to measure the software quality. Currently metrics is the core technology of software quality evaluation [23], which objectively assigns a value to characterize certain specific quality attribute. There exists difference between those quality attributes we really care about (external attributes) and what can be directly measured from the software (internal attributes).

This quality model comprises several works which focus on the problem of what a KBS evaluation method should measure in order to determine the quality. In order to allow comparisons among different proposals, we have grouped evaluation criteria into four broad classes:

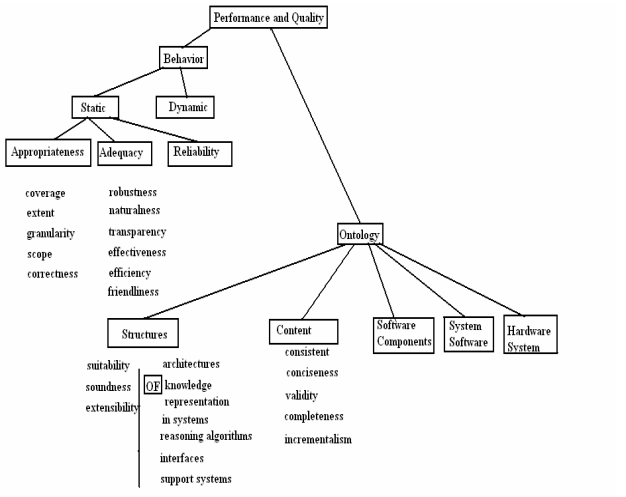

Fig. 1 Performance and Quality Evaluation Model

Validity: concerning correctness, quality, accuracy, and completeness of responses (solutions, decisions, advice);

Usability: concerning quality of human-computer interaction, understandability, explanation facilities;

Reliability: including hardware and software reliability, robustness, sensitivity;

Effectiveness: including cost-effectiveness, efficiency, development time and cost, maintainability and extensibility.

Defining and Measuring Performance and Quality (P & Q): An appropriate definition of P&Q can be obtained through decomposition, that is behavior and ontology can be decomposed into independent components, using a hierarchical classification method. Components thus obtained can be decomposed into finer grained components.

Behavior: Behavior of a KBS includes two main components: Static behavior, which concerns the behavior of a KBS over a time interval during which its ontology is fixed; Dynamic behavior, which concerns the behavior of a KBS over a time interval during which its ontology is changed. So, Static behavior concerns how a KBS responds to the inputs it receives during normal operation. Dynamic behavior concerns how a KBS reacts to the modifications performed to its structure and knowledge base by the project or maintenance team during its development life.

Ontology: Ontology of a KBS includes five main components, Structure, which refers to the architecture, knowledge representation structures, reasoning algorithms, interfaces, and support systems which are designed to implement a KBS. Content, which denotes the knowledge actually represented and stored in the knowledge base of a KBS. System software on which the KBS is implemented; Hardware system on which the KBS is installed.

Relation between Behavior and Ontology: On the basis of the taxonomy of components introduced previously, we can now analyze in greater detail the relations existing between ontology and behavior of a KBS. Ontology is in a sense the necessary condition, or in other words, the efficient cause of behavior. In fact, the ontology is purposely designed in such a way that it can produce the desired behavior, and thus meet the specifications. The elementary components of P&Q may be partitioned into two classes, namely: behavioral components (BC) and ontological components (OC), defined as the leaves of the P&Q taxonomy descending from behavior and ontology, respectively. While we may assume that components in each one of these two classes are independent from each other, there is, of course, a causal dependence between ontological.

3. Results and Discussion

The implementation of the system is done in Visual .NET and the following objectives are achieved:

- Refinement of design metrics,
- Framework for generating design for the AI based Systems,
- Determining quality factors and
- Measuring software systems quality.

In our active user interface, the framework engine works as analyzer which is used to analyze the design facets of the programs under consideration. It tends to evaluate the effectiveness of this approach as previously there was no such model or strategy which can be used to measure quality on the basis of metrics suite. There were many problems in measuring the quality as these AI programs are very complex in nature and their complexity increases as the size of the program increases. It is also clear from the survey reports that these AI programs are generally of low quality. There were no prolog programs that are of high quality because if for quality to be high, all the design descriptors should be high and that is not possible in a single prolog program. All the design facets cannot agree with each other for the quality of the program to be high.

Samples of the outputs generated upon application of metrics to the prolog code are shown as below.

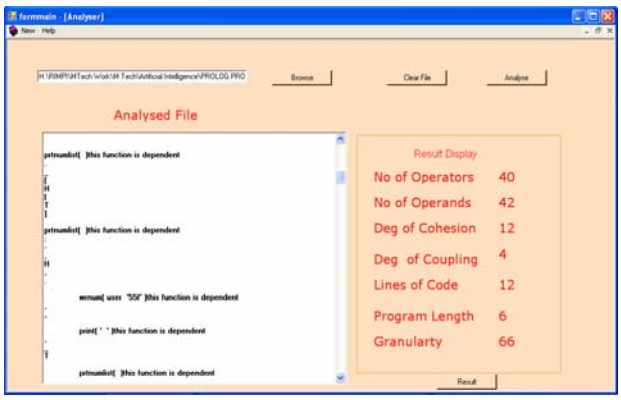

Fig. 2 Snapshot of Analysis of Prolog Code 1

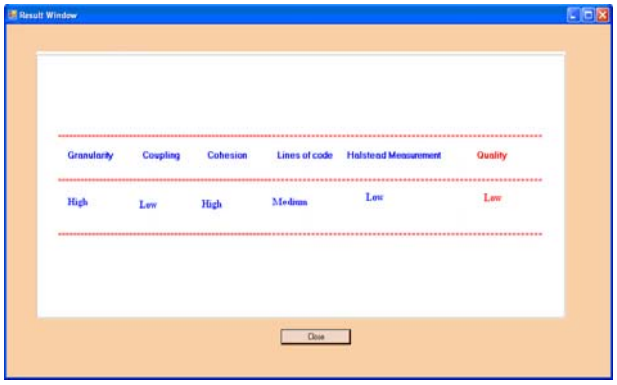

Fig. 3 Quality Evaluation of Prolog Code 1

Fig. 4 Snapshot of Analysis of Prolog Code 2

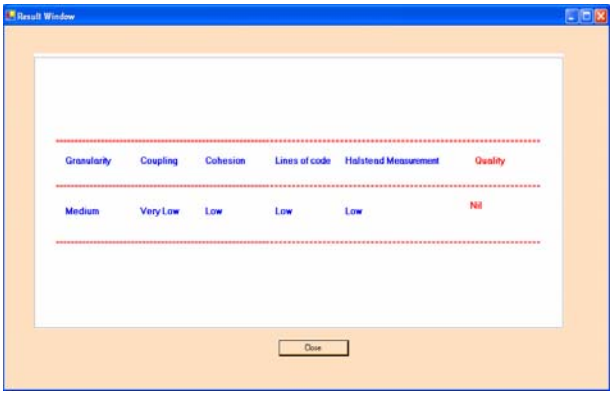

Fig. 5 Quality Evaluation of Prolog Code 2

Study will show the soundness of these metrics according to theory and report on their usefulness in practice. As a consequence, we propose using our metrics in order to improve quality of AI- based systems development, and developing other important metrics and assessing their theoretical soundness along these lines. The above shown are the screenshots of analyzing the prolog programs and evaluating the quality of prolog code depending on the values of design descriptors. Following table shows the results of analyzing different prolog codes and their quality evaluation in a tabular form.

Table 2. Results of analyzing the prolog programs using the metrics suite

Design Descriptors (Metrics)	Prolog pgm 1	Prolog pgm 2	Prolog pgm 3	Prolog pgm 4
Number of operators	9	5	32	40
Number of operands	11	6	41	42
Degree of Cohesion	4	3	20	12
Degree of Coupling	2	1	0	4
Lines of Code	3	8	12	12
Program Length	3	3	5	6
Granularity	14	6	58	66
Quality	Nil	Nil	High	Low

The question is how much autonomous we need in doing the job and how we can validate the new knowledge before committing it to the existing knowledge without causing any conflicting. Most of the approaches, although proposing interesting new ideas and methods, lack a sound foundation and are affected by several practical limitations. In particular: all knowledge-base oriented approaches are partial: in fact, knowledge-base checking is only one aspect, though very important, of AI evaluation; approaches based on evaluation criteria are weak for at least two reasons: i) the lists of evaluation criteria proposed lack precise definitions, and therefore, are often vague and poorly organized, ii) the problem of how evaluation criteria should be actually applied to real AI's is not dealt with adequately; approaches based on evaluation methods, although very interesting for the practical issues they face, are mostly based only on empiric principles and lack generality, being often oriented towards very specific application domains.

4. Conclusion and Future Scope

In this work entitled as "Evaluating Quality of AI- Based Systems", I have presented a simple approach to evaluate the quality of AI based systems (prolog programs) using the metrics suite framework. I have started by recapitulating the experiences with language dependent software metrics and have then abstracted from that context to come up with a general way to create metrics suite independently. I have reported on some experiences regarding the implementation of a metrics suite for analyzing the prolog programs. The benefits of this approach are the increased extensibility, i.e., one can introduce new metrics from arbitrary contexts which provide us with a standard metrics suite without having to implement new metrics each time such a new context is introduced. The objective of this approach are that currently software metrics suite designed can be used to evaluate the quality of the prolog programs and it is up to the best level in the area of artificial intelligence as there exists no such implementation or strategy that makes use of metrics in order to measure the quality of AI systems. Certain metrics tend to be very specialized and are thus difficult to define in a generic way. Another problem is that for some metrics, there is still no consensus about what is the best way to define them. For these reasons we did not consider efficiency metrics, content measurement metrics and effectiveness metrics. Furthermore, as, due to this research in reengineering, we come from a more pragmatic side, it is not solely the definition of the metrics which interest us. Indeed, it is the creation of concrete metrics, which we can then use as a quality measure, this interests us. In this context other problems arise, whose solutions lie mainly at the implementation level: what are the exact definitions of those metrics, how many can we generate and when does it make sense to stop generating metrics? Are the generated metrics actually usable and which ones do make sense in the given contexts? We plan to answer some of these questions in our future work. The critical literature survey reported in the previous sections allows us to identify a basic set of requirements for an ideal KBS evaluation approach. This includes: a precise definition of AI evaluation and related concepts and terminology; a sound foundation of the concept of AI evaluation, where it is clearly stated i) what characteristics of an AI are to be evaluated, and ii) what to evaluate them against; a general methodology, defined according to the concept of AI evaluation adopted, specifying what to measure and how; an effective procedure to apply the general methodology to real cases. In future, research work can be extended with determining quality factors or attributes, deciding and refining of metrics suite for AI systems in different contexts and framework for evaluating quality factors in detail for the AI based systems.

In simple words, we can explain the software quality attributes as follows:-

- Functionality: Does the software satisfy stated needs?
- Reliability: How often does the software fail?
- Usability: How easy is the software to use?
- Efficiency: How good is the performance of the software?
- Maintainability: How easy is the software to repair?
- Portability: How easy is the software to transport?

References

[1] Howden, W.E. (1985) "The Theory and Practice of Foundation Testing", IEEE Transactions on Software Engineering, vol. 2, no. 5, Page(s):6 – 17.

[2] Huang, C.Y. and Lin C.T. (2006) "Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag", IEEE Transactions on Reliability, vol. 55, no. 3, Page(s):436 – 450.

[3] Gutjahr, W.J. (1999) "Partition Testing vs. Random Testing: The Influence of Uncertainty", IEEE transactions on Software Engineering, vol. 25, no. 5, Page(s):661 – 674.

[4] Henry, S. and Kafura, D. (1981) "Software Structure Metrics based on Information Flow", IEEE Transactions on Software Engineering, vol. 7, no. 5, Page(s):510–518.

[5] Ranganathan, A. (2000) "A Task Framework for Autonomic Ubiquitous Computing", IIT Madras.

[6] Nickerson, J.V. (1994) "Visual Programming: Limits of Graphic Representation", IEEE Symposium on Visual Languages, IEEE Computer Society Press: Los Alamitos, CA, St. Louis, Missouri, Page(s):178-179.

[7] Halstead, M. (1977) "Elements of Software Science, Operating, and Programming Systems", Elsevier, volume 7.

[8] Albrecht, A. and Gaffney, J. (1983) "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, vol. 9, no. 6, Page(s): 639-648.

[9] Fuggetta, C., Ghezzi, D., Mandrioli M. and Morzenti, A. (1988) "VLP: A Visual Language for Prototyping", IEEE Workshop on Languages for Automation, College Park, MD, August 1988.

[10] M. Stefik, "Introduction to Knowledge Systems", Morgan Kaufmann Publications, San Francisco, CA, (USA), 1995.

[11] Boxall, A.S. and Araban, S. (1998) "Interface Metrics for Reusability Analysis of Components".

[12] Dufour, B., Hendren, L. and Verbrugge C. (2003) "Problems in Objectively Quantifying Benchmarks using Dynamic Metrics", Sable Technical Report Number: 2003-2006, School of Computer Science, Sable Research Group, McGill University.

[13] Cleary, J.G., Legg, S. and Witten, I.H. (1993) "AN MDL Estimate of Significance of Rules".
[14] Zadrozny, W. (1995) "Measuring Semantic Complexity", IBM Research.
[15] Kaiya, H. and Saeki, M. (2000) "Ontology Based Requirement Analysis: Lightweight Semantic Processing Approach", Japan.
[16] Chidamber, S., Darcy, D. and Kemerer, C. (1998) "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Transactions on Software Engineering, vol. 24, no. 8, Page(s): 629-639.
[17] Briand, L.C., Morasca, S. and Basili V.R. (1997) "Response to: Comments on "Property-based Software Engineering Measurement": Refining the Additivity Properties", IEEE Transactions on Software Engineering, vol. 23, no. 3, Page(s): 196-197.
[18] Briand, L.C. and Wust, J.K. (2001) "The Impact of Design Properties on Development Cost in Object-Oriented Systems", In Proceedings of IEEE Metrics'2001, IEEE Computer Society Press, Los Alamitos, Calif.
[19] Briand, L.C., Wust, J.K., Daly, J.W. and Porter, V. (2000) "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", J. System Software, vol. 51, Page(s): 245-273.
[20] Kramer, S. (Technische Univ.) and Kaindal, H. (Vienna Univ. of Technology) (2004) "Coupling and Cohesion Metrics for Knowledge-Based Systems Using Frames and Rules".
[21] Myers, M. and Kaposi, A. (1991) "Modeling and Measurement of Prolog Data", Software Engineering Journal, vol. 6, no. 6, Page(s): 413-434.
[22] Moores, T.T. (2001) "A Model to Size the Development of Prolog Programs", Department of Information Systems, City University of Hong Kong.
[23] Guida, G. and Mauri, G. (1993) "Evaluating Performance and Quality of Knowledge-Based Systems: Foundation and Methodology", IEEE Transactions on Knowledge and Data Engineering, vol.5, no. 2, Page(s): 234-274.