Cognitive Complexity Metrics and its Impact on Software Reliability Based on Cognitive Software Development Model

Dharmender Singh Kushwaha and A.K.Misra
Department of Computer Science and Engineering
Moti Lal Nehru National Institute Of Technology
Allahabad, India.
Email:dharkush@yahoo.com,arun_kmisra@hotmail.com

Abstract
Software metrics provide a quantitative basis for the development and validation of models of software development process. Information gained from metrics is used in managing the development process in order to improve the reliability and quality of software product. The software metric is used to estimate various parameters of software development lifecycle such as cost, schedule productivity, quality and reliability. In this paper, an attempt has been made to frame the cognitive complexity metrics that will aid in increasing the reliability of software product being developed during the development lifecycle.

Keywords
Cognitive Software Development Model,Cognitive Documentation Complexity, Metacognition, Cognitive Software Inspection Process, Cognitive Requirement Engineering.

1. Introduction

Metrics are a useful means for monitoring progress, attaining more accurate estimation of milestones and developing a software system that contains minimal faults thus improving the quality. Measures are necessary to identify weaknesses of the development process. They also prompt the necessary corrective activities and enable us to monitor the results. Hence they act as feedback mechanism that plays a vital role in the improvement of the software development process. There is an urgent need of software metrics to monitor the software development process for improving the overall quality of the software . Since complexity metrics are a significant and determinant factor of a systems success or failure, there is always a higher risk involved when the complexity measurement is ignored. Software metrics have been used for over three decades to assess or predict properties of software systems, but success has been limited by factors such as lack of sound models, the difficulty of conducting controlled repeatable experiments in educational or commercial context, and the inability to compare data obtained by different researchers.

It is hypothesized that an overly complex code (i.e. an unstructured code with low cohesion) will be difficult to maintain and is likely to be unreliable. In order to create software, our design decisions, cognition, metacognition learning process and problem comprehensibility should be able to guide us to create software such that overall complexity is reduced. The most efficient way to deal with developing reliable software for large systems is by creating smaller modules. Weyuker's [7] property 5 $(\forall P)(\forall Q)(|P| \leq |P;Q|$ and $|Q| \leq |P;Q|)$ just aims to achieve this. The implication of this property is that as the size of program segment is increased, its complexity should also increase. Hence the divide and conquers technique which relies on decomposing the original problem into sub-problems with well defined interactions will lead to a structured design that will make the software reliable and maintainable.

Cognitive aspects focus on the ease of understanding or the property of comprehension. Comprehension is the key feature that distinguishes any entity as being complex or simple. Comprehensibility of a problem helps in efficient design solution and improvement of software product quality. Thus property of comprehensibility can be used in all the different phases of software engineering.

2. Cognitive Software Development Model

Kushwaha and Misra [5] have proposed Cognitive Software Development Model, with emphasis on cognitive aspects. This cognitive model of software development is based on the following broad phenomena:

- Cognitive process of requirement engineering;
- Cognitive system analysis;
- Cognitive system design;
- Assigning team members to different tasks based on the cognitive phenomena;
- Cognitive approach to software inspection and testing;
- Cognitive documentation style.

The cognitive software development model of software development is illustrated in fig.1
It can be inferred from the cognitive software development model that the phases that also contribute to software reliability are:

1. Cognitive phenomena based team member selection;
2. Human centric requirement engineering;
3. Cognitive software inspection, a part of testing;
4. Documentation based on cognitive phenomena and
5. Validation by Metrics.

Hence the following sections of our paper will elaborate on how above-mentioned factors and metrics will help in development of reliable software systems.

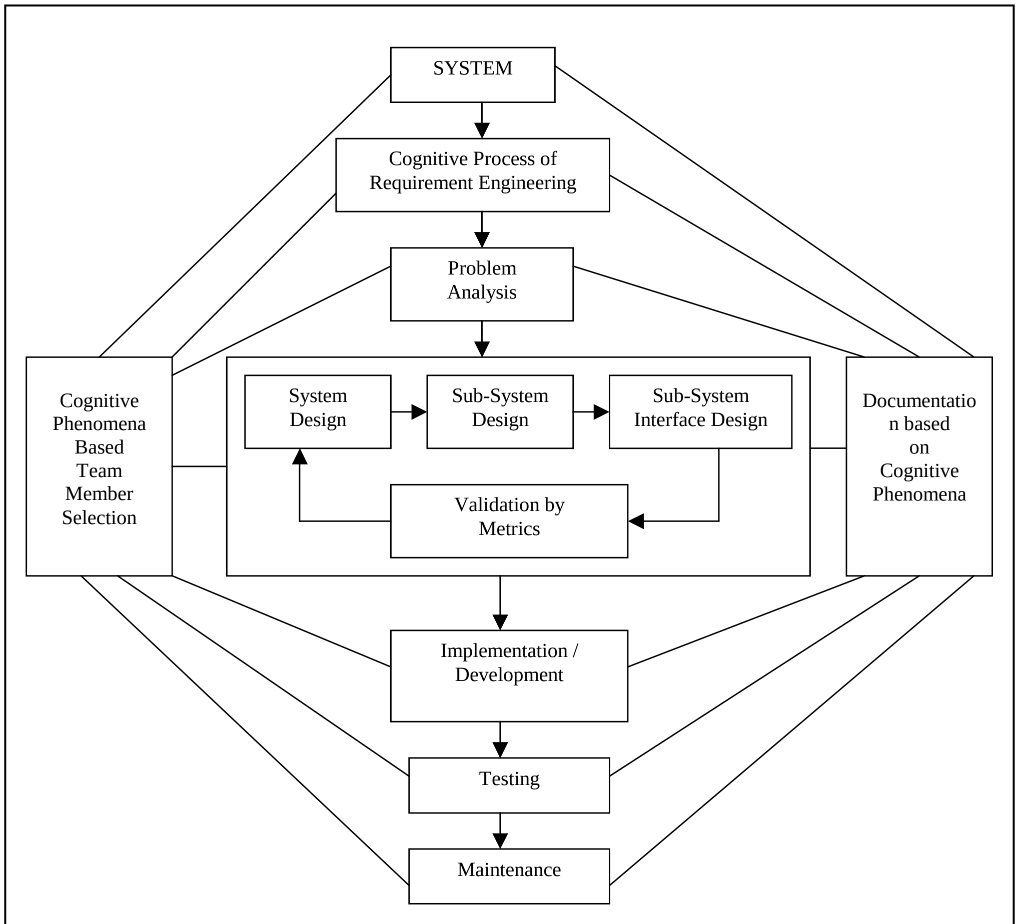
SYSTEM
Cognitive Process of Requirement Engineering
Problem Analysis
Cognitive Phenomena Based Team Member Selection
System Design
Sub-System Design
Sub-System Interface Design
Validation by Metrics
Documentation based on Cognitive Phenomena
Implementation / Development
Testing
Maintenance

Figure 1:Cognitive Software development moel

3. Reliability

The basic requirement for software system is correctness. A program is correct if the output satisfies the output requirements for every input specified by the input requirements. There are active and passive approaches to achieve program correctness. The active approach takes the form of program correctness proofs. Passive approach takes the form of traditional testing and debugging. Since testing can only indicate the presence of bugs, not their absence, hence it is not possible to estimate the number of bugs remaining in the software system. Hence testing cannot guarantee program correctness. Based on the above reasoning the reliability of software can be defined as the probability that the subsequent execution and invocation of the program is also correct.
Our approach to reliability is based on the theory of "Prevention is better than cure". To achieve this, we propose our metrics that are constructive, analytic and cognitive in approach. We begin with human centric approach of requirement engineering in the next section.

4. Cognitive Process of Requirement Engineering

Until now, the requirement engineering has focused on the following areas:

- Abstract functional requirements;
- Non-functional requirements;
- System properties such as availability and performance;
- System and environment requirements.

The areas that have a major impact in requirement engineering but often ignored are:

- They do not distinguish between the client and the end user. Since client is the one who first comes in contact with the software engineer / software practitioner representing the vendor and the end user is the one who has to use the software. The perspectives of both shall be different and should be gathered carefully.
- Client is not able to describe and establish the cognition and psychological level of the end users of the proposed system
- The requirement analysis is not performed from the users point of view and as such system oriented requirement analysis is performed
- The requirement-gathering phase doesn't distinguish between low level details and cognition characteristics of these details.
- System oriented design fails to uncover the need for domain specific training of the end user or change in human organization.

The cognitive requirement-engineering model is proposed to provide a holistic description of cognitive characteristics of the system under study (including the cognitive characteristics of the end user). Cognitive characteristics capture information such as user preferences. These also help in understanding the cognitive phenomena of the user, which will help in streamlining further interactions with the users in producing reliable software. We propose a model to describe the cognitive characteristics of the different user as depicted in fig. 2.
The requirement analyst has the responsibility of analyzing the requirements and views. The requirement gathering, if properly

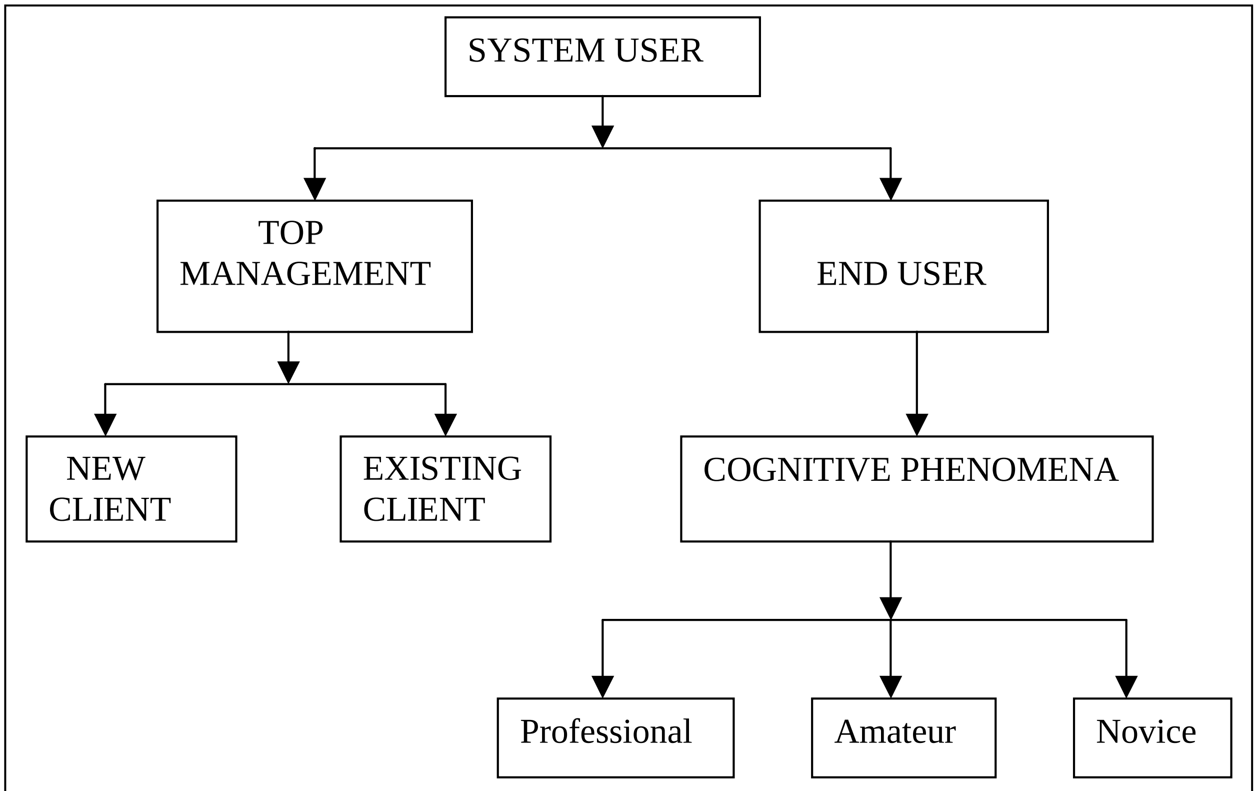

Figure 2: Cognitive characteristics of user

classified depending on the kind of user interacted with, will provide for a total system view, thereby reducing the number of interactions and the rework. These views can be organized as a tuple [R_{TP}, R_{PROF}, R_{AMAT}, R_{NOV}] representing requirements of top management, professional, amateur or novice user. Once the analyst is able to categorize the viewpoint of the users at different hierarchies with varying cognitive phenomena, the overall system developed will be in total synch not only with the explicitly stated requirements but also with the implicit ones. Also the information domain of the system to be developed must be clearly understood.

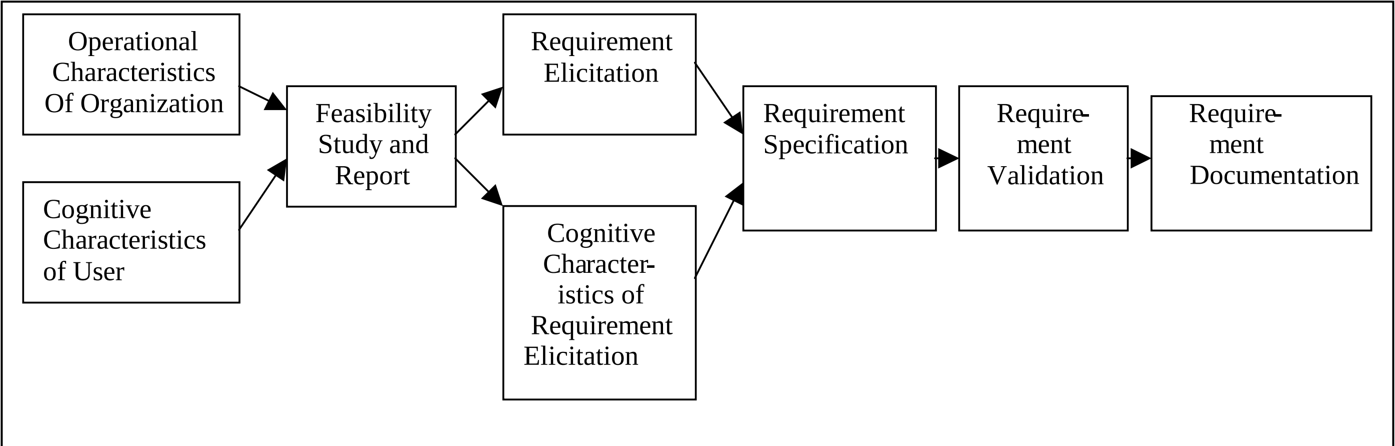
Operational Characteristics Of Organization
Cognitive Characteristics of User
Feasibility Study and Report
Requirement Elicitation
Cognitive Character-istics of Requirement Elicitation
Requirement Specification
Require-ment Validation
Require-ment Documentation

Figure 3 : Cognitive Requirement Engineering Model

The requirement-engineering model is depicted in fig. 3. The person who is the catalyst behind the request for the software will play a key role in extending the need analysis and hence carrying out the requirement analysis. Once proper requirement engineering has been carried out, we can safely proceed to problem analysis. This will reduce the amount of rework.

5. Cognitive Software Inspection and Team Member Selection

As the software system evolves and grows larger, the effort and cost needed by verification process grows astronomically. Dolan [6] reports a 30 times return on investment for every hour devoted to the software inspection. Software inspection process is still far from optimal, and one area that has seen limited research is the impact of team member selection on it.

- Lack of research on team member selection
- Incorrect selection mechanism of team members
- Team members sharing / possessing similar view point to a particular problem

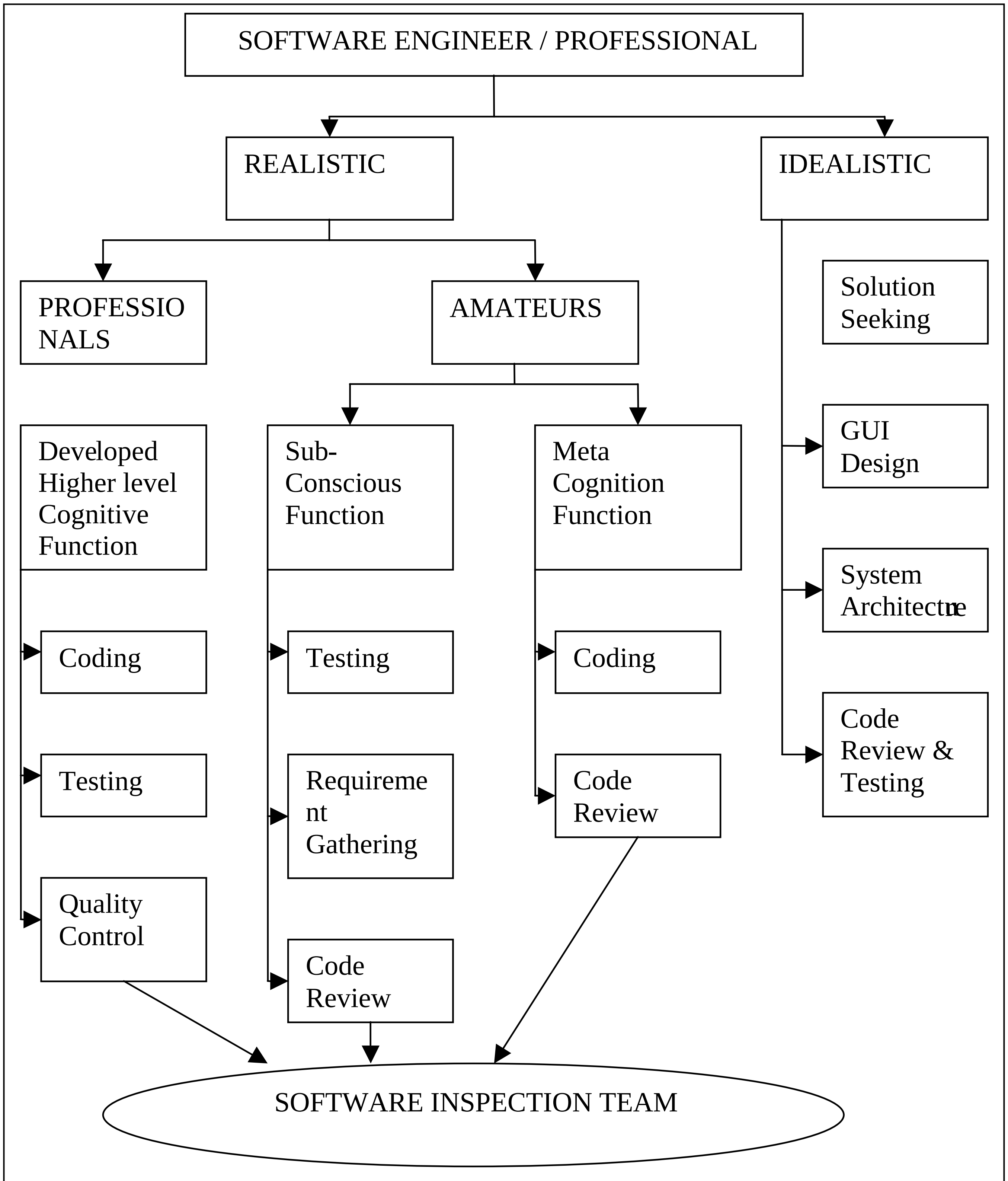

Figure 4: Cognitive phenomena based team member selection mechanism

- It is a cognitive phenomenon, but unfortunately, it has been seen in terms of technical maturity of team member performing the code inspection.

To overcome the above bottlenecks, a cognitive model of team selection procedure for software inspection is proposed based on the functionalities best performed depending on the cognitive activity level of team members.

Wang [10], proposed cognitive phenomena, which classifies cognitive functions as sub-conscious, meta-cognitive and higher cognitive. Wang [8] described two types of programmers – realistic and idealistic. It is important issue to contrast and analyze the cognitive levels of the team members. It is agreed among researchers that increasing diversity among team members is the key to increasing the software inspection effectiveness. Hence creating inspection team with a combination of members from different cognitive levels will increase the effectiveness of code review, thereby reducing the testing effort.

The difference between professionals and amateurs is whether their knowledge and skills are wired or temporary programmed in the brain [9]. For e.g. Professional software engineers possess wired skills in programming, and with a global view on software development. They focus not only on required functions, but also on exceptions-handling and fault tolerance. However, amateur programmers possess ad-hoc programming knowledge, eager to try what is directly required, and tend to focus on details without a global and systematic view. Hence software inspection and testing team should include members from this amateur category also, since they have the quality to have a narrow focus as illustrated in fig. 4. The amateurs try to deal with individual concepts before taking on the new ones are also candidate for testing and debugging. This will ascertain that the software inspection so carried out increases the reliability of software product.

6. Cognitive Documentation Complexity (CDC)

Very little attention has been paid to the comprehensibility of various documents created during software development lifecycle. If a novice is able to understand the various documents, then we are sure of creating a reliable and quality artifact. Cognitive documentation complexity of a class can be measured in terms of the cognitive phenomena and the associated weight. There are numerous documentation types that will speak about the level of useful information provided by the documentation. Even today, header information, comments and use of good identifier names are considered to be the quality factors of best documentation. These practices, do not count on the comprehensibility and the cognitive phenomenon of the mind that assists the software developer in reducing the comprehension effort and improving the coding standard.
If the documentation provides reasonable amount of useful information about the class to a novice, average and expert software practitioner, it implies that ready understandability was present in the documentation of the class and is termed to be of high quality. On the contrary, if only expert practitioner understands the documentation, then the documentation quality is termed to be low. This also implies that sub-conscious cognitive functions (the one that are not wired) are not enough in comprehending the documentation. The classification of cognitive phenomenon is as described by Wang [10]. This is summarized in the table below.

S.No	Cognitive Phenomena associated with Documentation	Quality of Documentation
1.	Sub – Conscious Cognitive function	High-Quality
2.	Meta – Cognitive function	Average-Quality
3.	Higher Cognitive functions	Low-Quality

Table 1: Quality of Documentation Based on Cognitive Phenomena

High quality documents are very useful and enable us to:

- Enhance comprehensibility of software product
- Reduce maintenance cost
- Effectively exploit the system
- Reduce re-engineering cost and effort.

7. Cognitive Conceptual Complexity of Class / Module

The syntactic metrics measure the complexity of the software. It is the details of implementation that determines the comprehension complexity [1, 2]. We also need some metrics to measure the psychological complexity that measures the difficulty of understanding of the software module. This psychological complexity is based on the number of distinct key concepts in the class or module and is defined as

$$CCCC = \sum_{i=1}^{m} [(\text{No. of distinct Cognitive Concepts}) * (\text{Weight of Concept})]_i$$

Where 'm' is the number of distinct concepts in the class or module.

The weighing factor of cognitive concepts is based on the classification of cognitive phenomenon as described by Wang [8], is as follows:

S.No	Cognitive Phenomena	Weight
1.	Sub – Conscious Cognitive function	1
2.	Meta – Cognitive function	2
3.	Higher Cognitive functions	3

Table 2: Weights Based on Cognitive Phenomena

Higher weight indicates that greater amount of comprehension effort is required in understanding the software module under consideration. If there exists a distinct concept but is not understood by the practitioner (because he may be novice or semi-skilled), it is the subconscious life function that will guide him to identify it. Hence higher weight is associated with it. Higher cognitive functions will require lower comprehension effort and hence lower weight associated with it.

The mind is an artifact model of oneself and a thinking engine. The mind, as a virtual model of a person in the brain, is partially programmed and partially wired. The former is evolved for the flexibility of the life functions while the latter is formed for the efficiency of frequently conducted activities. The complexity of the class can be calculated by using the CICM metric that is a robust cognitive complexity measure [3, 4].

8. Conclusion

This paper is based on the cognitive software development model. It has made an attempt to emphasize the importance of cognitive metrics and its impact in achieving reliable software development. It also identifies those areas that are vital to reliable software development process but have been either ignored or given lower degree of importance by the researcher community in the past. In future, we shall work to propose cognitive complexity metrics for all the phases of cognitive software development model in order to produce software that is not only reliable but is of highest quality.

References

[1] Kushwaha, D.S. and Misra, A.K.,: A Complexity Measure Based on Information Contained in the Software, *5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS 2006)*, Madrid, Spain, To Appear, Feb. 2006.
[2] Kushwaha, D.S. and Misra, A.K.: Cognitive Complexity Measure of Object-Oriented Software – A Practitioners Approach, *5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS 2006)*, Madrid, Spain, To Appear. Feb. 2006.
[3] Kushwaha D.S.and.Misra A.K: “Robustness Analysis of Cognitive Information Complexity Measure using Weyuker Properties”, *ACM SIGSOFT Software Engineering Notes*, Vol. 31, No. 7, January 2006.
[4] Kushwaha D.S.and.Misra A.K: “Evaluating Cognitive Information Complexity Measure ”, *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)* To Appear,2006.
[5] Kushwaha, D.S. and Misra, A.K.: A Cognitive Complexity Metric Suite for Object-Oriented Software”, *WSEAS Transactions on Computers*, To Appear, Feb. 2006.
[6] E.P.Doolan, :Experience with Fagans Inspection Method, *Software Practice and Experience*. Vol. 2, No. 2, pp. 173-182, Feb. 1992.
[7] Weyuker, E.,: Evaluating software complexity measure. *IEEE Transaction on Software Engineering* Vol. 14(9): 1357-1365, september1988.
[8] Wang,Y.,: On The Cognitive Informatics Foundations of Software Engineering, *IEEE International Conference on Cognitive Informatics*, 2004.
[9] Wang,Y.,: On The Informatics Laws of Software, *IEEE International Conference on Cognitive Informatics*, 2004.
[10] Wang,Y.,: On Cognitive Informatics, Keynote Lecture, *Proceedings of IEEE International Conference on Cognitive Informatics*, 2002, pp. 34 – 42.