Software Metrics: A Survey
Aline Lopes Timóteo¹, Alexandre Álvaro¹, Eduardo Santana de Almeida², Silvio Romero de Lemos Meira¹,²
¹Centro de Informática – Universidade Federal de Pernambuco (UFPE) and ²Recife 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 to present a brief 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
Currently, the management of software development is an integral part of the industry, 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 trebled producing good quality software [1]. Losses due to inefficient development practices lead to inadequate quality that costs for 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 process improvement activities [57].

One of the important determinants of success in software process improvement is the presence of a metric program and many reports involving metrics programs had been reported by the industry [3; 4; 5].

In addition, in the past few years, some papers have addressed the software metrics areas such as: metrics related to performance [6], productivity [7], among others [8; 9; 10]. However, our focus in this paper is metrics related to source code, because, we want to analyze the available software metrics and verifies the evolution of this area.

Therefore, the main goal of this analysis is selecting a set of metrics that can be used to measurement code quality. The selected metrics must be able to quantify the most important quality aspects for a reuse environment. According to [58] the main quality attributes for a reuse environment are: reusability, modularity, complexity, coupling, cohesion, maintainability and documentation.

Finally, in this work we will evaluate the state-of-the-art in software metrics related to source code in order to analyze the available software metrics and verify the evolution of this area and why some metrics could not survive. Moreover, after analyze the existent metrics we will select a set of them that can quantify the code quality attributes related before.

The remainder of this paper is organized as follows: Section 2 presents the software metrics background necessary to understand the metrics analysis. Section 3 surveys the state-of-the-art related to software code metrics. Section 4 presents the set of metrics selected to quantify the quality attributes, and, Section 5 presents the concluding remarks and future directions.

2. Software Metrics Background
Quality is a phenomenon which involves a number of variables that depend on human behavior and cannot be controlled easily. Some definitions for software metrics can be found in the literature [5; 11; 12]. In this paper, the most common will be adopted, according to Daskalantonakis [12]: “Software metrics is a method to quantify attributes in software processes, products and projects”.

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

In the recent literature, a large number of metrics have appeared for capturing software attributes in a quantitative way. However, few metrics have survived and are really used on the industry and 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 in this survey. The main problem is related to Metrics Validation, we will talk about it in the next section.

2.1. Metrics Validation

There are a number of problems related to theoretical and empirical validity of many metrics [13; 14; 22] and the most relevant are:

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

The set of validation problems presented before will be used to compound the critical analysis about the metrics reported in this survey.

In the next section, we will present a critical view about other surveys founded on the literature.

2.2. Related Work

In 1985 Kafura [56] published the first survey about software metrics. In this work, Kafura explain some of the most important complexity metrics [26; 33; 36; 37], these metrics will be detailed in the next section. This survey was the first step to make a comparative study among complexity metrics [27]. In this survey, Kafura made a short overview about the existent code metrics. Also, the positive aspects in this work are: it has a good review about empirical validation of these metrics; Kafura makes his own validation and present his results. In general these experiments have shown that significant relationships exist between the software metrics and quality attributes such as error characteristics, comprehensibility of code, length of coding time, and structural soundness. The negative aspect of this survey is that not present a rising review of the code metrics area, in other words, it not present a good overview about the evolution of this area.

After this, Navlakha [29] and Ince [13] published two complete surveys about system design metrics. Firstly, the Navlakha [29] survey presents poor information about system metrics area. In this survey the author selected two important system metrics, and presents all aspects about them, like a detailed description of the metrics, their theoretical and empirical validation. However, this survey not presents a criticism about the presented metrics or a good review about the area.

Ince [13] presents an innovative work, when the research in system design was beginning, he present the main metrics in system design, a critical analysis about the validation of these metrics and a review about metrics automation. The main goal in this work is the critical analysis about code metrics area. Ince made a strong defense about use system metrics instead code metrics, because using system metrics is possible detect design problems before implementation. Another contribution in this work is it identifies the main problems in system metrics area and some of them remain until now, some of them was reported in 2.1 section.

Another survey analyzed [17] is about software metrics for Unified Modeling Language1 (UML) technology. This survey presents a big overview about Object Oriented (OO) metrics that can be applied in a UML context. This overview includes critical analyses about the metric definition, reports about their theoretical and empirical validation, and works about metrics automation. The main contribution of this survey are reports about the existing relevant works related to metrics for class diagrams at initial stages of development, and which are the available metrics that can help practitioners in making decisions in the early phases of OO development.

Our survey is a complete approach because makes a complete time line about the code metrics evolution since the first metric in beginning of the software metrics research until nowadays. Another important view in our survey is analized why some metrics could not survive and they are not used until now.

3. Software Metrics: A Survey

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, our focus is to Predict Metrics, because predict metrics measure the static and dynamic characteristics of the software [12]. According to some characteristic is possible to calculate the code complexity, instability, coupling, cohesion, inheritance, among others product attributes. Analyzing these attributes is possible infer about the quality of the products and suggest improvement points around effort, maintainability, testability and reusability.

We will present a timeline about software code metrics, and it can be “divided” into 2 ages: the first one, before 1991, where the main focus was on metrics based on the code complexity; and the second one, after 1992, where the main focus was on metrics based on the concepts of Object Oriented (OO) systems (design and implementation).

3.1. Age 1: Metrics-Based on Code Complexity

According to [23; 24; 25], the first key metric used to measurement programming productivity and effort was Lines of Code (LOC or KLOC for thousands of lines of code). It is still 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 metrics rather than only LOC, especially with the increasing diversity of programming languages. Also, the main problem identified in LOC metric is: the measurement goal is variably according to the context or environment where it will be measured. For example, if you use LOC to measurement effort in an assembly context, it is not comparable in effort, to a LOC in a high level language. 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 metrics for software complexity. Many works about software complexity can be found in literature, we will talk about some of them in the next sections. The complexity metrics can be divided in two categories [29]: program complexity metrics (Section 3.1.1) and system complexity metrics (Section 3.1.2).

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.

We could identify some advantages about v (G) metric, some of them are: this metric is strongly used for measure complexity in industry and academy, because it has a clear measurement goal, McCabe specifies clearly what is complexity and how to quantify complexity using cyclomatic complexity metric. Another advantage of this metric is about measurement complexity in a structural or logical context, it is great because the measure is not dependent of technology or program language used.

Another advantage identified was about the validation of the cyclomatic complexity metric, this metric has a mathematical validation based on graphical theories. Also, this metric has a empirical validation in many industrial and academic works [27; 31; 32]. This metric had good acceptance, then we could find many metrics extraction tool that giving support for this metric some of them are [15; 19; 21].

However, we could identify some unfavorable points related to v (G) metric. The main of them is that metric must be applied when the source code development is done, it is bad because after develop the program we have a high coast in rework if some problem was identified.

Another program complexity metric found in the literature are Halstead metrics [33], they were created in 1977 and they were determined by various calculations involving the number of operators and the number of operands in a program. The Halstead metrics are applicable to development efforts once the code has been written, because maintainability and testability should be a concern during development.

The main advantage of the Halstead metrics 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 can be criticized for a variety of reasons, among them the claim that they are a weak metric because they measure lexical and/or textual complexity, it is bad because the measurement is coupled with the technology used on the analyzed context. Another unfavorable point in these metric that it 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 does not have a large validation by industrial works [27; 31]. A good example about Halstead metrics extraction can be founded in [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 metrics 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 are used to compute a worst-case estimate of the communication complexity of this component.

The main advantage about this work, it gave rise to the first reported example of a software tool used for design [35].

These metrics can be criticized for a variety of reasons. We can find a clear conflict in the measurement goal, because the reusability concerns about modules would make such these metrics a poor indicator of system complexity. The validation of these metrics is poor, because they had ignored the use of modules on the system design. Some research obtained a high correlation between values of the metric and error counts, but only when the analyzed system has small number of modules [13].

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. The main advantage about McClure metric is about his good definition, because it has a clear measurement goal and context. However, this metric is not validated theoretically and about his empirical validation we found only one report about this metric application [27].

After sometime, Woodfield [37] published another complexity system metric. He observed 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. The main advantage about Woodfield metric is about his good definition, because it has a clear measurement goal and context. However, this metric is not validated theoretically and about his empirical validation we found only one report [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.

The main advantage in Henry and Kafura’s approach is that it is more detailed than the other system complexity metrics related, because it observes all information flow rather than just flow across level boundaries. It has another major advantage since this information flow method can be completely automated.

However, it had been criticized for some definitions, like flows definition and modules definition, are confusing. Consequently, different research has interpreted the metric in different ways thus disturb the comparison of results [27]. According to [13], another problem with 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.

This section presented an overview of the main work in the software metrics area until 90’years. This
analysis shows that the large concern of the research with some project aspects such as productivity, maintainability, testability and effort, and how the complexity was considered the main form of measure these aspects.

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

In 90’s occurred many changes in the 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 the software metrics research. Some new metrics were created and the main target was reflecting the impact of the new techniques and paradigms in the software development. In this paper, we will focus on 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). 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 specific attributes, like polymorphism and abstraction (not measured all), the attributes of inheritance and encapsulation (only partially measured).

The main advantage of these metrics is their validation. The CK metrics theoretical validation catch our attention because is a complete work. Firstly, Chidamber and Kemerer made this validation, but we found many research making comments or stretching this work [17]. These metrics have a large set of works in industry and academy [42; 43; 44], using many programmer languages to make their empirical validation. The CK metrics are the most referenced [40; 41] set of OO metrics. They have a large number of commercial tools implementing these metrics [15; 19; 21]. The main critic about these metrics is that metrics could not measurement all OO aspects, like polymorphism and abstraction.

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 in how the operations are reused in hierarchy class and metric for internal class look at cohesion and code-oriented issues, and the external metrics examine coupling and reuse.

The main advantage of these metrics is that metrics can measurement static characteristics of software design, such as the usage of inheritance and the amount of responsibility in a class. However, to our knowledge, any worked related to the theoretical validation of this metric has been published. The empirical validation of these metrics is criticized because it was made by the own authors and only for a unique program language (C++)

In [46] were defined the MOOD metrics, they can measurement the use of OO design mechanisms such as inheritance metrics, information hiding, polymorphism and the consequent relation with software quality and development productivity. An advantage of these metrics are that metrics can cover a large number of OO attributes and they can be used in many granularity levels, for example you can measurement aspects in a package, in a class or in a class diagram. The biggest problem with these metrics is their theoretical validation. The validation for this set of metrics is questionable for Polymorphism Factor metric (PF) [46], because it is not valid, in a system without inheritance the value of PF is not defined, being discontinuous.

The Briand and Devanbu metrics [47] are the measurement of the coupling between classes. Their empirical validation concludes that if one intends to build quality models of OO design, coupling will very likely be an important structural dimension to consider. The main advantage of these metrics is the validation. In their empirical validation was demonstrated that all of these metrics fulfill the properties for coupling.

The research in software metrics continue intense in 90’s decade. Some other OO metrics were created like [48; 49] but they do not have the some relevance that the works related before. We can found too many works analyzing metrics [50; 51] and validating metrics [52; 53; 54] in this time.

The software metrics scenario, after 2000, presents few reports about new metrics. The proposed metric in [55] is not based on 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 mains advantage in Chatzigeorgiou metric is his theoretical validates. This metric was compared with classics OO software metrics. In the first analysis was verified that metric can measurement the distribution of responsibility among classes better than CK metrics. Also, when compared with MOOD metrics, Chatzigeorgiou metric presented a better value
for polymorphism than Polymorphism Factor metric (PF) [46]. However, we could not find an empirical validation for this metric.

This section presented the main works based on OO source code metrics available in the 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 present a brief overview about metrics automation.

3.3. Metrics Automation

The automation of metrics extraction is one of the most important ways for a success in a metrics approach. The main problem in metric automation is the lack of tool support and the integration of system design metrics into advanced development environments. We analyze two of the most popular metrics extraction tools for OO and some conclusions will be presented in this section.

The Metrics plug-in [19] provides metrics calculation and dependency analyzer. This tool measurement many metrics with average and standard deviation and detect cycles in package and type dependencies and graph them, some of these metrics are: CK Metrics, LOC, cyclomatic complexity, among others. The positive points in this tool are that implements some of the best accepted metrics. Another important point, it can be integrated in a Java\(^2\) development framework and it is an open source tool. However, this tool can be criticized by the poor documentation, in other words, if you need check the tool source code or integrate it in another framework, it is so difficult because of the bad documentation.

The JHawk tool [21] is used to measurement some important metrics and you can improve the performance, reusability, maintainability and overall quality of code. This tool not has a free documentation and is a property tool. It implements some of the most important metrics, for example: Halstead Metrics, cyclomatic complexity, CK Metrics, Yin and Winchester, among others. We make a simple example with the free trial, and we could identify some positive aspects in this tool, like: it organizes the metrics in a set with the same target, for example: metrics for classes are together in a view, metrics for package are together in another view. However, some negative aspects are: it is a property tool, it is not integrated with development environment and it have a poor free documentation.

We identify other tools like: refactorit [15] for code refactoring, CheckStyle [16] to analyze quality related to program language style and Coverlipse [20] a tool to give support for unit tests.

In each analyzed tools we could identify an implementation of a different set of metrics. However it is a static situation, in other words, if someone need introduce a different metric in one of these tools, it is not possible. Also, all analyzed tools not present a good documentation for to be reusable in other context or to be integrated in other development framework. The main problem identified in these tools is according to the definition adopted in this paper metrics are defined to quantify code attributes [12], but these tools only extract the metrics and not show the relation between metrics values and code quality attributes. In all analyzed tools the relationship between metrics and quality attributes is not automated.

In next section we will present a summary about this section and a timeline that show us the rising of the code metrics area.

3.4. Summary

The timeline about software metrics can be clearly divided in two main ages: before 1992, when the research where about complexity; and after 1992, where the main focus was on metrics based on the concepts of Object Oriented (OO).

In the first age we could identify a big preoccupation about measure the quality attributes by complexity and a beginning research about system metrics. The main transaction idea between these ages was a growing of the research in system metrics area. In the second age 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 research. We found few reports about metrics creation and many works about validation, institutional metrics program and approaches to create metrics. We believe it can be another transaction point in the future, but it is not clear now. The motivation for this new way in metrics research can be the perception that measure before implementation can reduce the coast, for example problems with maintainability, reusability, testability, and all quality attributes can be discovered before the implementation and it will reduce rework.

The Figure 1 summarizes the timeline of research on code metrics area. There is a dotted line which marks the thought transition between the ideas that complexity is the most important to be measured and

\(^2\) http://java.sun.com/
quality attributes must be measurable. 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).

We could identify to main ages in software code metrics timeline, the first one, before 1991, where the main focus was on metrics based on the code complexity; and the second one, after 1992, where the main focus was on metrics based on the concepts of Object Oriented (OO) systems (design and implementation). The first age had been worry about find problems on the developed code. However, the second age had been worry about find problems before code development, in the analysis faze.

In this survey we could see that 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 do not have a large support (tools) for metrics extraction.

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.

5. 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 research to get a comprehensive view of the direction of works in measurement area.

4. Towards Metric Framework to Evaluate Source Code Quality

The main goal of this survey is selecting a set of metrics that can be used to measurement code quality. The selected metrics must be able to quantify the most important quality aspects for a reuse environment. According to [58] the main quality attributes for a reuse environment are: reusability, modularity, complexity, coupling, cohesion, maintainability and documentation.

We select a set of metrics that have best acceptance and confidence in industry and academy. According to our context and the analysis made in this survey, the selected metrics are: LOC to measurement documentation; cyclomatic complexity, to measurement complexity and maintainability; CK metrics, to measurement reusability, coupling and cohesion; and Lorenz and Kidd metrics to calculate modularity.

References

[37] N. Woodfield, "Enhanced effort estimation by extending basic programming models to include modularity factors", West-Lafayette, USA, 1980.
[40] M. Alshayeb, M. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software