International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-3, Issue-4)

Research Article | **April 2014**

An Empirical Study of Software Metrics and Associated Aspects

Prachi Chhabra*, Lalit Bansal
CSE Kurushetra University,
India

Abstract—

Software Metrics are categorized to be an approach to quantify the characteristics in programming techniques, items and undertakings. The software measurements area has an absence of reports about measurements utilization and true domain of the measurements. In this aspect, the principle objective of this paper is informing survey investigate in programming measurements in regards to source code so as to guide our examination in programming quality estimation. The software metrics are proposed by number of researchers in different era and the major objective is to study these approaches. The fundamental approach of this work is an overview with some applicable works about programming code measurements. In this manuscript, the software metrics are analyzed proposed by assorted researchers.

Keywords: Metrics, Complexity, Object-oriented.
Relevant Topics: Software quality, Software Reuse and Software Metrics.

I. INTRODUCTION

Software Metrics refers to different estimations of machine programming and its advancement, which the primary imperativeness part is on understanding and regulating programming improvement practices and items. Existing written work discuss about advancement practices and items in programming measurements demonstrates to some significant examination, as might be seen in [ddaskalantonakis, 1992; Nasa, 1995] and a few reviews [kafura, 1985; Navlakha, 1987; Bocco, 2005]. Be that as it may, the entire reports present few insights about genuine tests, with this, is not conceivable break down the metric connection requisition. Then, in this proposal we will assess the state-of-the-workmanship in programming measurements identified with source code, which the principle objective is to break down the existents programming measurements and confirms the advancement of this territory, seeing how the source code quality assesses all around of the years. In any case, this work does not blanket other programming measurements recognized with execution [corwin, 1992], benefit [numbers, 1999], around others [ifpug, 1994; Bohem, 1995; Fenton, 2000].

II SOFTWARE METRICS MOTIVATION

A very few definitions might be found on the expositive expression recognized with programming measurements [daskalantonakis, 1992; Pressman, 1997; Somerville, 2003].As per [Somervvile, 2003] the best motivation to measures is that product estimation must discover a numerical quality for some product item characteristics or programming methodology qualities. At that point, those qualities could be contrasted against one another and models material in an association. As these information could be make determinations about nature of the item or nature of the product methodology utilized. Somerville characterizes the measurements in two classifications: (i) Control Metrics for the most part connected with programming procedure; and (ii) Predict Metrics, typically connected with programming item. In this work we focus the Predict Metrics, on the grounds that the foresee measurements measures the static and element qualities of the product [sommerville, 2004]. 04]. Agreeing with some trademark is conceivable ascertain intricacy of the code, flimsiness, coupling, union, legacy, around others item qualities. Dissecting this characteristics is conceivable deduce about the nature of the items and recommend change focuses around exertion, man tenability, testability and reusability. This segment will display a review identified with the state-of-the-craft in programming measurements research. The course of events about programming measurements can be "partitioned" into 2 ages: before 1991, where the principle center was on measurements dependent upon the unpredictability of the code; and after 1992, where the primary center was on measurements dependent upon the ideas of Object Oriented (OO) frameworks (outline and usage).This segment will display a review identified with the state-of-the-craft in programming measurements research.

III AGE 1 : METRICS BASED ON CODE COMPLEXITY

As stated by [li 1987; Zuse 1995; Fenton, 2000] the first key metric used to measure modifying gainfulness and exertion was Lines of Code (LOC or KLOC for many lines of code) metric. It still is utilized routinely as the support for measuring programmer benefit. As stated by [li 1987; Zuse 1995; Fenton, 2000] the first key metric used to measure modifying gainfulness and exertion was Lines of Code (LOC or KLOC for many lines of code) metric. It still is utilized routinely as the support for measuring programmer benefit.. The Zuze and Fenton [zuse, 1995; Fenton, 2000] concur that in the mid-1970, there was a requirement for more separating measures as opposed to just LOC measure, particularly with the expanding differences of customizing dialects. When its all said and done, a LOC in a low level computing construct is not equivalent in exertion, practicality, or multifaceted nature to a LOC in an abnormal amount dialect. The

1970's begun was a blast of enthusiasm toward measures of programming unpredictability. Numerous works about programming unpredictability could be found in writing [mccabe, 1976; Kafura, 1987; Ramamurty, 1988]. The intricacy measurements might be isolated in two classes [navlakha, 1987]: system unpredictability measurements and framework multifaceted nature measurements. In the following segment we will exhibit the many-sided quality measurements found in literary works identified with these classes.

A PROGRAM COMPLEXITY METRICS

The most referenced system multifaceted nature metric is the Cyclomatic Complexity, v(g), [McCabe, 1976]. The Cyclomatic Complexity was inferred from a stream chart and was numerically figured utilizing diagram hypothesis (i.e. it is found by deciding the amount of choice explanations in a project). The cyclomatic multifaceted nature might be connected in a few zones, including [vandoren, 1997]: (i) Code advancement hazard dissection, which measures code being worked on to evaluate innate hazard or danger development; (ii) Change hazard investigation in support, where code many-sided quality has a tendency to expand as it is upheld about whether; and (iii) Test Planning, numerical examination has demonstrated that cyclomatic intricacy gives the careful number of tests required to test each choice point in a system for every conclusion.

Another metric found on expositive expression is Halstead metric [Hal stead, 1977], it was made in 1977 and it was controlled by different figuring including the amount of drivers and the amount of operands in a project. Fundamentally, an admin is an inherent capacity or an image or an aggregation of images in the code that prepares an activity and the operands is essentially a consistent or a variable. The Halstead measures are appropriate to improvement deliberations once the code has been composed, on the grounds that maintainability ought to be a worry throughout advancement. The Halstead measures ought to be recognized for utilization throughout code improvement to take after unpredictability patterns. A huge intricacy measure increment throughout testing may be the indication of a fragile or high-chance module. Halstead metric have been censured for a mixed bag of reasons, around them the claim that they are a feeble metric in light of the fact that they measure lexical or text based unpredictability instead of the structural or rationale stream multifaceted nature exemplified by Cyclomatic Complexity metric. Halstead metric [Hal stead, 1977] which is diverse of the McCabe measurements [McCabe, 1976], in light of the fact that the McCabe metric decides code unpredictability dependent upon the amount of control ways made by the code and this one is dependent upon numerical connections around the amount of variables, the multifaceted nature of the code and the sort of customizing dialect proclamations.
In the next area displayed more paramount framework multifaceted nature measurements.

B. SYSTEM COMPLEXITY METRICS

Yin and Winchester, [yin, 1978] made two metric aggregations called: essential measurements and auxiliary measurements. The essential metric are communicated through concentrated qualities of the determination of configuration. These measurements are dependent upon two configuration properties: coupling and effortlessness. These measurements have been utilized within a few associations [navlakha, 1987] and all reports show their achievement in pinpointing slip inclined territories in the outline.

The auxiliary measurements can give an evidence about the principle framework module or database table. The auxiliary measurements as: fan-in and fan-out, are utilized to register a most noticeably awful case assessment of the correspondence unpredictability of this part. This multifaceted nature measure endeavours to measure the quality of the part's correspondence relationship one another.

Another many-sided quality metric was characterized by McClure [McClure, 1978]. This work keeps tabs on the multifaceted nature connected with the control structures and control variables used to regulate strategy conjuring in a system. In this metric a little conjuring multifaceted nature is allotted to a part which, for instance, is summoned unconditionally by one and only other segment. A higher intricacy is relegated to a segment which is conjured restrictively and where the variables in the condition are altered by remote precursors or descendents of the part.

Woodfield [Woodfield, 1980] 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.

In 1981, Henry and Kafura [Henry, 1981] 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, [Yin, 1978] 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.

In next section will be presented some important works where these presented metrics are used or compared.

IV COMPLEXITY METRICS: REPORTS

After 1981, we can not identified publications with new metrics but many studies about existent software metrics were finding. In 1985 Kafura publish a complete survey about software metrics [Kafura, 1985]. In this work, Kafura explain

about some more important complexity metrics, such as: McCabe metric [McCabe 1976], Halstead metric [Halstead, 1977], McClure metric [McClure 1978], Yin and Winchester metric [Yin, 1978], Woodfield metric [Woodfield, 1980] and Henry and Kafura metric [Henry, 1981]. This survey was a kick off to comparative study among complexity metrics did by Kafura.

In [Kafura, 1987], Kafura explored the relationship between a variety of software complexity metrics and the effect of maintenance activities on a medium-size system. Kafura uses 7 different complexity metrics in your study: 3 program metrics (McCabe's Metric [McCabe, 1976], Halstead´s Effort Metric [Halstead, 1977] and Lines of Code [Li, 1987]) and 4 system metrics (McClure's Control Flow Metric [McClure, 1978], Woodfield´s Syntactic Interconnection Measure [Woodfield, 1980], Henry and Kafura Metrics [Henry, 1981], Yau and Collofello´s Logical Stability Metric [Kafura, 1985]).

The Kafura´s work [Kafura, 1987] was important because confirmed the distinction between code and structure complexity metrics. This distinction was evident in that the maintenance changes to components might dramatically alter the values of metrics in one class of metrics without changing materially the values of metrics in the other class. This study also confirmed that it is useful to examine carefully those components which are "outliers" of one or more metrics. These outliers appear to be few in number in realistic systems and also seem to be fruitful places in which review and quality assurance resources might well be invested.

After that, in 1989, Lind [Lind, 1989] verifies the relationship between metrics and the software development effort. The relation metrics are: McCabe's metric [McCabe, 1976], Halstead´s program length [Halstead, 1977].

Lind's work [Lind, 1989] is important work because done interesting observation, as: confirm the low correlation between McCabe's metric and program length, above believed bigger the program length and bigger McCabe's metric value. Another Lind conclusion was about the relationship between effort and complexity metrics. Lind affirms that the program changes density declines then increasing McCabe and Halstead metrics values.

In 1992, Gill [Gill, 1991] described the relationship between McCabe's cyclomatic complexity and software maintenance productivity, given that a metric which measures complexity should prove to be a useful predictor of maintenance costs.

Gill's conclusions about relationship between McCabe metric and software maintenance productivity as: the influence of the program's size and complexity in the maintenance productivity, the maintenance productivity declines with increasing complexity density.

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.

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 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.

The first suites of OO design metrics was proposed by Chidamber and Kemerer [Chidamber, 1994], which proposed 6 class-based design metrics for OO system (CK Metrics).

However, the CK metrics can be used to analyze 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.

Sometimes ago, Lorenz and Kidd created a set of OO design metrics [Lorenz, 1994]. They divided the classes-based metric in 4 categories [Pressman, 1997]: 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 [Chidamber, 1994] are more known and complete then Lorenz and Kidd metrics [Lorenz, 1994] because include more OO attributes in its analysis.

The research in software metrics continue intense in 90´s decade. Some other OO metrics were created [Brito, 1994; Briand, 1997; Harrison, 1998], many works analyzing the metrics [Mayer, 1999; Schneidewind, 1999] and about validating metrics [Basili, 1996; Briand 1999; Emam, 2001] were published.

After 2000, we just can find little works reporting creation of new metrics, but some interesting works happened. In [Aalshayeb, 2003], Alshayeb investigate the relation between development effort and software maintenance in an agile process and in a process with long cycle of evolution. In this research Alshayeb used Chidamber and Kemerer metric [Chidamber, 1994].

Alshayeb´s work presented a practical example of the CK metrics usage. The main Alshayeb´s contributions are: with OO metrics, we can predict maintenance effort and refectory effort in classes in the short-cycled XP iterative process when the system design accumulates to certain mass. In other words, in the end of the interaction when have many code implemented, it is possible predict maintenance effort and refactory effort using CK metrics in a XP Process.

Alshayeb´s work indicate that results not had been satisfactory when was used a project with long cycle, because the used variable had values below of the established reliable values level.
In other work [Subramanyam, 2003], Subramanyam presented an interesting study about evidence in support of the association between a subset of CK metrics and defects. The subset of the CK metrics [Chidamber, 1994] used was Weighted method of class, Coupling between object and Depth of imheritance
Subramanyam´s work presented a detailed explanation about the experiment and show the influence of the languages in software metrics values. Subramanyam start the work verifying the relationship between class size and defect. His experiment indicates the size is associated with high number of defects. Another conclusion that is the effect of size varies according with program languages.
After, they affirm the relation between the WMC metric with defects. The results indicate that, for C++ classes, the effect of WMC in defects is significant. However, the analyzed indices not show support to conclusions about Java classes. The inherent differences between two languages may affect the influence of WMC on defects.
Last, Subramanyam verify the relation between CBO and DIT with defects. The same from other experiments [ref.] the results are different to specify languages. The conclusions are: C++ classes with higher CBO are associated with higher defects, whereas they are associated with fewer defects for the java; C++ classes as well Java classes with higher DIT values are associated with higher defects; and C++ classes with higher DIT as well higher CBO are associated with higher defects, wherever Java classes with higher DIT as well higher CBO are associated with fewer defects.
The software metrics scenario, after 2000, present little reports about new metrics. The proposed metric in [Chatzigeorgiou, 2003] 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.
This research [Chatzigeorgiou, 2003] shown a new metric for design evaluation. In this approach, Chatzigeorgiou considers the application of HITS algorithm (Hyperlink Induced Topic Search) to evaluate the OO design quality. Basically Chatzigeorgiou modifying the algorithm in order to count the number of discrete messages exchanged between classes, becoming possible to identify class more required and elements which imply a poorly design model.
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 [Lorenz, 1994] these metrics not fulfilled to the ability to account for the significance of the related classes, but, although it fulfills ability to consider both incoming and outgoing flows of messages.
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.
The research in software code metrics area follows two main directions:

- **Metrics Based on Complexity:** many researches was done about complexity in order to be possible measure effort, testability, maintenance and productivity; and
- **Metrics Based on the Concepts of Object Oriented:** 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

V CONCLUSION AND FUTURE WORK

This paper projects the study about programming code measurements, giving a flow on what has been carried out as of late, and it will additionally help analysts to get a thorough perspective of the course of works in region of estimation.

In this manuscript we have highlighted few issues in programming code measurements range, for example, measurements acceptance, on the grounds that a few measurements are accepted in principle; measurements range definition, the reports not demonstrate the victory reach of the metric or the test is completed in a couple of information set; and poor subtle elements in programming measurements reports, numerous reports have few data about the metric use, getting challenging comprehend the utilization connection and extent.
In light of this overview and further work we will construct an algorithm to break down source code quality. The principal step is selecting a programming measurements. The second step is to do a dissection of the existent apparatuses that executes the introductory set of measurements and relate the chose measurements with quality characteristics picks. In future papers we will give the overview about the product measurements apparatus and, sequentially, our proposal instrument.

REFERENCES

[1] Alshayeb, 2003] Alshayeb, M., Li, M.; (2003), "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", IEEETransactions on Software Engineering archive, Volume 29, page(s): 1043 – 1049.
[2] Chatzigeorgiou, 2003] Chatzigeorgiou, A.; (2003), “Mathematical Assessment of Object-Oriented Design Quality”, IEEE Transactions on Software Engineering, Volume 29, page(s): 1050-1053.
[3] Daskalantonakis, 1992] Daskalantonakis, M. K.; (1992), “A Pratical View of Software Measurement and Implementation Experiences Within Motorola”, IEEE Transactions on Software Engineering, Volume 18, page(s): 998 – 1010.

[4] Fenton, 2000] Fenton, N. E., Neil, M.; (2000), "Software Metrics: Roadmap", International Conference on Software Engineering, page(s): 357–370, Limerick, Ireland.
[5] Gill, 1991] Gill, G. K., Kemerer, C. F.; (1991), "Cyclomatic Complexity Density and Software Maintenance Productivity", IEEE Transactions on Software Engineering, page(s):1284-1288.
[6] Halstead, 1977] Halstead, M.H.; (1977), "Elements of Software Science", page(s): 128, New York, USA.
[7] Henry, 1981] Henry, S., Kafura, D.; (1981), "Software Structure Metrics Based on Information Flow", Software Engineering, IEEE Transactions, page(s): 510-518.
[8] Li, 1987] Li, H. F., Cheung, W. K.; (1987), "An Empirical Study of Software Metrics", IEEE Transactions on Software Engineering, Volume 13, page(s): 697-708.
[9] Lind, 1989] Lind, R. K., Vairavan, K.; (1989), "An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort", IEEE Transactions on Software Engineering, page(s): 649-653, Piscataway, NJ, USA.
[10] Lorenz, 1994] Lorenz, M. and Kidd J.; (1994), "Object-Oriented Software Metrics: A Practical Guide", Prentice Hall, Englewood Cliffs, New Jersey, USA.
[11] McCabe, 1976] McCabe, T. J.; (1976) "A Complexity Measure". IEEE Transactions of Software Engineering, Volume SE-2, page(s): 308-320.
[12] McCabe, 1989] McCabe, T. J., Butler, C. W.; (1989), "Design complexity measurement and testing", ACM Publication, Volume 32 , page(s): 1415–1425, New York, NY, USA.
[13] McClure, 1978] McClure, C. L.; (1978), "A Model for Program Complexity Analysis", 3rd International Conference on Software Engineering, page(s): 149-157, Piscataway, NJ, USA.
[14] Nasa, 1995] Software Engineering Laboratory; (1995) "SOFTWARE MEASUREMENT GUIDEBOOK", Web publication, accessed in: http://mdp.ivv.nasa.gov/index.html
[15] Navlakha, 1987] Navlakha, J. K.; (1987), "A Survey of System Complexity Metrics", The Computer Journal, Volume 30, page(s): 233-238, Oxford, UK.
[16] Pressman, 1997] Pressman, R. S.; (1997), "Software engineering a practitioner's approach", 4th.ed, McGraw-Hill, page(s): 852, New York, USA.
[17] Kafura, 1985] Kafura, D.; (1985), "A survey of software metrics", ACM annual conference on the range of computing: mid-80's, page(s): 502-506, New York, USA.
[18] Ramamurty, 1988] Ramamurty, B., Melton, A.; (1988), "A Syntheses of Software Science Measure and The Cyclomatic Number", IEEE Transactions on Software Engineering, Volume 14, page(s): 1116-1121, Piscataway, NJ, USA. [Sommerville, 2004] Sommerville, I.; (2004), "Engenharia de Software", Editora Addison Wesley, 6º Edição, São Paulo – SP.
[19] Subramanya, 2003] Subramanya, R., Krishnan, M. S.; (2003), "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implication for Software Defects", IEEE Transactions on Software Engineering, Volume 29, page(s): 297-310.
[20] Woodfield, 1980] Woodfield, N.; (1980), "Enhanced effort estimation by extending basic programming models to include modularity factors", ACM publication.
[21] Yin, 1978] Yin, B. H., Winchester, J. W.; (1978), "The establishment and use of measures to evaluate the quality of software designs", Software quality assurance workshop on Functional and performance, page(s): 45-52, New York, NY, USA.