International Journal of Information Technology and Knowledge Management
January June 2009, Volume 2, No. 1, pp. 25-31

OBJECT-ORIENTED METRICS SUITES & COMPLEXITY: A SURVEY

Jitender Kumar Chhabra*, Ravinder Singh Bhatia* & V. P. Singh*

The research in the area of software engineering has progressed a lot during the last decade. The whole stress of development has shifted from the procedure oriented to object-oriented environment. Most of the software development nowadays is done in Java, .Net, C++ due to obvious advantages of these languages. This paradigm shift has triggered the researchers to propose new metrics for the object-oriented software. Recent trend in the object-oriented metrics has been to proposed a set of metrics in form of metrics suites, which provide a framework to software managers and programmers to control the software characteristics, especially complexity, which is the most important characteristics of the software and affects almost all internal and external attributes of software quality. The extensive knowledge of these metrics suites and complexity metrics is must for all researchers working in any area of object-oriented software engineering. This paper presents an organized survey of the object-oriented metrics suites and complexity. The last decade has seen development of a set of metrics measuring different aspects of the software design, which have been discussed in the first category. Then the researchers concentrated on specific aspects of the quality of the design like coupling, fault-proneness, cohesion, reusability, and complexity. All of these attributes of the quality have been reported to be largely dependent on the complexity. So the last section gives details about the metrics proposed in the direction of complexity of object-oriented software.

1. Introduction

A critical distinction between software engineering and other better established branches of engineering is the shortage of well-accepted measures, or metrics, of software development. The question is why is software (engineering) measurement so problematic? One answer may be, following Roche *et al*. [1], that software engineering is a highly complex process producing highly complex products. Without metrics, the tasks of planning and controlling software development and maintenance will remain stagnant in a craft-type mode, wherein greater skill is acquired only through greater experience, and such experience cannot be easily communicated to the next system for study, adoption, and further improvement. With metrics, software projects can be quantitatively described, and the methods and tools used on the projects to improve productivity and quality can be evaluated [2]. The earliest time to control the quality & productivity etc. of the software is the design phase. The fundamental reality that "you can not control what you can not measure" highlights the importance of good design metrics.

2. Traditional Versus Object-Oriented Metrics

The o-o paradigm for software development differs from the traditional procedure paradigm in many ways. That is why many authors have opposed the use of traditional metrics to o-o systems. O-O concepts and abstractions such as classes, inheritance, polymorphism, overloading, encapsulation etc. do not get addressed by traditional metrics [3-8]. Henderson-Sellers [9] noted that "the traditional and o-o paradigms differ in that the traditional paradigm requires more effort during the coding and maintenance phases than its o-o counterpart" and that "the o-o methodologies put more emphasis on the earlier stages of analysis and design", thus implying that a new set of metrics is needed to reflect those differences. Moreau and Dominick [3] pointed out that many existing metrics that have been utilized within conventional programming environments are inappropriate for evaluating o-o systems in certain circumstances. They mentioned that traditional LOC metric as an example would be a poor indicator of development complexity within o-o systems, since only a small part .of the code is likely to be unique to an object because of inheritance related reuse. Other traditional metrics such as McCabe's cyclomatic complexity [10] and Halstead's software science measures [11] also need to be recalibrated to o-o systems to be effective [12-13]. At present, it is a well-established fact that either the new metrics need to be defined along with the traditional metrics or a new set of metrics is needed to measure different aspects of o-o software.

In this paper, we present a survey of various design metrics proposed in the literature for o-o software. Although the total number of publications in the area of o-o design metrics in various national/international journals and conferences may be very large, we have tried to focus on the main developments in the direction of o-o design measurement. Two major categories of the publications are devoted to metrics suites and complexity metrics, each of which is described below in separate sections.

* National Institute of technology, Kurukshetra-136119 INDIA, *E-mail: jitenderchhabra@rediffmail.com, rsibhatia@yahoo.co.in, vpsingh72@yahoo.com*

3. METRICS SUITES

It has been pointed out that any single metric is inadequate to capture all aspects of software development process and respective product [14]. So many researchers have proposed a set of metrics capturing different aspects of o-o software. The first well known set of metrics was proposed by Morris [15], which consisted of 9 metrics named - Methods per Class, Inheritance Dependencies, Degree of Coupling between Objects, Degree of Cohesion of Objects, Object Library Effectiveness, Factoring Effectiveness, Degree of Reuse of Inheritance Methods, Average Method Complexity, and Application Granularity. These metrics provided a new direction for object-oriented (referred as o-o afterwards) software measurement, but these metrics were not tested and thus lacked in validation. Moreau & Dominick [3] suggested three metrics for o-o graphical information software – Message Vocabulary Size (MVS), Inheritance Coupling (IC), Message Domain Size (MDS). These 3 metrics needed clarifications such as what exactly is meant by sending messages or how the metrics are to be computed. Moreover the metrics were neither tested nor validated [3]. Many metrics were proposed during 1990-92 for measurement of different design characteristics of o-o software [16-18], but all of these neither offered a theoretical base or any validation attempt using empirical data.

Chidamber & Kemerer (referred as C & K afterwards) [19] proposed a preliminary set of metrics in 1991. These metrics were tested and measured by some authors [7, 20, 21], but these measurements were done on students' projects only instead of some commercial applications, and hence needed further investigations. Then came the most widely discussed metrics suite for object-oriented design [22]. C & K [22] proposed six design metrics – Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object classes (CBO), Response For a Class (RFC), & Lack of Cohesion in Methods (LCOM). The empirical data of the metrics was also provided with the help of o-o libraries of two commercial organizations and the theoretical ground was provided based on Weyuker's nine axioms [23], out of which two properties (number 7 and 9) were not satisfied by any of the proposed metrics. For these two properties, C & K concluded that these are not applicable to o-o software [22]. The non-applicability of property 9 has also been generalized and proved by Gursaran [24] with reference to inheritance metrics based on a directed graph abstraction of the inheritance structure. But their proof was found to have some discrepancies in case of some extreme circumstances [25].

Churcher and Shepperd [26] pointed out that definitions of some of the basic direct counts used in metrics of [22] are imprecise, which could have an impact on the defined metrics. Their main concern was with the number of methods in a class count, used directly in computation of WMC and indirectly in LCOM. Due to the various possible alternatives in counting the methods, the results could vary, leading to confusion. The various possibilities resulted from count of inherited methods, methods with same names but different signatures, and so on. They [26] further pointed out that effectiveness of these 6 metrics could be enhanced by providing guidelines concerning the application of these metrics to specific language and by empirical validation. In reply to these remarks, C& K [27] clarified that the methods that require additional design effort and are defined in the class should be counted and those that do not should not. Hitz & Monatzeri [28] also criticized some of the metrics of [22] and agreed with comments of [26]. Their main focus was on CBO & LCOM metrics [28]. The details of their work [28] have been discussed in coupling and cohesion metrics separately below. Gursaran [29] tried to empirically validate DIT and NOC metrics of C&K [22], and his results indicated that these two metrics did not preserve the numerical relation, which should be preserved from measurement theory perspective. In another study [30], it was shown that five out of the six C&K metrics (except LCOM) [22] were useful in predicting the class fault-proneness during the high and low level design phases of the life cycle. The metrics were found to be statistically independent and did not capture a great deal of information. The authors [30] concluded that C&K metrics proved to be better predictors than the metrics collected at the later phases of the software life cycle. Li [31] evaluated C&K metrics by using Kichenham's metric evaluation framework and found some deficiencies in these metrics. The framework proposed by Kitchenham et al [32] for metric-evaluation consisted of 5 models – unit definition, instrumentation, attribute relationship, measurement protocol, and entity population. In the unit definition model, a unit was defined for all measures including ratio, scale, nominal and ordinal. An instrumentation model determined the method to capture a measure. When an attribute is composed of other attributes, the attribute relationship model defines the relationship among the attributes. A measurement protocol model was concerned with how to measure an attribute consistently on a specific entity & the entity population model sets the normal value for an attribute [32]. Based on this framework, Li [31] proposed a metrics suite consisting of 6 new metrics – Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract Data Type (CTA), and Coupling Through Message Passing (CTM). These metrics were an extension/renaming of metrics proposed earlier in [7], where the authors evaluated their metrics on two commercial systems written in Ada. The authors [7] collected the data for three years and measured maintenance effort in terms of the number of lines changed per class. Their statistical analyses of the data led to the conclusion that their metrics could be useful in predicting maintenance effort.

Sheriff *et al.* [33] analyzed two o-o software projects developed at jet propulsion lab – micro generic controller, and sequence generator. These two projects were analyzed and compared using 3 C & K metrics – WMC, DIT, and NOC. The empirical results collected for these metrics were used to get insight into the complexity of two projects [33]. Wilkie *et al* [34] collected the data from 114 class samples for C & K metrics and noted some observations:

(1) Simple count of member functions can be used as definition of WMC & on average gives valid results.

(2) Cyclomatic Complexity can be used to measure the class complexity.

(3) WMC & DIT can be used to predict those classes that are likely to contain faults.

(4) Higher values of LCOM point out those classes, which could be good candidates for decomposition into more classes.

(5) CBO had no effect on fault-proneness, but had an impact on maintenance efforts.

(6) DIT & NOC combination gives a more thorough assessment of inheritance.

Emam *et al.* [35] studied the effect of class size on various C & K metrics [22] and a subset of metrics proposed in [12] and concluded that there is confounding effect of class size on validity of o-o software metrics, and thus class size effect needs to investigated further. A set of metrics for o-o design was presented in [36], which consisted of– Operation Complexity (OpCom), Operation Argument Complexity (OAC), Attribute Complexity (AC), Class Coupling (ClCpl), Class Hierarchy (CH), Cohesion (Coh), and Reuse (Re). These metrics were validated by using statistical regression model based data collected from two small projects and the data collected from the judgment of expert designers to the complexity of the design [36].

Another popular metrics suite consisting of 6 metrics was proposed in [37] and was given the name MOOD (Metrics for Object-oriented Design). The proposed metrics were Method Hiding Factor (MHF), Attribute Hiding factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (PF), and Coupling Factor (CF). MHF & AHF were measures of encapsulation, MIF & AIF concentrated on inheritance, and PF & CF tried to measure polymorphism.

The 6 MOOD metrics were discussed from measurement theory viewpoint and empirical data for these metrics was collected from three different projects [38]. This empirical data was analyzed keeping in view 4 aspects of o-o programming – encapsulation, inheritance, coupling, and polymorphism. The author [38] concluded that MOOD metrics could be used to provide an overall assessment of software, but he stressed upon the need of more empirical studies, before generalization of the results.

Lorenz & Kidd [12] proposed a set of metrics grouped into 4 categories – size, inheritance, internals, and externals. Size oriented metrics for o-o class focused on counts of attributes and operations of an individual class and average values for o-o software as a whole. Inheritance based metrics concentrated on the manner in which operations were reused through class hierarchy. Metrics for class internals were oriented towards cohesion, while external metrics were used to examine coupling and reuse [12]. Neal [39] did a study for validation of o-o software metrics and found that many of the then proposed metrics couldn't be considered valid measures of the dimension they claim to measure. He defined a model based on measurement theory for validation of o-o metrics and proposed 10 new metrics – Potential Methods Inherited (PMI), Proportion of Methods Inherited by a Subclass (PMIS), Density of Methodological Cohesiveness (DMC), Messages and Arguments (MAA), Density of Abstract Classes (DAC), Proportion of Overriding Methods in a Subclass (POM), Unnecessary Coupling through Global Usage (UCGU), Degree of Coupling Between Class Objects (DCBO), Number of Private Instance Methods (PrIM), and Strings of Message Links (SML).

Abreu *et al* [40] provided a new classification framework named as TAPROOT. This framework was defined along two independent vectors – category and granularity. Six categories of o-o metrics were defined – design metrics, size metrics, complexity metrics, reuse metrics, productivity metrics, and quality metrics. They [40] also proposed 3 granularity levels – software, class, and methods. But no empirical/theoretical base for the metrics was provided. Effectiveness of frameworks was studied in [41] and the authors found that the frameworks were not able to deliver the concept of flexibility and reusability. Then they developed a conceptual model for frameworks and a set of guidelines to build o-o frameworks. Aims of the guidelines were to improve usability of frameworks, flexibility, and reusability [41]. More recently in 2003, the usefulness of o-o framework to a domain specific business application has been studied from viewpoint of saving cost and improving quality of the software [42]. Two case studies were conducted, in each of which four kinds of applications were developed. All applications were developed in two ways – with and without o-o framework and results have shown that the frameworks were more useful in achieving a more efficient reusability [42]. The reusability has been reported to be useful in improving the quality and productivity. This fact has been emphasized with the help of an automated tool to measure the reusability and its effect on quality [43]. But at the same time, excessive usage of frameworks for reusability has been reported to increase the

overall complexity [44], in which the authors have tried to provide empirical evidence on the change in code organization with the evolution of the software product.

O-O design metrics have been used to assign the high-level design quality attributes of o-o software with the help of hierarchical model. The model has related design properties such as encapsulation, coupling, cohesion to quality attributes such as reusability and flexibility. The model has the capability of modification to include different relationships and weights [45]. Subramanian & Corbin [46] analyzed a Smalltalk software having 600 classes and collected various metrics. The authors tried to identify some metrics, which could be useful predictors of quality attributes such as size, reusability and complexity in o-o software.

6. COMPLEXITY METRICS

The applicability of traditional metrics of software complexity like McCabe's cyclomatic complexity, Halstead's science measures to o-o software was studied by Tegarden et al [4] and it was found that the traditional metrics were not directly applicable to o-o software. The same authors pointed out in [47] that the complexity of o-o software differs from structure software because of polymorphism and inheritance. Tegarden and Sheetz stressed that assessment of complexity of o-o software is possible through combination of structural complexity aspects and perceptual complexity concepts [17]. Sheetz et al [48] proposed that complexity of o-o software could be represented by a set of measures defined at the variable, method, object, and application levels of the systems. At the variable level, characteristics of data type, fan in, fan out, and variable polymorphism were considered. The method level complexity was defined to be based on number of input/output variables, fan in, fan out, fan down, and method polymorphism. Psychological complexity was accounted for object level complexity measure, which was divided into six groups- input/output variables, property definitions, message passing, inheritance structure, inheritance conflicts, and object polymorphism. The measures proposed at the application level consisted of number of abstract classes, number of concrete classes, maximum depth of the object hierarchy, maximum breadth of the object hierarchy, and the number of unique messages sent between objects [48].

The different complexity measures suggested at four different levels in [48] were extended by the same authors in [41] and they presented a model of o-o software complexity, which consisted of 44 different measures. This model tried to integrate all of then existing o-o complexity measures and some measures were identified, which accommodated for cohesion and coupling aspects of the system [41].

C & K proposed a simple measure of o-o software complexity as WMC [19, 22]. In the WMC metric, a weight for each method in a class was to be computed, which represented the complexity of the method. However, no specific metric was specified by C&K to measure this weight, as pointed out in [49]. C&K complexity metric [22] did not differentiate between complexity of reused classes and that of newly developed ones, although former was expected to have lower complexity and thus better quality [50]. Thus Kamiya *et al* [50] proposed a revised complexity metrics, which could be applied to the software, which had been constructed by reusing software components and validity & usefulness of the revised metrics have been proved with the help of the estimation of efforts to fix faults. Ebert and Morschel [51] also identified some o-o metrics for measuring complexity of o-o software. The identified metrics were integrated in SmallTalk Development support system and were used for quality analysis of various Smalltalk software [51]. Misic and Esic [52] tried to identify some very simple and easily countable metrics, which could estimate effort and complexity of o-o software. The metrics considered were grouped into two categories – class model measures and source code measures. These authors suggested some formulas based on these two categories for effort and complexity estimation. But the authors themselves have stressed upon the need of more empirical data and investigation of integrated effects of several metrics on complexity and efforts [52].

Almost all of the above mentioned metrics concentrated on the structural complexity of the object oriented software. However very large and complex software (consisting of lacs of lines of code and thousands of classes and objects) being developed in the software industry during the 21st century have given rise to another aspect of complexity named as cognitive complexity, which can be useful in improving the software engineering process [53]. Last decade did not give much attention to this complexity, but during this decade, spatial complexity has been reported to be of prime importance specially from maintenance view point [54, 55] Cognitive complexity metrics are always more difficult to be trusted and validated due to less-understood effect of human factors and human mind's working on the computation of such metrics [54, 56, 57]. In order to measure human efforts needed in comprehending the software, the concept of cognitive complexity was initiated by Douce et al in [56], where they introduced the concept of spatial complexity, which was based on theory of working memory and was reported to affect the understandability of source code [58]. Spatial ability is a term that is used to refer to an individual's cognitive abilities relating to orientation, the location of objects in space, and the processing of location related visual information. Spatial ability has been correlated with the selection of problem solving strategy, and has

played an important role in the formulation of an influential model of working memory. This concept of spatial abilities was applied to object oriented environment by [59] and two metrics class spatial complexity and object spatial complexity were introduced. This method was based on computing the distance among the definition and usage of class members and objects. The metrics were validated over a set of object oriented software and were found to be useful indicators of understandability of object-oriented software[59]. The spatial complexity measures for the Java based development were also defined and validated in [60.] Another way of computing cognitive complexity was proposed by Wang & Shao as Code Functional Size (CFS) in terms of cognitive weights [61] which were further modified in [62]. This measure was based on the internal structure of the source code and assigned different weights to Basic Control Structures (BCS) depending on their psychological complexity. This idea was further extended by also incorporating the effect of operators and operands [62]. Both of these proposed metrics were based on architectural aspect of the genitive informatics. But these measures were not exploiting the effects of encapsulation, inheritance and polymorphism. Although these measures were evaluated and validated using Weyuker's properties [63] and Briand et. al. renowned framework [64], but all of these validations were not explicitly proved for object oriented environment [65-68]. Another method of finding the usefulness of cognitive metrics, [69] identified a set of laws and properties specific to semantic and informatics of software. He also developed a new type of deductive semantics which can be very useful in future evaluation of cognitive and semantic based measures. But all of the researchers have stressed on cognitive complexity's importance and its strong impact on understandability and comprehension.

CONCLUSION

In this paper, a survey of o-o metrics has been presented. This paper has concentrated on the metrics suites available in the literature along with the metrics proposed for the most important attributeof quality i.e. complexity. In general, the researchers have found that metrics suites are more useful than a single metric to truly reflect the characteristics of object-oriented software. Further, complexity of the o-o software has been found to affects the quality and maintainability of the software. The understandability of the object-oriented software has been reported to be more dependent on cognitive complexities instead of structural complexity. Most of these stuies have been validated either empirically or theoretically, but these validations and evaluations are not yet well-established and need more extensive evaluation based on their semantic behavior. There is also a need for more empirical validations of many of the proposed metrics. At the same time, new metrics, with a strong theoretical and empirical background will be very useful in controlling various aspects of quality of the object-software during the designand maintenance phases of software life cycle.

References

[1] J. Rochester, M. Jackson, "Software Measurement Methods: Recipes for Success", *Information and Software Technology*, **36**, (3), (1994), 173–189.

[2] G. K. Gill, C. F. Kemerer, "Cyclomatic Complexity Density and Software Maintenance Productivity", *IEEE Transactions on Software Engineering*, **17**, (12), (1991), 1284–1288.

[3] D. R. Moreau, W. D. Dominick, "Object-Oriented Graphical Information Systems: Research Plans and Evaluation Metrics", *Journal of Systems and Software*, **10**, (1989), 23–28.

[4] D. P. Tegarden, S. D. Sheetz, D. E. Monarchi, "The Effectiveness of Traditional Metrics for Object-Oriented Systems", *Proceedings of Twenty-Fifth Hawaii International Conference on System Sciences*, IEEE Computer Society Press, **IV**, (Jan 1992), 359–368.

[5] S. L. Pfleeger, J. D. Palmer, "Software Estimation for Object-Oriented Systems", Proceedings of 1990 International Function Point User Group Fall Conference, San Antonio, TX, (1990), 181–196.

[6] S. C. Bilow "Applying Graph-Theoretic Analysis Models to Object-Oriented System Models", OOPLSA'92 Workshop on Metrics for Object-Oriented Software Engineering, Position Paper, (1992).

[7] W. Li, S. Henry, "Object Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, **23**, (1993), 111–122.

[8] W. Li, S. Henry, D. Kafura, R. Schulman, "Measuring Object-Oriented Design", *Journal of Object Oriented Design*, (July-August 1995), 48–55.

[9] B. Henderson-Sellers, "Object-Oriented Metrics –Measures of Complexity", Prentice Hall PTR, Upper Saddle River, New Jersey, (1996).

[10] T. J. McCabe, A Complexity Measure, *IEEE Transactions on Software Engineering*, **SE-2**, (4), (Dec 1976), 308–319.

[11] M. H. Halstead, "Elements of Software Science", North Holland, New York, (1977).

[12] M. Lorenz, J. Kidd, "Object Oriented Software Metrics", Prentice Hall, NJ, (1994).

[13] J. R. Abounader, D. A. Lamb, "A Data Model for Object-Oriented Design Metrics", *External Technical Report*, (Oct 1997).

[14] V. R. Basili, H. D. Rombach, "The TAME Project: Towards Improvement Oriented Software Environments", *IEEE Transactions on Software Engineering*, **14**, (1988), 758–773.

[15] K. L. Morris, Metrics for Object-Oriented Software Development Environments", unpublished Masters Thesis, M.I.T. Cambridge, MA, (1988).

[16] A. Lake, C. Cook, "A Software Complexity Metrics for C++", Technical Report 92-60-03, Oregon State Univ., (1992).

[17] D. P. Tegarden, S. D. Sheetz, "Object-Oriented System Complexity: An Integrated Model of Structure and Perceptions", OOPLSA'92 Workshop on Metrics for Object-Oriented Software Development, Washington DC, (1992).

[18] S. A. Whitmire, "Measuring Complexity in Object-Oriented Software", *Third International Conference on Application Software Measures*, La Jolla, CA, (1992).

[19] S. R. Chidamber, C. F. Kemerer, "Towards a Metrics Suite for Object-Oriented design". *Proc. OOPSLA'91, SIGPLAN Notices*, **26**, (11), (1991), 197–211.

[20] C. Rajaraman, M.R. Liu, "Some Coupling measures for C++ Programs", *Proceedings of TOOLS USA'92*, Prentice Hall, Englewood Cliffs, NJ, (1992), 225–234.

[21] C. Rajaraman, M. R. Liu, "Reliability and Maintainability Related Software Coupling Metrics in C++ Programs", *IEEE*, (1992), 303–311.

[22] S. R. Chidamber, C. F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, **20**, (6), (1994), 476–493.

[23] E. Weyuker, "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering*, **14**, (9), (1988), 1357–1365.

[24] Gursaran, G. Roy, "On the Applicability of Weyuker Property 9 to Object-Oriented Structural Inheritance Complexity Metrics", *IEEE Transactions on Software Engineering*, **27**, (4), (April 2001), 381–384.

[25] L. Zhang, D. Xle, "Comments on 'On The Applicability of Weyuker Property 9 to Object-Oriented Structural Inheritance Complexity Metrics'", *IEEE Transactions on Software Engineering*, **28**, (5), (2002), 526–527.

[26] N. L. Churcher, M. J. Shepperd, Comments on "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, **21**, (3), (1995), 263–264.

[27] S. R. Chidamber, C. F. Kemerer, Authors Reply to: "Comments on: A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, **21**, (3), (1995), 265–265.

[28] M. Hitz, B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", *IEEE Transaction on Software Engineering*, **22**, (4), (1996), 267–271.

[29] Gursaran, G. Roy, "Viewpoint Representation Validation: A Case Study on Two Metrics from the Chidamber and Kemerer Suite", *Journal of Systems and Software*, **59**, (1), (2001), 83–97.

[30] V. R. Basili, L. C. Briand, W. L. Melo, "A Validation of Object-Oriented Design Metrics As Quality Indicators", *IEEE Transactions on Software Engineering*, **22**, (10), (1996), 751–761.

[31] W. Li, "Another Metric Suite for Object-Oriented Programming", *Journal of Systems and Software*, **44**, (1998), 155–162.

[32] B. Kitchenham, S. S. Pfleeger, N. E. Fenton, "Towards a Framework for Software Measurement Validation", *IEEE Transactions on Software Engineering*, **21**, (12), (1995), 929–944.

[33] J. S. Sherif, P. Sanderson, "Metrics for Object-oriented Software Projects", *Journal of Systems and Software*, **44**, (1998), 147–154.

[34] F. G. Wilkie, B. Hylands, "Measuring Complexity in C++ Application Software", *Software- Practice and Experience*, **28**, (5), (1998), 513–546.

[35] E. K. Emam, S. Benlarbi, N. Goel, S. N. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", *IEEE Transactions on Software Engineering*, **27**, (7), (2001), 630–650.

[36] J. Y. Chen, J. F. Lu, "A New Metric for Object Oriented Design", *Information and Software Technology*, **35**, (4), (1993), 232–240.

[37] F. B. E. Abreu, "The MOOD Metrics Set", *Proceedings of ECOOPS'95 Workshop on Metrics*, (1995).

[38] R. Harrison, S. J. Counsell, R. V. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Metrics", *IEEE Transactions on Software Engineering*, **24**, (6), (June 1998), 491–496.

[39] R. D. Neal, "The Measurement Theory Validation of Proposed Object-Oriented Software Metrics", *Dissertation, Virginia Commonwealth University*, (1996).

[40] F. B. E. Abreu, R. Carapuca, "Candidate Metrics for Object-Oriented Software Within a Taxonomy Framework", *Journal of Systems and Software*, **26**, (1994), 87–96.

[41] J. V. Gurp, J. Bosch, "Design, Implementation and Evolution of Object-Oriented Frameworks: Concepts and Guidelines", *Software- Practice and Experience*, **31**, (3), (2001), 277–300.

[42] H. Fujiwara, S. Kusumoto, K. Inoue, A. Suzuki, T. Ootsubo, K. Yuura, "Case Studies To Evaluate a Domain Specific Application Framework Based on Complexity and Functionality Metrics", *Information and Software Technology*, **45**, (1), (Jan 2003), 43–49.

[43] L. H. Etzkorn, W. E. Hughes, C. G. Davis, "Automated Reusability Quality Analysis of OO Legacy Software", *Information and Software Technology*, **43**, (5), (2001), 295–308.

[44] G. Manduchi, C. Taliercio, "Measuring Software Evolution at a Nuclear Fusion Experiment Site: A Test Case for Applicability of OO and Reuse Metrics in Software Characterization", **44**, (10), (2002), 593–600.

[45] J. Bansiya, C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, **28**, (1), (2002), 4–17.

[46] G. Subramanian, W. Corbin, "An Empirical Study of Certain Object-Oriented Software Metrics", *Journal of Systems and Software*, **59**, (1), (2001), 57–63.

[47] D. P. Tegarden, S. D. Sheetz, D. E. Monarchi, "A Software Complexity Model of object-Oriented Systems", *Decision Support Systems: The International Journal*, **13**, (1995), 241–262.

[48] S. D. Sheetz, D. P. Tegarden, D. E. Monarchi, "Measuring Object-Oriented System Complexity", *Proceedings of First Workshop on Information Technologies and Systems WITS'91*, (Dec 1991), 285–307.

[49] R. Kalakota, S. Rathnam, A.B. Whinston, "The Role of Complexity in Object-Oriented System Development, Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences", *IEEE Computer Society Press*, (Jan 1993), 759–768.

[50] T. Kamiya, S. Kusumoto, K. Inoue, Y. Mohri, "Empirical Evaluation of Reuse Sensitiveness of Complexity Metrics", *Information and Software Technology*, **41**, (5), (1999), 297–305.

[51] C. Ebert, I. Morschel, " Metrics for Quality Analysis and Improvement of Object-Oriented Software", *Information and Software Technology*, **39**, (7), (1997), 497–509.

[52] V.B. Misic, D.N. Tesic, "Estimation of Effort and Complexity: An Object-Oriented Case Study", *Journal of Systems and Software*, **41**, (2), (1998), 133–143.

[53] N. M. Carod, A. M. Gabriela, N. Aranda, A. Cechich, "A Cognitive Approach to Improve Software Engineering Processes", *7th Workshop on Researchers in Computer Science* WICC (2005).

[54] Jitender Kumar Chhabra, K. K. Aggarwal, Yogesh Singh, Code & Data Spatial Complexity: Two Important Software Understandability Measures, *Information and Software Technology*, **45**, (8), (2003), 539–546.

[55] A. Mohan, N. Gold, P. Layzell, "An Initial Approach to Assessing Program Comprehensibility using Spatial Complexity, Number of Concepts and Typographical Style", *IEEE Working Conference on Reverse Engineering WCRE'04*, (2004).

[56] C. R. Douce, P. J. Layzell, J. Buckley, Spatial Measures of Software Complexity, *Technical Report*, Information Technology Research Institute, University of Brighton, UK, (Jan 1999).

[57] Y. Wang, J. Shao, A New Measure of Software Complexity based on Cognitive Weights, *Canadian Journal of Electrical & Computer Engineering*, **28**, (2), (2003), 69–74.

[58] A. Baddeley, Human Memory: Theory and Practice, Revised Edition, Hove Psychology Press, (1997).

[59] Jitender Kumar Chhabra, K. K. Aggarwal, and Yogesh. Singh, "Measurement of Object-Oriented Spatial Complexity," *Information and Software Technology*, **46**, (10), (2004), 689-699.

[60] J. K. Chhabra, Varun Gupta, "Towards Spatial Complexity Measures for Comprehension of Java Programs" *IEEE International Conference on Advanced Communications and Computing ADCOM 2006*, (Dec 2006), 430–433.

[61] S. Misra, "A Complexity Measure Based on Cognitive Weights", *International Journal of Theoretical and Applied Computer Science*, **1**, (1), (2006), 1–10.

[62] S. Mishra, "Modified Cognitive Complexity Measure", *Proceedings of 21st ISCIS'06 Lecture Notes in Computer Science*, **4263**, 1050–59.

[63] E. Weyuker, "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering*, **14**, (1988), 1357–1365.

[64] L. C. Briand, S. Morasca , V. R. Basili, "Property-Based Software Engineering Measurement", *IEEE Transactions on Software Engineering*, **22** (1), (Jan 1996), 68–86.

[65] S. Misra, A. K. Misra, "Evaluating Cognitive Complexity Measure with Weyuker Properties" *Proceedings of Third IEEE International Conference on Cognitive Informatics (ICCI2004),* 103–108.

[66] S. Misra, A. K. Misra, "Evaluation and Comparison of Cognitive Complexity Measure", *ACM SIGSOFT Software Engineering Notes*. **32**, (2), (Mar 2007), 1–5.

[67] S. Misra, 'Validating Modified Cognitive Complexity Measure' *ACM SIGSOFT Software Engineering Notes*. **32**, (3), (2007), 1–5.

[68] D. S. Kushwaha., A. K. Misra, "Robustness Analysis of Cognitive Information Complexity Measure using Weyuker's Properties". *ACM SIGSOFT Software Engineering Notes*, **31**, (1), (2006), 1–6.

[69] Y. Wang, "On the Informatics Laws and Deductive Semantics of Software", *IEEE Transactions on Systems, Man and Cybernetics*, **36**, (2), (2006), 161–171.