# Software Metrics: A Survey

Aline Lopes Timóteo[1], Alexandre Álvaro[1], Eduardo Santana de Almeida[2], Silvio Romero de Lemos Meira[1, 2]

*[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE) and [2]Recife Center for Advanced Studies and Systems (C.E.S.A.R), Brazil*

*{alt, aa2 }@cin.ufpe.br, {esa, silvio}@cesar.org.br*

## Abstract

*Software metrics were defined to be a method to quantify attributes in software processes, products and projects. The software metrics area has a lack of reports about metrics usage and real applicability of those metrics. In this sense, the main goal of this paper is briefing review research in software metrics regarding source code in order to guide our research in software quality measurement. The metrics review is organized in two ages: before 1991, where the main focus was on metrics based on the complexity of the code; and after 1992, where the main focus was on metrics based on the concepts of Object Oriented (OO) systems (design and implementation). The main contribution of this work is a large overview about software code metrics that can show us the evolution in this area, and a critical analysis about the main metrics founded on the literature.*

## 1. Introduction

The management of software development is an integral part of industry today but most software organizations face significant barriers in managing this activity. An Information Week survey found that 62 percent of the respondents feel that the software industry has troble producing good quality software [1]. Losses due to inefficient development pratices lead to inadequate quality that cost the US industry approximately $60 billion per year [2]. One approach that has been shown to result in improved quality and reduced costs is the use of software improvement activities.

One of the important determinants of success in software process improvement is the presence of metric programs. Many reports about metrics program had been found on the industry [3; 4; 5].

In the past few years, there have been a number of papers addressing software metrics, like: metrics related to performance [6], productivity [7], among others [8; 9; 10].However, we will foccus in metrics related to source code, because, we want to analyze the existents software metrics and verifies the evolution of this area.

We could identify some other surveys in software code metrics area [13; 17; 29; 56]. They not present a good view about this area at the time, they present a view about specific goups of metrics on their time. They not identify why some metrics could not survive and how the software code metrics area had been evaluate at the time.

In this work we will evaluate the state-of-the-art in software metrics related to source code, which the main goal is to analyze the existents software metrics and verifies the evolution of this area and why some metrics could not survive. Thus, we can understand how the source code quality evaluates throughout of the years.

The remainder of this paper is organized as follows: Section 2 software metrics knowledge necessary to understand the metrics analysis. Section 3 is surveys the state-of-the-art related to software code metrics. Section 4 presents the concluding remarks and future directions.

## 2. Software Metrics Knowledge

Quality is a phenomenon which involves a number of variables that depend on human behavior and can not be controlled easily. The metrics approaches might measure and quantify this kind of variables.

Some definitions for software metrics can be found on the literature [5; 11; 12]. The most common will be adopted in this paper, according to Daskalantonakis: “Software metrics is a method to quantify attributes in software processes, products and projects”.

In agreement with Daskalantonakis in [12] we can found the best motivation to measures, it is finding a numerical value for some software product attributes or software process attributes. Then, those values can be compared against each other and with standards applicable in an organization. Through these data could be draw conclusions about quality of the product or quality of the software process used.

In the recent literature, a large number of measures have appeared for capturing software attributes in a quantitative way. However, few measures have survived and are really used on the industry. A number of problems are responsible for the metrics failure, some of them are identified in [13; 14]. We select some of these problems to analyze the set of metrics presented on this survey. The main problems are:

- Metrics automation;
- Metrics Validation;

In the next sections we will present some aspects related by these problems, the section 2.1. presents a view about metrics automation; the section 2.2. presents some comments about metrics validation and the last section, 2.3. presents some aspects based on measurement goal or metrics definition.

## 2.1. Metrics Automation

We identified little work in developing tools for metrics extraction [13; 15; 16; 17; 18; 19; 20; 21]. It occurs because a large number of metrics is developed, but they don't have a clear definition. Normally a code metric is defined in a large context and it is validated only for a few set of programmer languages.

In our critical analysis we will verify if the selected metrics are implemented in a tool and the industrial relevance of this tool.

## 2.2. Metrics Validation

There are a number of problems related to theoretical and empirical validity of many measures [13; 14; 22], the most relevant of which are summarized next.

- Measurement goal, sometimes measurers are not defined in an explicit and well-defined context;
- Experimental hypothesis, sometimes the measure don't have a explicit experimental hypothesis, e.g. what do you expect to learn from the analysis?;
- Environment or context, the measure sometimes can be applied in an inappropriate context;
- Theoretical Validation, a reasonable theoretical validation of the measure is often not possible because the metrics attributes are not well defined.
- Empirical validation, a large number of measures have never been subject to an empirical validation.

This set of problems about validation will be used on our analysis. In next section we will be presented a survey about software metrics.

## 3. Software Metrics: A Survey

This section will present a survey related to the state-of-the-art in software metrics research.

Sommerville [12] classifies metrics in two categories: (i) Control Metrics generally associated with software process; and (ii) Predict Metrics, normally associated with software product.

In this work we focus is Predict Metrics, because the predict metrics measures the static and dynamic characteristics of the software [12]. According with some characteristic is possible calculate complexity of the code, instability, coupling, cohesion, inheritance, among others product attributes. Analyzing this attributes is possible infer about the quality of the products and suggest improvement points around effort, manutenability, testability and reusability.

In this paper we will present a timeline about software metrics, and it can be "divided" into 2 ages: section 3.1 presents the Age 1, before 1991, where the main focus was on metrics based on the complexity of the code; section 3.2 presents the Age 2, after 1992, where the main focus was on metrics based on the concepts of Object Oriented (OO) systems (design and implementation) and 3.3 section presents a summary about the this area.

## 3.1. Age 1: Metrics-Based on Code Complexity

According to [23; 24; 25] the first key metric used to measure programming productivity and effort was Lines of Code (LOC or KLOC for thousands of lines of code) metric. It still is used routinely as the basis for measuring programmer productivity.

Zuse and Fenton [24; 25] agree that in the mid-1970, there was a need for more discriminating measures rather than only LOC measure, especially with the increasing diversity of programming languages. After all, a LOC in an assembly language is not comparable in effort, functionality, or complexity to a LOC in a high level language. Also, there are easy identify the main problems in this measure, Environment or context and Measurement goal, the ruler not specify what kind of context the metric can be used.

Nowadays, the LOC metric is implemented in many used metrics tools [15; 21] and it can be used to calculate other metrics.

The 1970's started with an explosion of interest in measures of software complexity. Many works about software complexity can be found in literature [26; 27; 28]. The complexity metrics can be divided in two categories [29]: in section 3.1. we will present the program complexity metrics and in section 3.2. we will present the system complexity metrics.

### 3.1.1. Program Complexity Metrics

The most referenced program complexity metric is the Cyclomatic Complexity, v(G), [26]. The Cyclomatic Complexity is derived from a flow graph and is mathematically computed using graph theory (i.e. it is found by determining the number of decision statements in a program).

The cyclomatic complexity can be applied in several areas, including [30]: (i) Code development risk analysis, which measures code under development to assess inherent risk or risk buildup; (ii) Change risk analysis in maintenance, where code complexity tends to increase as it is maintained over time; and (iii) Test Planning, mathematical analysis has shown that cyclomatic complexity gives the exact number of tests needed to test every decision point in a program for each outcome.

This measure is based upon the premise that software complexity is strongly related to various measurable properties of program code. Nowadays, this measure is strongly used for measure complexity in industry and academy, because it has a clear measurement goal, McCabe specify clearly what is complexity and how to quantify complexity using Cyclomatic Complexity metric. This metric measure complexity in a structural context, it is great because the measure is not dependent of technology or program language used. This metric have been implemented in many metrics tools [15; 19; 21] and it had been validated in many industrial works [27; 31; 32].

Another program complexity metric found on literature is Halstead metric [33], it was created in 1977 and it was determined by various calculations involving the number of operators and the number of operands in a program. The Halstead measures are applicable to development efforts once the code has been written, because maintainability should be a concern during development. The Halstead measures should be considered for use during code development to follow complexity trends. A significant complexity measure increase during testing may be the sign of a brittle or high-risk module.

Halstead metric have been criticized for a variety of reasons, among them the claim that they are a weak metric because they measure lexical and/or textual complexity rather than the structural or logic flow complexity exemplified by Cyclomatic Complexity metric.

Halstead metric [33] which is different of the McCabe metrics [26], because the McCabe metric determines code complexity based on the number of control paths created by the code and this one is based on mathematical relationships among the number of variables, the complexity of the code and the type of programming language statements.

Nowadays, Halstead metric is not used frequently, because in your measurement goals are clearly related to the program language used, it doesn't have a large validation by industrial works [27; 31]. We find some tools implementing this metric [21].

In next section are presented more important system complexity metrics.

### 3.1.2. System Complexity Metrics

In the age 1, before 1991, we identify few works in system design metrics area. Yin and Winchester, [34] created two metric groups called: primary metrics and secondary metrics. The primary metric are expressed through extracted values of the specification of design. These metrics are based on two design attributes: coupling and simplicity. These metrics have been used in some organizations [29] and all reports indicate their success in pinpointing error-prone areas in the design.

The secondary metrics can provide an indication about the main system module or database table. The secondary metrics as: fan-in and fan-out, are used to compute a worst-case estimate of the communication complexity of this component. This complexity measure attempts to measure the strength of the component's communication relationship each other.

The validation of this metric is poor, because this measure ignores the use of modules on the system design. Some researches obtained a high correlation between values of the metric and error counts, but only when the analyzed system has small number of modules. One aspect to note about this work is that it gave rise to the first reported example of a software tool used for design [35].

Another complexity metric was defined by McClure [36]. This work focuses on the complexity associated with the control structures and control variables used to direct procedure invocation in a program. In this metric a small invocation complexity is assigned to a component which, for example, is invoked unconditionally by only one other component. A higher complexity is assigned to a component which is invoked conditionally and where the variables in the condition are modified by remote ancestors or descendents of the component. We don't find reports about tools that implements this metric and we found some researches about this metric application [27].

After sometime, Woodfield [37] publish another complexity system metric. He observes that a given component must be understood in each context where it is called by another component or affects a value used in another component. In each new context the given component must be reviewed. Due to the learning from previous reviews, each review takes less effort than the previous ones. Accordingly, a decreasing function is used to weight the complexity of each review. The total of all of these weights is the measure assigned to the component. Woodfield applied this measure successfully in a study of multiprocedure student programs. We don't find reports about tools that implements this metric. We found some reports about this metric application [27].

In 1981, Henry and Kafura [38] created another system complexity metric. Henry and Kafura´s metric determine the complexity of a procedure, which depends on two factors: the complexity of the procedure code and the complexity of the procedure's connections to its environment.

Henry and Kafura´s approach is more detailed than Yin and Winchester, [34] metric, because it observes all information flow rather than just flow across level boundaries. It has another major advantage in that this information flow method can be completely automated.

However, some definitions, like flows definition and modules definition, are confusing. Consequently different researches have interpreted the metric in different ways thus disturb the comparison of results [27]. According to [13] another problem in Henry and Kafura´s approach is the validation, because the algebraic expression on the metric definition is seems arbitrary and the application of parametric tests to data which is skewed is questionable. We don't find metrics tools implementing this metric.

This section presented an overview of the main works in software metrics area even 90´years. This analysis show that the large worry of the research with some project aspects like productivity, maintainability, testability and effort, and how the complexity was considered the main form of measure these aspects. In next section we will presented the main works in software metrics area, after 1992 until today.

## 3.2. Age 2: Metrics-Based on the Concepts of Object Oriented

In 90's occurred many changes in metrics research. Initially, in 70's and 80's, the research was about complexity metrics. In 90´s some events like the maturity of the software engineering techniques and the use accomplish of paradigm Object Oriented, OO, was responsible by a new direction in software metrics research. Some new metrics were created and your main target was reflecting the impact of the new techniques and paradigms in the software development. In this paper we will focus in software code metrics for OO.

The first suites of OO design metrics was proposed by Chidamber and Kemerer [39], which proposed six class-based design metrics for OO system (CK Metrics).

However, the CK metrics can be used to analyse coupling, cohesion and complexity very well, but the CK metrics suffer from unclear definition and a failure to capture OO specifics attributes. The attributes of data-hiding, polymorphism and abstraction not measured all and the attributes of inheritance and encapsulation are only partially measured.

The CK metrics are the most referenced [40; 41] and most commercial metrics collection tools available at the time of writing also collect these metrics [15; 19; 21]. The CK metrics validation catch our attention because is a complete work if we compare to other metrics. We could find researches in industry and academy [42; 43; 44], using many programmer languages.

Sometimes ago, Lorenz and Kidd created a set of OO design metrics [45]. They divided the classes-based metric in 4 categories [11]: size, inheritance, internals and externals. Size-oriented metrics for the OO classes focus on counts of attributes and operations. Inheritance-based metrics focus on the manner in which operations are reused in hierarchy class. Metric for internal class look at cohesion and code-oriented issues, and the external metrics examine coupling and reuses.

Probably CK metrics [39] are more known and complete then Lorenz and Kidd metrics [45] because include more OO attributes in its analysis.

To our knowledge no worked related to the theoretical validation of this metric has been published. According to [17], a tool called OO Metric was developed to collect these metrics, applied to code written in Smalltalk and C++.

In [46] was defined to measure the use of OO design mechanisms such as inheritance metrics, information hiding, polymorphism and the consequent relation with software quality and development productivity. The validation for this set of metrics is questionable for Polymorphism Factor metric (PF), because it is not valid, in a system without inheritance the value of PF is not defined, being discontinuous. According to [17] the MOODKIT is a tool for metrics extraction from source code, which supports the collection for C++, Smalltalk and Eiffel code.

The [47] metrics are the measurement of the coupling between classes. Their empirical validation conclude that if one intends to built quality models of OO design, coupling will very likely be an important structural dimension to consider. We could find a tool for this for metrics extraction.

Chatzigeorgiou validate your metric comparing it with classics OO software metrics. In the first analysis was verified the ability to account for the significance of the related classes and the ability to consider both incoming and outgoing flows of messages. The Lorenz and Kidd [45] these metrics not fulfilled to the ability to account for the significance of the related classes, but, although it fulfils ability to consider both incoming and outgoing flows of messages. We could find a tool for this for metrics extraction.

This section presented the main works based on OO source code metrics available on literature. Some problems were identified, analyzed and discussed in order to provide insights to our metric proposal in software quality. In the next section we will summarize this research and present it in a timeline.

## 3.3. Summary

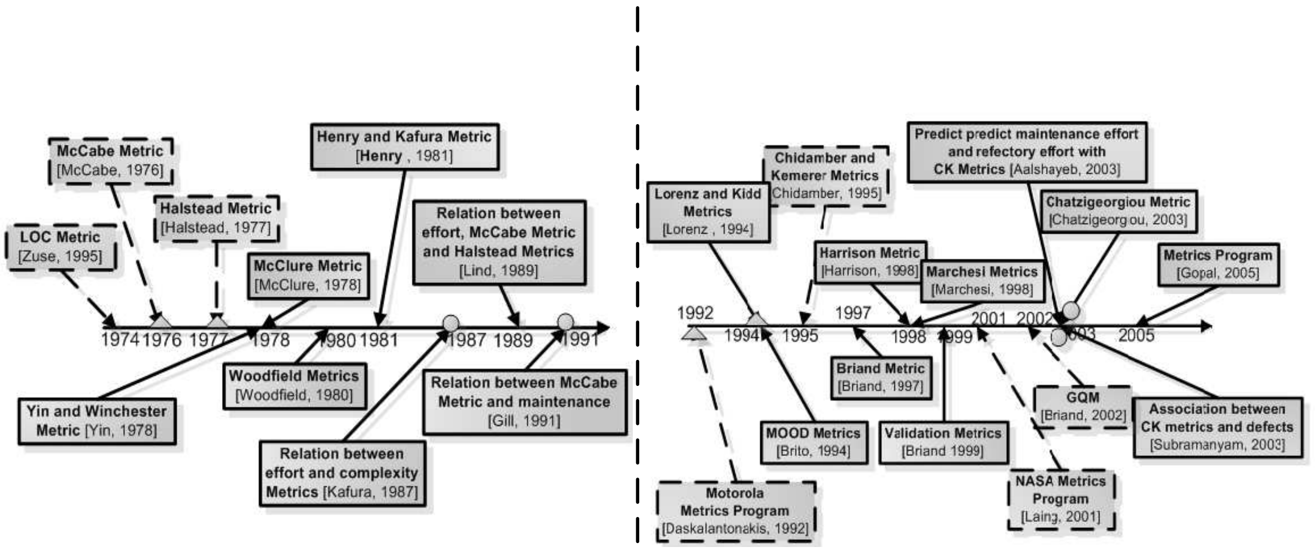
McCabe Metric [McCabe, 1976]
LOC Metric [Zuse, 1995]
Halstead Metric [Halstead, 1977]
McClure Metric [McClure, 1978]
Henry and Kafura Metric [Henry , 1981]
Relation between effort, McCabe Metric and Halstead Metrics [Lind, 1989]
1974 1976 1977 1978 1980 1981 1987 1989 1991
Yin and Winchester Metric [Yin, 1978]
Woodfield Metrics [Woodfield, 1980]
Relation between effort and complexity Metrics [Kafura, 1987]
Relation between McCabe Metric and maintenance [Gill, 1991]
Lorenz and Kidd Metrics [Lorenz , 1994]
Chidamber and Kemerer Metrics [Chidamber, 1995]
Predict predict maintenance effort and refectory effort with CK Metrics [Aalshayeb, 2003]
Chatzigeorgiou Metric [Chatzigeorgiou, 2003]
Harrison Metric [Harrison, 1998]
Marchesi Metrics [Marchesi, 1998]
Metrics Program [Gopal, 2005]
1992 1994 1995 1997 1998 1999 2001 2002 2003 2005
Briand Metric [Briand, 1997]
MOOD Metrics [Brito, 1994]
Validation Metrics [Briand 1999]
GQM [Briand, 2002]
Association between CK metrics and defects [Subramanyam, 2003]
NASA Metrics Program [Laing, 2001]
Motorola Metrics Program [Daskalantonakis, 1992]

**Figure1. Summarize the time line about code metrics.**

The research in software metrics continue intense in 90´s decade. Some other OO metrics were created like [48; 49], many works analyzing the metrics [50; 51] and about validating metrics [52; 53; 54] were published.

The software metrics scenario, after 2000, present little reports about new metrics. The proposed metric in [55] is not based in the classical metrics framework. The Chatzigeorgiou´s work is innovative because apply a web algorithmic from verify the relation between design classes and not use the traditional and existents metrics.

The timeline about software metrics can be clearly divided in two main ages: before 1992, when the researches where about complexity and the influence of it in quality attributes like maintenance, productivity, testability, effort.

After 1992 when the researches were affected by the internet revolution and the advent of the new technologies like OO, with the growing of the OO technology usage it was necessary develop metrics to measure coupling, cohesion, inheritance, and all important aspects of the OO technology. However, the 2000 years we could find a large diversity of researches. We found reports about metrics creation and many works about validation, institutional metrics program and approaches to create metrics.

The Figure 1 summarizes the timeline of research on software metrics area. There is a dotted line which marks the main change in this research area; some works were innovative in that time (represented by a "●" character on the timeline) and the works more referential (represented by a "▲" character on the timeline).

## 4. Conclusions

This paper presented a survey about software code metrics, providing an overview on what has been done in recent years, and it will also help researchers to get a comprehensive view of the direction of works in area of measurement.

According to this paper we can see the evolution of the software code metrics area by the time. In 70´s and 80´s years, the researches tried quantify the quality attributes by metrics strongly related with the used technology. It was a problem especially with the increasing diversity of programming languages.

In 90´s years, we can se the revolution in this area with advent of the new technologies like internet and OO. Many metrics used on the past could not survive a new time. Many new metrics were created and the researches had more variety then the other time. We found few reports about metrics creation and many works about validation, institutional metrics program and approaches to create metrics.

However, the same problems can be founded in all code metrics history. Lots of metrics did not survive the proposal phase. The identified reasons for this are theoretical and empirical validation problems, the metrics have not been built by using a clear defining process, and the metrics don't have a large support (tools) for metrics extraction.

Although, one of the main contribution of this paper is identify that some metrics get the success and confidence of the industry and they are largely used, like: cyclomatic complexity and Chindember and Kemerer metrics.

Based on this survey we will build a tool to analyze source code quality. The first step is selecting a set of software metrics. The second step is to do an analysis of the existent tools that implements the initial set of metrics and relate the selected metrics with quality attributes chooses. In future papers we will provide the survey about the software metrics tool and, consecutively, our proposal tool.

## References

[1] M. Hayes, "Precious Connection", InformationWeek, 2003, pp. 34-50.

[2] P. Thibodeau, L. Rosencrance, "Users Losing Billions Due to Bugs", Computerworld, vol. 36, 2002, pp. 1-2.

[3] Software Engineering Laboratory, "Software Assurance Technology Center", online, last update: 06/1995, available: http://satc.gsfc.nasa.gov/metrics/index.html

[4] T. Kilpi, "Implementing a Software Metrics Program at Nokia", IEEE Software, 2001, pp. 72-77.

[5] M. K. Daskalantonakis, "A Pratical View of Software Measurement and Implementation Experiences Within Motorola", IEEE Transactions on Software Engineering, vol 18, 1992, pp. 998–1010.

[6] B. N. Corwin, R. L. Braddock, "Operational performance metrics in a distributed system", Symposium on Applied Computing, Missouri - USA, 1992, pp. 867-872.

[7] R.Numbers, "Building Productivity Through Measurement", Software Testing and Quality Engineering Magazine, vol 1, 1999, pp. 42-47

[8] IFPUG - International Function Point Users Group, online, last update: 03/2008, available: http://www.ifpug.org/

[9] B. Boehm, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", U.S.Center for Software Engineering, Amsterdam, 1995, pp. 57-94.

[10] N. E. Fenton, M. Neil, "Software Metrics: Roadmap", International Conference on Software Engineering, Limerick - Ireland, 2000, pp. 357–370.

[11] R. S. Pressman, "Software engineering a practitioner's approach", 4th.ed, McGraw-Hill, New York - USA, 1997, pp. 852.

[12] I. Sommerville, "Engenharia de Software", Addison-Wesley, 6º Edição, São Paulo – SP, 2004.

[13] D. C. Ince, M. J. Sheppard, "System design metrics: a review and perspective", Second IEE/BCS Conference, Liverpool - UK, 1988, pp. 23-27.

[14] L. C. Briand, S. Morasca, V. R. Basili, "An Operational Process for Goal-Driven Definition of Measures", Software Engineering - IEEE Transactions, vol 28, 2002, pp. 1106-1125.

[15] Refactorit tool, online, last update: 01/2008, available: http://www.aqris.com/display/ap/RefactorIt

[16] O. Burn, CheckStyle, online, last update: 12/2007, available: http://eclipse-cs.sourceforge.net/index.shtml

[17] M. G. Bocco, M. Piattini, C. Calero, "A Survey of Metrics for UML Class Diagrams", Journal of Object Technology 4, 2005,pp. 59-92.

[18] JDepend tool, online, last update: 03/2006,available: http://www.clarkware.com/software/JDepend.html

[19] Metrics Eclipse Plugin, online, last update: 07/2005, available: http://sourceforge.net/projects/metrics

[20] Coverlipse tool, online, last update: 07/2006, available: http://coverlipse.sourceforge.net/index.php

[21] JHawk Eclipse Plugin, online, last update: 03/2007, available: http://www.virtualmachinery.com/jhawkprod.htm

[22] S. Morasca, L. C. Briand, V. R. Basili, E. J. Weyuker, M. V. Zelkowitz, B. Kitchenham, S. Lawrence Pfleeger, N. Fenton, "Towards a framework for software measurementvalidation", Software Engineering, IEEE Transactions, vol 23, 1995, pp. 187-189.

[23] H. F. Li, W. K. Cheung, “An Empirical Study of Software Metrics”, IEEE Transactions on Software Engineering, vol 13, 1987, pp. 697-708.

[24] H. Zuse, “History of Software Measurement”, online, last update: 09/1995, lavailable: http://irb.cs.tu-berlin.de/~zuse/metrics/History_00.html

[25] N. E. Fenton, M. Neil, “Software Metrics: Roadmap”, International Conference on Software Engineering, Limerick - Ireland, 2005, pp. 357–370.

[26] T. J. McCabe, “A Complexity Measure”. IEEE Transactions of Software Engineering, vol SE-2, 1976, pp. 308-320.

[27] D. Kafura, G. Reddy, “The Use of Software Complexity Metrics in Software Maintenance”, IEEE Transactions on Software Engineering archive, vol 13 , New Jersey - USA, 1987, pp. 335-343.

[28] B. Ramamurty, A. Melton, “A Syntheses of Software Science Measure and The Cyclomatic Number”, IEEE Transactions on Software Engineering, vol 14, New Jersey - USA, 1988, pp. 1116-1121.

[29] J. K. Navlakha, “A Survey of System Complexity Metrics”, The Computer Journal, vol 30, Oxford - UK, 1987, pp. 233-238.

[30] E. VanDoren, K. Sciences, C. Springs, “Cyclomatic Complexity”, online, last update: 01/2007, available: http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.ht ml

[31] R. K. Lind, K. Vairavan, “An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort”, IEEE Transactions on Software Engineering, New Jersey - USA, 1989, pp. 649-653.

[32] G. K. Gill, C. F. Kemerer, “Cyclomatic Complexity Density and Software Maintenance Productivity”, IEEE Transactions on Software Engineering, 1981, pp. 1284-1288.

[33] M. H. Halstead, Elements of Software Science, Operating, and Programming Systems, vol 7, New York - USA, 1977, page(s): 128.

[34] B. H. Yin, J. W. Winchester, "The establishment and use of measures to evaluate the quality of software designs", Software quality assurance workshop on Functional and performance, New York - USA, 1978, pp. 45-52.

[35] R. R. Willis, "DAS - an automated system to support design analysis", 3rd international conference on Software engineering, Georgia - USA, 1978, pp. 109-115.

[36] C. L. McClure, “A Model for Program Complexity Analysis”, 3rd International Conference on Software Engineering, New Jersey - USA, 1978, pp. 149-157.

[37] N. Woodfield, "Enhanced effort estimation by extending basic programming models to include modularity factors", West-Lafayette, USA, 1980.

[38] S. Henry, D. Kafura, "Software Structure Metrics Based on Information Flow", Software Engineering, IEEE Transactions, 1981, pp. 510-518.

[39] S. R. Chidamber, C. F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, vol 20, Piscataway - USA, 1994, pp. 476-493.

[40] M. Alshayeb, M. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", IEEE Transactions on Software Engineering archive, vol 29, 2003, pp. 1043–1049.

[41] R. Subramanya, M. S. Krishnan, “Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implication for Software Defects”, IEEE Transactions on Software Engineering, vol 29, 2003, pp. 297-310.

[42] L. C. Briand, S. Morasca, V. R. Basili, "Property-based software engineering measurement", Software Engineering, IEEE Transactions, vol 22, 1996, pp. 68 - 86.

[43] S. R. Chidamber, D. P. Darcy, C. F. Kemerer, "Managerial use of metrics for object-oriented software: anexploratory analysis", Software Engineering, IEEE Transactions, vol 24, 1998, pp. :629–639.

[44] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen, "An empirical study on object-oriented metrics", Software Metrics Symposium, 1999, pp. 242-249.

[45] M. Lorenz, J. Kidd, “Object-Oriented Software Metrics: A Practical Guide”, Englewood Cliffs, New Jersey - USA, 1994.

[46] A. F. Brito, R. Carapuça, "Object-Oriented Software Engineering: Measuring and controlling the development process", 4th Interntional Conference on Software Quality, USA, 1994.

[47] L. Briand, W. Devanbu, W. Melo, "An investigation into coupling measures for C++", 19th International Conference on Software Engineering, Boston - USA, 1997, pp. 412-421.

[48] R. Harrison, S Counsell, R. Nithi, "Coupling Metrics for Object-Oriented Design", 5th International Software Metrics Symposium Metrics, 1998, pp. 150-156.

[49] M. Marchesi, "OOA metrics for the Unified Modeling Language", Second Euromicro Conference, 1998, pp. 67-73.

[50] T. Mayer, T. Hall, “A Critical Analysis of Current OO Design Metrics”, Software Quality Journal, vol 8, 1999, pp. 97-110.

[51] N. F. Schneidewind, "Measuring and evaluating maintenance process using reliability, risk, and test metrics", Software Engineering, IEEE Transactions, vol 25, 1999, pp. 769-781.

[52] V. R. Basili, L. C. Briand, W. L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, vol 22, New Jersey - USA, 1996, pp. 51-761.

[53] L. C. Briand, S. Morasca, V. R. Basili, "Defining and validating measures for object-based high-level design", Software Engineering, IEEE Transactions, vol 25, 1999, pp. 722-743.

[54] K. E. Emam, S. Benlarbi, N. Goel, S. N. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", IEEE Transaction on Software Engineering, vol 27, 2001, pp. 630-650.

[55] A. Chatzigeorgiou, “Mathematical Assessment of Object-Oriented Design Quality”, IEEE Transactions on Software Engineering, vol 29, 2003, pp. 1050-1053.

[56] D. Kafura, "A survey of software metrics", ACM annual conference on The range of computing, New York - USA, 1985, pp. 502-506.