# An Approach for the Empirical Validation of Software Complexity Measures

**Sanjay Misra**

Department of Computer Engineering Atilim University, Ankara, Turkey
smisra@atilim.edu.tr

*Abstract: Software metrics are widely accepted tools to control and assure software quality. A large number of software metrics with a variety of content can be found in the literature; however most of them are not adopted in industry as they are seen as irrelevant to needs, as they are unsupported, and the major reason behind this is due to improper empirical validation. This paper tries to identify possible root causes for the improper empirical validation of the software metrics. A practical model for the empirical validation of software metrics is proposed along with root causes. The model is validated by applying it to recently proposed and well known metrics.*

*Keywords: Empirical validation; Preliminary empirical validation; advanced empirical validation; software metrics*

## 1 Introduction

The popularity of empirical studies has been rising in the field of software engineering since the 1970s. Its importance for software metrics is pointed out by several researchers [1], [2], [3], [4]. It is one of the major ways through which academicians and scientists can assist industry in selecting new technology. On the other hand, it is a common observation that the standards of empirical software engineering research is not up to a satisfying level [5]. According to surveys on the papers on empirical validation [1], examples of poor experimental design, the inappropriate use of statistical techniques and conclusions that do not follow the reported result were found. Another survey on 600 published papers reported that a considerable amount of research papers lack experimental validation. Further, they use informal (assertion) forms of validation and use case studies approximately 10% of the time; it was also observed that their experimentation terminology was sloppy [4]. In addition, empirical study is often used synonymously with experiments and used in an inconsistent manner [6].

In the case of software complexity measures, the situation is quite similar. There is no match in content between the increased level of metric's activity in academia

and industry [7]. Evidence shows that any successful adoption and implementation of these metrics is limited [8]. We are aware of this fact that there are other factors which impact on the successful metric program in industry, e.g. institutional forces [8]; however, this stage occurs when a metric is applied in the industrial environment. The major problem with the existing metrics available in literature is that they have not been tried to implement in the industrial environment. One can easily find many academic works in the literature of complexity in measures without proper empirical validation. These poor findings are not only due to the lack of clear-cut guidelines and explanations for different types of empirical studies; additionally, the absence of availability of real environments for the implementation of metrics is another issue to consider. In many cases it is a challenging task to identify any direct link between researchers and the software industry, and this makes it difficult to validate the metrics against the projects in software business. As a consequence, researchers try to validate their metrics through other means, such as experiments in laboratories, class rooms or with available data/programs from the Internet. Most of the time, those means can provide only a partial empirical validation. However, for a proper empirical validation, we believe that one must apply the new technology/metric to real data/projects from industry.

Based on the above rationale and motivation from the insufficient discussion on empirical studies in literature, we have developed a model for practical empirical validation. This model is based on the evaluation of the common practices adopted for empirical validation of software metrics. In this model we suggest the application of empirical validation in two parts, namely as preliminary empirical validation, advanced empirical validation and acceptance. The preliminary empirical validation includes the initial validation of the metric by applying it to different test cases and examples. In advanced empirical validation a new metric is tested by using real projects from the industry. Finally, after the replicated experimentations in different environments, the acceptance of the metric(s) by industry is the final step of our model. In fact all the steps given in the model are not new; we have accumulated these different approaches and compiled/presented them in a formal way. We continue this discussion in the next section. The validity of our model is checked by applying it to some well known metrics, e.g. CK metrics suite [9] and entropy metrics [10].

**Research Questions and Methodology**

In line with what we have presented so far, we have identified that discussions on already proposed metrics may need further exercises for empirical validation. With this motivation, we found this point has received less attention and care in the literature on software metrics; hence it constitutes a potential gap. To address this gap, we have generated the following research questions:

*Why have most of the proposed metrics available in literature not received acceptance from the software industry? (1)*

*Are the researchers following any proper guideline while proposing and validating (empirically) their metrics? (2)*

To address these questions, we made an exhaustive literature survey for metrics, as we had already started to build a body of knowledge on measurements in software engineering. With the large number of metrics limiting our survey, we focus on the following two points:

Firstly, we have selected metrics including the CK metric suite and the Entropy metric, whose goals were similar and proposed at the same time. An additional motivation arose, namely that the CK metric suit has considerably higher reputation and adoption in the industry, while the Entropy metric has not achieved such acceptance, although both metrics bear strong background work. This drove us to investigate and evaluate by which practices the CK metric suite has become a benchmark. We are leaving out of the discussion other useful and widely accepted metrics, such as function point analysis, due to the limitations of our research.

Secondly, we have selected metrics which were proposed in last two years (2009 and 2010), which is another limitation within this work.

To answer (1) we have performed the survey on different types of empirical validation techniques which are commonly adopted by the developers of software metrics. We have reviewed all to our knowledge and propose a simple model, which mainly consists of all the important stages required for what we call "proper empirical validation". We validate our model on two metrics.

Again to validate (1) by the model which is developed to overcome the problem, we have analyzed the metrics proposed recently (in last two years).

The paper is organized in the following way. The next section summarizes the different types/ways of the empirical validation which are commonly used for the validation of software metrics. In section three, we introduce the metrics on which we apply our model. We present a model for the proper empirical validation in Section 4. A case study is presented for demonstrating our model in Section 5, which is further validated with newly proposed metrics in Section 6. Some observations and suggestions are given in Section 7. The conclusions drawn from this work are given in Section 8.

Before we go any further, we want to clarify that the terms software metric, software measure and software complexity metric/measure are used synonymously in the literature. Although one can find several definitions [11] for software complexity, we follow the IEEE [12] definition, which defines software complexity as "the degree to which a system or component has a design or implementation that is difficult to understand and verify". Additionally, IEEE [12] also defines metric as "as a quantitative measure of the degree to which a system, component, or process possesses a given attribute".

# 2 Common Practices Adopted for Empirical Validations of Software Metrics: An Analysis

There are several practices adopted by developers of software metric programs. Common practices adopted for empirical validations of software complexity measures include:

1 Small application programs to demonstrate the metric(s) or measure(s)

2 Formal experiments/examples reported in literature

3 Case studies/surveys available on the web

4 Experiments in laboratories or classrooms

5 Experiments at workplaces in the industry but with users off the subject i.e. who are not potential users.

## 2.1 Small Application Programs to Demonstrate the Applicability of Metric(S) or Measure(S)

Normally, most of the metrics are demonstrated through implementation on small application program(s). However, if a researcher is applying only this method for their validation, it may not be sufficient even for complete demonstration of the metric(s). For example in [13], the authors claim that their metric provides some indications for the level of coupling; however, from the example program which they provide in the paper, it is not straight-forward to identify how coupling can be estimated. In addition to this issue with this type of practice, if these programs are developed by the developer of the metric(s), these programs cannot be taken even for demonstrating the metric. It is because the developer of the metric(s) knows what he wants to prove with his metric and these programs may be specifically developed for this purpose (i.e. to validate the metric). It is worth mentioning that we do not claim that the researchers are stretching the truth; however, the application of the metric on these programs cannot be justified as they do not provide the reader a transparent implementation. For example, in the validation of improved cognitive information complexity measure [14], the authors themselves developed all the programs for theoretical validation then they claimed that their metric were properly validated; however, there are considerable "dark spots" left to the reader, including how one can prove its practical usefulness without implementing on real projects.

## 2.2 Formal Experiments/Examples Reported in Literature

The validation of software complexity measures with formal experiments performed on example(s) available in literature provides the only way how to implement the metric, and they stand without practical application. In most of the

cases these inappropriate examples do not satisfy the purpose of real empirical validation of the metrics proposed. Small application programs can only be used to get a preliminary idea about a metric; however, it is hard to accept as a proof of practical applicability as a metric, i.e. this cannot be the way of complete empirical validation. An example of this type of practice can be observed in the proposal of the unified complexity measure [15]. The authors have considered several examples from a book on a programming language.

## 2.3 Case Studies/Survey Available on the Web

One of the other common practices to validate software metrics/measures is through case studies. These case studies are sometimes small projects reported in literature or on the web [16], [17], or some large programs. Naturally, this way can only demonstrate the implementation of metric on big software products; however, in no case does it represent the practical applicability of the programs. In fact if the proposer of the metric is applying it on some programs/projects available on the web, we can treat it as a survey. One of the examples for this method is the metrics proposed by Aggarwal et al. [18], where the authors have proposed two metrics and validated them with JAVA programs available on the web. Observations show that their process of validation is open to discussion [19]. Further similar to the small application programs, the application of the metrics on such projects available on the web cannot be justified as they do not provide the reader a transparent implementation.

## 2.4 Experiments in the Laboratory or Classroom

Some authors [20-21] try to validate their metrics with students in classrooms or laboratories. In fact, in software engineering, several researchers suggest performing empirical validation with students in a classroom environment [22, 23]. The examples of experiments in the classroom are: controlled experiments (with graduate students), observational studies (professionals, graduate students) and case studies (projects as part of class work). Although it is arguable that students are the future software developers, experiments with students may reduce the practical value of experiments [1]. A validation process based on such data may be acceptable only for gaining initial knowledge regarding some quality factors, such as understandability.

## 2.5 Experiments at Workplaces in the Industry but with Users off the Subject, i.e. Who Are Not Potential Users

Sometimes, authors/developers of a new metric program try to validate their work in small and medium scale software industries. This may be a convenient way for academicians whose students are working in those companies. Although there is

no harm in utilizing the available software industries in the vicinity, a worrying issue is that these companies may not really evaluate or validate the metric program. In common practice, it is not common to find such small- and medium-sized software industries who adopt a software metric program in the organisation [24]. Hence, assuming a real validation of a new metric program may be open to discussion.

By adopting all these practices, one can only guess the preliminary idea for the metric. Also it is easier to adopt these practices for a freshman developer of a metric program; however, as a result, they can barely go beyond contributing to a publication. This is the reason we want to point out that, while most of papers in the area of software metrics/measures are gradually increasing, their adoption from industries is limited. In fact this is not only the situation in the case of software metrics, but also a problem in general; i.e. a lack of proper experimentations in software engineering. In an analysis of the eight papers published in IEEE [1] transactions on software engineering, the reviewer found examples of poor experimental design, inappropriate use of statistical techniques and conclusions that did not follow from the reported results. The reviewer further commented that the authors of those papers are well-known for their empirical software engineering work.

# 3 Chidamber et al.'s Metric Suite and Kim et al.'s Metrics

In this section, we introduce two different metric sets. In one of them, empirical validation is in the core of the development of the metric program, and in the other, the authors have adopted the casual process. We want to show the result of both practices. It is worth mentioning here that we do not discuss those papers in which rigorous empirical validation is the core part of the reported research.

We introduce metrics which were developed by Chidamber and Kemerer [9] and Kim, Shin and Wu [10] for object-oriented (OO) programming. Both groups proposed their metrics approximately at the same time, in 1994 and 1995, respectively. Although both proposals of metrics were introduced for OO systems at the same time and for the same objective (i.e. measuring the complexity of OO systems) Chidamber's and Kemerer's metric suite has gained more popularity in the software industry. On the other hand, the metrics proposed by Kim et al. have not found that much acceptance in the industry and are used only for literature support in research papers. We want to clarify that our intention is not to evaluate or criticize any particular metric(s), but rather to evaluate whether they have success or failure in practice. Furthermore, we will use these metrics as a case study to validate the effectiveness of our model in Section 5.

### 3.1 The Chidamber and Kemerer (CK) Metrics Suite

Chidamber et al. [9] proposed a suite of metrics which includes five well know metrics: WMC: weighted method per class; RFC: response for a class; NOC: number of children; LCOM: lack of cohesion in methods; and CBO: coupling between objects.

In the WMC, they suggested that one can calculate the weight of the method by using any procedural metric. They used cyclomatic complexity [25] for measuring WMC and assumed the weight of each method to be one.

The second metric, RFC, is defined as the total number of methods that can be executed in response to a message to a class. This count includes all the methods available in the class hierarchy. The depth of inheritance tree (DIT) and the number of children (NOC) are other two important CK measures. The former represents the maximum length from the node to the root of the tree and the latter is the number of immediate subclasses subordinated to a class in class hierarchy.

The LCOM metric is for cohesion and is counted as the number of common attributes used by different methods.

Another metric in their suite is CBO, which measures interactions between objects by counting the number of other classes to which the class is coupled. As stated previously, these are most accepted metrics in the OO domain in the industry; we are not providing the detail of the each metric and refer readers to the original paper [14].

### 3.2 Complexity Measures for OO Programs Based on Entropy

Entropy [26] is a common concept and applied by several researchers [27-30] to measure the complexity of software. Kim et al. [10] proposed three metrics for OO programs based on the entropy concept. These metrics are: the class complexity, the inter-object complexity, and the total complexity for OO programs. Basically their first metric, class complexity, evaluates the information flows between attributes and functions in a class. Their second metric, inter-object complexity, measures the information flows between objects. The third metric, total complexity, adds the class complexity and the inter-object complexity.

## 4 A Model for Empirical Validation for Software Complexity Measure

Empirical studies [22] are used to investigate software development and practices for understanding, evaluating, and developing in proper contexts. It allows the analyst to test out the theories with the support of empirical observations. It

includes formal experiments, case studies and surveys observed in industry, the laboratory or classroom [1]. All these different types of empirical validation techniques can also be applied to the validation of software metrics. However, as we have demonstrated in Section 2. All these independent validations are not suitable for the empirical validation of software complexity measures. They provide only a preliminary understanding of the proposed metrics. In other words, all these practices are not without benefit, but they are only suitable for introductory validation. With this point of view, we recommend these practices for preliminary empirical validation.

Ideally, we believe that when a new metric is applied to real projects from industry, its validity should later be evaluated against other similar metrics. However, in many cases, the type of empirical study depends upon situation and circumstances and, in the initial phases of any new proposal, it is not always possible to apply a new metric directly to the real projects in industry. A reason for this might be the following: if the developer of the metrics is an academician and at the particular time does not have access to the proper real (industrial) environment, then he tries to validate his proposal through other means (data and projects on the web).

In considering these practical problems related to empirical validation, the suggested guidelines in the proposed framework are categorized in two major stages as preliminary and advanced empirical validations, which are further classified into different stages. Accordingly, we suggest seven steps in total.

a) **Preliminary Empirical Validation**:

Preliminary empirical validation is divided into four stages. The first phase includes small experiments, case studies, and the comparative study and analysis of work. The second stage includes the application of the metric on real cases from industry.

**1 Demonstration of the Metric(s)**

This stage is based on short experimentations. The metric(s) should be demonstrated with real example(s). Here the meaning of real example is that the examples must be complete enough to demonstrate each aspect of the metric. In this respect, the example may be developed by the developer of the metric. Further, several programs/examples can be taken, if it is needed for demonstrating the metric. For example, for the demonstration of the cognitive functional size metric [20], the authors have developed three small programs. This is the first stage and most of the developers of the metric complete this step.

**2 Case Study**

After demonstrating the metric with short example(s), a case study is required. Here the case study refers to a relevant real project to which a new metric program should be applied. In fact, preliminary ideas regarding

the usefulness of a new metric are only validated by applying it to data collected from a real project. One can chose a real project from the web, literature or working projects in the departments, to which he can apply the metric to verify its applicability. In fact this stage is recommended when the data from industry is not readily available.

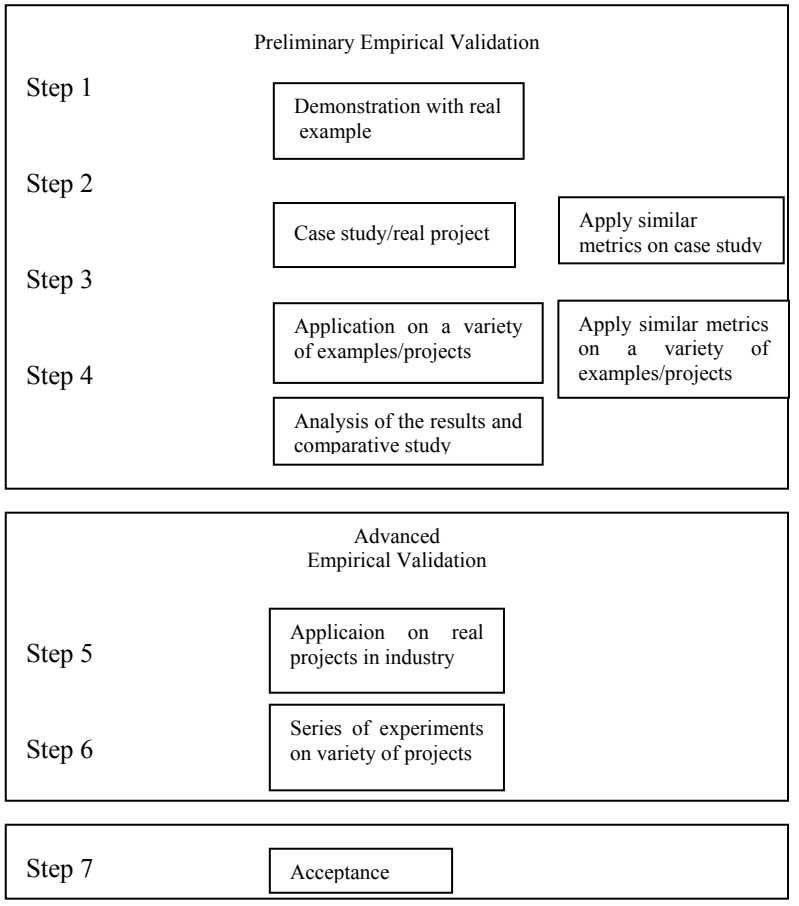


Figure 1
Proposed Model for Empirical Validation of Software

In parallel, similar metrics should also be applied on the case study and the results should be compared with the results of the proposed metrics. If the developer cannot find measureable differences between the proposed metrics and the available metrics, then he can either withdraw his proposal or can improve it.

**3 Test Cases**

A comparison with similar metrics provides valuable information on the usefulness of the new metric. It will also help in the analysis of the behavior of the metric in a variety of projects. In this stage, the metrics should be applied on a variety of examples/test cases/cases studies, and it should also be applied in different environments, if possible. Here we recommend a variety of examples/test cases/ case studies to check the applicability of the metric(s) on a variety of projects. Further, for comparison, similar metrics must be applied on those projects where the new metric is applied.

4 **Analysis**

The last stage of the preliminary empirical validation is to perform an analysis of the results and the comparative study, which have been done in previous stages. The results of the analysis and comparative study will help the developer to convince industry to apply the new metric program for advanced empirical validation.

**b) Advanced Empirical Validation**

**5 Real Project in Industry**

The acceptance of a new software metric is open to discussion if its usefulness is not proved in the software industry. For its acceptance, firstly the proposed metric must be applied by software developers on real working projects. It is worth mentioning here that the program should be applied by the professionals who are working on the projects. This is because they are key informants who can really evaluate the practical applicability of the new metric program.

**6 Family of Experiments**

The proposed metric must be applied in different projects and in different environments. The reason for this is that, after performing a family of experiments, one can build up the cumulative knowledge to extract useful measurement conclusion to be applied in practice [31].

**7 Acceptance**

After the series of experiments, the results should be analyzed and compared. If the new metric(s) is proved to be better than the existing metrics/metric programs in an organization it can be considered for acceptance. Otherwise it may require further improvement. After improvement of the metric(s), the same validation process should be revisited from the beginning, i.e. from the first stage of this model.

We have suggested seven steps for the empirical validation of metrics. These steps are not new; however, the proper adoptions by the developers of software metrics are limited. With this point in mind, we have presented them in a formal way in order to provide a straightforward guideline to help developers of software metrics

to gain a clear understanding of the proper empirical validation process. Further, the steps provided in this model are only guidelines and the practical implementation of this model may depend on the actual situation and circumstances. For example, after the demonstration of the metric in the preliminary phase (step 1), if one can find an opportunity to apply the metric(s) program on a project in industry, then the developer does not need to follow the remaining stages of the preliminary validation process. On the other hand, the developer should not forget to apply similar metrics for comparison while applying the proposed metric(s) to an industrial project, since without comparison the usefulness of a new measure cannot be justified.

# 5 Case study: Applicability for our Model

We have applied our model on the metrics developed by Chidamber and Kemerer [9] and Kim, Shin and Wu [10]. In this section, we want to show that, although both metrics were developed at almost the same time, for evaluating the complexity of OO systems, CK has become a milestone in the field of metrics. Their work is not only used by practitioners and widely adopted by the software industry, but also a simple search by Google can yield 2791 citations. Complexity measures for OO program based on the entropy proposed by Kim et al. is not found to be used by practitioners and has only became a paper which is sometimes used by some new developers of software metrics for citation purposes. We have intentionally used these metrics to evaluate which practices and rules make a metric useful and how our model can be handy while adopting those practices.

We start our evaluation with complexity measures for OO program based on entropy [10]. We have already given introductory information regarding the metrics in Section 3. Here we are evaluating them: how they validated their work? The authors have demonstrated all three metrics: class complexity, inter object complexity and total complexity, with a simple example. This covers the first step of our model.

For the validation of their metrics, they showed that their complexity values fall between $\log_2 n$ and zero [10]. Further, the authors evaluated their metrics theoretically with Weyuker's properties [32]. There is no harm in validating metric(s) theoretically, but theoretical validation only proves that the measure/metric(s) is developed on some sound theoretical base; in no case does it prove any practical applicability in industry. Further, for experimentation, i.e. empirical validation, the authors measured the class complexity for C++ classes using 68 classes that were extracted from classes of user interfaces and data structures [33-36]. We have surveyed these references from books on C++ programming. This part of validation is our third step of the preliminary empirical validation. Furthermore, the authors claim that their experimental results of C++

classes proved the effectiveness of the proposed metrics. They further wrote that in the future, they would gather many OO programs for analysis and for the calculation of the correlation coefficient. This is the summary of their experimentations on their proposed metrics.

As per our empirical validation guidelines, the metrics proposed by Kim, Shin and Wu [10] barely fulfill the preliminary stage of empirical validation. It is because the authors only demonstrated their metric with a small program and then used several classes for showing the implementation of the metrics. Neither did they perform a case study/a real project, nor any comparative study.

Now we evaluate the way the CK metrics suite was introduced. Chidamber et al. [9] firstly provided the fundamental and theoretical background for developing the metrics. All the metrics proposed by Chidamber et al. are straight forward for computing the different features of OO systems, so they are easily countable from any OOS system; e.g. NOC can be easily counted by counting the number of immediate descendants of the class. They compared their metrics with all available OO metrics at that time. If we evaluate their methodology for empirical validation, we find that it closely matches with our advanced empirical validation process. According to Chidamber et al. [9] '*They have applied their metrics through automated tool developed for this research at two different organizations and referred as Site **A** and Site B. Site **A** was a software vendor that uses OOD(object oriented design)..... Metrics data from 634 classes from two C++ class libraries ..... Site B was a semiconductor manufacturer and used the Smalltalk programming language for developing flexible machine control and manufacturing systems. .....Metrics data from 1459 classes from Site B were collected'.*

The quote supports that the metric were developed not only on a sound theoretical background, but also via the adoption of proper validation criteria; i.e. the authors proved the worth of their metrics through the above rigorous empirical validation, which we name as advanced empirical validation in our model. Also, after the development of these metrics, they were adopted by several major organizations, e.g. NASA. Additionally, after gaining popularity, several researchers worked on these metrics and again validated them empirically [37]. This case study proved that the fifth and sixth stage in our model is a necessary requirement for a complete empirical validation process.

# 6 Validation of the Model with Newly Proposed Metrics

We have applied our model to other recently proposed metrics, reported in 2009 and 2010 for a figure of new proposals. As a start, we divided the metrics into two groups: as the first group, we have only considered those metrics which are

published in highly influential journals, especially in science citation indexed/ expanded Science citation indexed. The second group consists of the papers which were published in conferences and other journals. The metrics, under consideration in the first group are cognitive complexity measure (CCM) [20], [38], (2009,) unified complexity measure (UCM) [15] (2010,) complexity metrics for evaluation of metaprogram complexity [5] (2009), package coupling measurement (PCM) in OO Software [39] (2009), and weighted class complexity [13] (2009). The other groups belong to the metrics: a complexity measure based on requirement engineering (2010) [40], OO cognitive-spatial complexity measures [41] (2009), structured software cognitive complexity measurement, [42] (2009).

Firstly, we evaluate the cognitive complexity metric, which were initially proposed in 2004 as cognitive functional size measure (CFS) [20]; later they were promoted by considering the remaining features (explained in forthcoming lines). For CFS, the authors demonstrated their metrics via a case study (a single program in three different languages) and validated their work by applying CFS on a set of 20 programs which were taken from a book. They have also performed a comparative study with a line of code. Later, they extended their work and proposed cognitive complexity measure [38], which was dependent on architectural and operational complexities. The work was validated via demonstration with a case study, examples and a comparative study. This metric (CFS/CCM) satisfies all the steps of preliminary empirical validation; however, it was not implemented in any industry project. Since this metric satisfies only the first stage of empirical validation, acceptance can only be partial.

The second metric under evaluation against our model is a package coupling measurement in OO software [39]. This metric is a coupling metric and takes into consideration the hierarchical structure of packages and the direction of connections among package elements. The metric was demonstrated with examples, theoretically validated and empirically validated by using 18 packages taken from two open source software systems. The authors claimed that they had found a strong correlation between package coupling and the understandability of the package, and hence the metrics could be used to represent other external software quality factors. As per our model, the metric is well supported ($1^{st}$ stage) with example(s); a case study was performed on a java project ($2^{nd}$ stage); it was applied to 18 packages from an open sources ($3^{rd}$ stage); and it was analyzed properly ($4^{th}$ stage). This metric completed all the steps of preliminary stage of empirical validation, except for a comparative study with similar measures. However this metric is not applied in an industry project.

Another metric under examination is the metrics for the evaluation of metaprogram complexity [5]. The authors proposed five complexity metrics: relative Kolmogorov complexity, metalanguage richness, cyclomatic complexity, Normalized difficulty and cognitive difficulty for measuring complexity of metaprograms at information, metalanguage, graph, algorithm, and cognitive

dimensions. For validation of their metrics, the authors demonstrated their metrics with examples/case studies, then applied their metrics on open PROMOL, a metaprograms created from Altera's library for OrCAD VHDL components library. These metrics also passed all the 4 steps of the preliminary empirical validation; however no evidence is found for the implementation of their metrics in industry.

Table 1
Validation of the proposed model against the newly proposed metrics

| | Steps of our model<br>_____<br>Complexity measure | 1 | 2 | 3 | 4 | 5 | 6 | 7(Acceptance) |
|---|---|---|---|---|---|---|---|---|
| Complexity measures form SCIE Journal | Cognitive Complexity Metric | Y | Y | Y | Y | N | N | Partially satisfied |
| | META Program Complexity | Y | Y | Y | Y | N | N | Partially satisfied |
| | Unified Complexity Metric | Y | Y | Y | Y | N | N | Partially satisfied |
| | Package coupling Measurement | Y | Y | Y | Y | N | N | Partially satisfied |
| | Weighted Class complexity | Y | Y | Y | N | N | N | Partially satisfied |
| Complexity metrics published in conferences and non SCI journals | Complexity metric for req. Engg. | Y | N | Y | Y | N | N | Not Recommended |
| | Object-Oriented Cognitive-Spatial Complexity Measures | Y | N | N | Y | N | N | Not Recommended |
| | Towards Structured software Cognitive complexity measurement | Y | N | Y | Y | N | N | Not Recommended |

Our next metric under examination is unified complexity measure (UCM) [15]. This metric includes all major factors responsible for the complexity of a program, including cognitive aspects. The authors claimed that the applicability of the measure is evaluated through empirical, theoretical and practical validation processes. The authors performed test cases and a comparative study. According to our model, UCM also satisfies preliminary empirical validation. UCM was demonstrated with an example, validated theoretically and empirically with test cases/ a number of examples (more than 30) and the developers performed a rigorous comparative study with similar measures. However, it was not applied in industry.

The last metric from the first group under examination is the weighted class complexity (WCC) [13]. The metric was proposed to compute the structural and cognitive complexity of class by associating a weight to the class. The authors claimed that the theoretical and practical evaluations based on information theory

had shown that the proposed metric was on ratio scale and satisfied most of the parameters required by the measurement theory. When we evaluated this metric against our framework, we found that WCC is demonstrated with examples (1st stage), case study was performed (2nd stage), it was also applied on a number of classes (3rd stage), and the authors performed a comparative study with CK metrics suite (4th stage). However, they failed to apply it in a real industry environment.

In [31], a complexity measure based on a requirement engineering document, the authors claim that their paper attempts to empirically demonstrate the proposed complexity metric, which is based on IEEE Requirement Engineering Document [43]. However in summary, they demonstrated their metrics with a single program and then applied it on 16 small programs. They also apply the other metrics on these sixteen programs and concluded their results. Furthermore, neither did they apply their metric on a real example/project, nor did they apply it to any industrial project. As result, this metric achieves only three steps, and even could not cover the steps of preliminary empirical validation process. Hence it does not satisfy the preliminary empirical validation.

In [41], OO cognitive-spatial complexity measures, the authors proposed a metric by combining cognitive and spatial aspects of programming. Further, they claimed that the proposed measures were evaluated using standard axiomatic frameworks (which are Weyuker's properties [32] and Briand's framework [44]), and that they were compared with the corresponding existing cognitive complexity measures as well as the spatial complexity measures for OO software; hence their proposed measures were better indicators of the cognitive effort required for software comprehension than the other existing complexity measures for OO software. Using our model: on the validation part, the authors demonstrated their metric by two programs of 21 and 45 lines, and applied similar metrics on those two programs to perform a comparison. They did not perform any of the following: a case study, test cases, and industrial applications. As result, this metric covers only two steps of preliminary empirical validation; hence it does not pass all steps of our preliminary empirical validation.

In [42], towards structured software cognitive complexity measurement with granular computing strategies, the authors integrated the concept of granular computing and cognitive complexity. They claim that they performed the empirical studies, which were conducted to evaluate the virtue of their metric and also the universal applicability of granular computing concepts. In fact, the authors demonstrated their measure with a short program and then applied it on 12 small programs. This measure satisfies three steps from our model. No case study and test cases were performed; hence all the steps of preliminary empirical validation of our model were not satisfied.

The examination of eight recently developed metrics against our model provides valuable information regarding the actual scenery and facts of software metrics.

1 All the metrics which are published in the SCI/SCIE indexed journals performed better in the empirical validation process.

2 All the metrics in the first group (published in SCI/SCIE journals) passed all the steps of preliminary empirical validations.

3 Most of the developers of the metrics in the first group claimed that they had completed the validation process, except the developer of UCM and WCC. Although they only completed the first phase of validation, i.e. preliminary, they do not have the intention to do later steps in future either. This is because most of the developers are academicians and they do not realize the real needs of the software industry. They may defend their positions by applying their metrics on open source projects and a complete validation process may be achieved; however, this is not sufficient for the acceptance of the metric from the software industry.

4 The metrics in the second group, B (conferences and non SCIE indexed journals), were weak in empirical validation; they did not even perform simple test cases/case studies available on the web.

5 One can find the same evaluation results if he applies our model to most of the available metrics of group B.

The above evaluation validates each step of our proposed model. Most of the metrics in group A satisfy all the steps of preliminary validation process. On the other hand, the failure to satisfy the steps of advanced empirical validation is due to a lack of proper guidelines for the complete empirical validation process. Our model is an attempt to fill this gap by mentioning the need of advanced empirical validation in real industrial environments. In fact, the steps in the model are not new; we have formally integrated what we have surveyed, which can act as a guide to a developer of a new metric. With this model, a new developer can understand all the required steps of complete empirical validation, which can help to gain acceptance of his metric or formula within the industry.

It may be too early to assess the future of all the above metrics, because all of them have only recently been proposed. It is possible that some of them may be evaluated by the industry. On the other hand, in the present scenario, by using our our model, it is proved that none of them satisfies the steps of advanced empirical validation; i.e. they have not been evaluated in the industry. In this respect, there is less chance of the adoption of these metrics in the industry, though they do contribute to the community in the form of research papers. These observations proved our statement that most of proposed metrics are inherently irrelevant to industrial needs, which is because of improper empirical validation.

# 6 Discussion and Recommendation

We want to point out that there is a practical difficulty in advanced empirical process because most of the software industries are likely to be unwilling to apply a new technology/metric since it is difficult to convince them that the metric is more beneficial in comparison to the existing ones. This is one of the reasons why most of the new metrics are not properly empirically validated. Nevertheless, advanced empirical validation is a considerable requirement in validating a new metric and hence we propose the developer of a new metric should follow the following steps:

1 He should prepare a software tool for measuring the complexity value. This tool can be used for the advanced empirical validation as well as for preliminary one. A simulator can also be developed at this stage to help evaluate the new measure before applying it to real industrial data.

2 Based on preliminary observations/result of simulations, a group of practitioners should be appointed for real observations from past sample projects/sub-projects in the industry.

3 This group must apply the proposed technique as well as existing similar metrics on the sample industrial data. It is worth mentioning that once this job is done by practitioners, it will solve the problem of searching for current data from the industry.

4 Further, this group must analyze the results by comparing them with similar metrics. This activity will lead to the evaluation of the proposed metric.

5 At this stage, it is observed that if the developer of the new metric/measure belongs to industry, it is relatively easier for him to follow these steps. On the other hand, if the developer belongs to academia, then again he faces the same practical problem, if the funding is not available. This is actually a common problem, especially in developing countries, and when parental organizations that are not willing to provide any funds for this purpose. Bearing this in mind, we suggest including one practitioner in the starting phase of the proposed metric. His contribution may be in the last phase, which is the most important task for the proof and value of the new metric/measure. It is also worth mentioning that advanced empirical validation may take a few days, weeks or months, depending on the situation (e.g. availability of project) and complexity (number and size) of the proposal.

**Conclusion and Future Work**

It is generally observed that for most of the new metrics/measures, the developer tries to prove his metric to be the most suitable measure for any particular attributes e.g. [20]. The academicians/developers of the new metrics try to prove their claim by evaluating their proposal through different means. These different

evaluating standards can be theoretical validation (for example evaluation through measurement theory), experimentation in the classroom, case studies, different examples from the web, etc. However, they can be essential but not complete. It is proved that none of the newly proposed metrics are validated against an industry project, and hence the chances for the success of these measures are not promising. As a result, without proper preliminary and advanced empirical validation according to our model, any other criteria for the metric validation cannot be effective.

In general, the empirical methods suggest proposing a model, developing statistical / qualitative methods, applying to case studies, measure and analyzing, validating the model and repeating the procedure [45]. All these forms of empirical validation are recommended for any empirical study in software engineering. Our analysis has suggested that all these steps are required for the proper empirical validation of software metrics. Accordingly, we have accumulated them in our model in order to validate software metrics empirically.

**Acknowledgment**
The author is thankful to the Editor and reviewers for their valuable comments. I am also thankful to Dr. Tolga Pusatli for improving English of the paper and several rounds of discussions for finalization of the paper.

**References**

[1] Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El-Emam, K., Rosenberg, J. (2002) Preliminary Guidelines For Empirical Research In Software Engineering, IEEE Transaction on Software Engineering, 28(8), pp. 721-734

[2] Singer J., Vinson N. G.(2002) Ethical Issue In Empirical Studies of Software Engineering, IEEE Transaction on Software Engineering, 28(12), pp. 1171-1180

[3] Brilliant, S. S., Kinght, J .C. (1999) Empirical Research in Software Engineering, ACM Sigsoft, 24(3), pp. 45-52

[4] Zelkowitz, M. V., Wallace, D. R. (1998) Experimental Models For Validating Technology, IEEE Computer, May issue, pp. 23-40

[5] Damaševičius, R., Štuikys, V. (2009) Metrics for Evaluation of Metaprogram Complexity, Journal of Computer and Information science, 16(3), pp. 1-20

[6] Hanny, J. E., Sjoberg D. I. K., Dyba T. (2007) A Systematic review of theory use in software engineering experiments, IEEE Transaction on Software Engineering, 33(2), pp. 87-107

[7] Fenton N. E. (1999) Software Metrics: Success, Failure and New Directions, J.of System and Software, 47(2-3), pp. 149-157

[8] Gopal, A., Mukhopadhyay, T., Krishnan M. S. (2005) The Impact Of Institutional Forces of Software Metrics Programs, IEEE Transaction on Software Engineering, 31(8) pp. 679-694

[9] Chidamber, S. R., Kemerer, C. F. (1994) A Metric Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 20(6), pp. 476-493

[10] Kapsu, K., Shin,Y., Chisu W. (1995) Complexity Measures For Object-Oriented Program Based On The Entropy, In Proceedings of Asia Pacific Software Engineering Conference, 6-9 Dec., pp. 127-136

[11] Zuse, H. (1991) Software Complexity Measures and Methods, de Gruyter Publisher

[12] IEEE Computer Society (1990) IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12 – 1990

[13] Misra, S., Akman, I. (2008) Weighted Class Complexity: A Measure of Complexity for Object Oriented Systems, Journal of Information Science and Engineering, 24, pp.1689-1708

[14] Kushwaha, D. S., Misra, A. K. (2006) Improved Cognitive Information Complexity Measure: A Metric that Establishes Program Comprehension Effort, ACM SIGSOFT SEN, 31(5), pp. 1-7

[15] Misra, S., Akman, I. (2010) Unified Complexity Measure: a Measure of Complexity' The Proc. Nat. Acad. Sci. India, (Sect. A), 80(2), pp. 167-176

[16] Basci, D., Misra, S. (2009) Data Complexity Metrics for Web-Services, Advances in Electrical and Computer Engineering, 9(2), pp. 9-15

[17] Basci, D., Misra, S. (2011) A Metric Suite for Maintainability of XML Web-Services' IET Software, In press

[18] Aggrwal, K. K., Singh Y., Kaur, A., Melhotra, R.(2006) Software Design Metrices for object orinted Software Journal of Object Technology, 6(1), pp. 121-138

[19] Misra, S., Akman, I. (2008) Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings", Journal of Computer and Information Sciences, 15(1), pp. 17-24

[20] Wang. Y., Shao J. (2003) A New Measure Of Software Complexity Based On Cognitive Weights, Canadian Journal of Electrical and Computer Eng., 28(2), pp. 69-74

[21] Wang. Y., Shao J.(2006) Psychological Experiments on the Cognitive Complexities of Fundamental Control Structures of Software Systems, In Proc. IEEE ICCI 2006, pp. 1-2

[22] Basili, V. (2007) The Role of Controlled Experiments In Software Engineering Research, Empirical Software Engineering Issues, LNCS, 4336, 2007, pp. 33-37

[23] Zazworka, N., Basili, V., Zelkowitz, M. V.(2008), An Environment for Conducting Families of Software Engineering Experiments, Advances in Computers,74, pp. 175-200

[24] Pusatli, T., O., Misra, S. (2011) Software Measurement Activities in Small and Medium Enterprises: An Empirical Assessment', Accepted for publication, Acta Poletechnica Hungarica, 4, In Press

[25] McCabe, T. J. (1976) A Complexity Measure. IEEE Transactions Software Engineering, 2(6), pp. 308-320

[26] Shannon, C. E., Weaver, W. (1949) The Mathematical Theory of Communication, Urbana, IL: University of Illinois Press, USA

[27] Davis, J., LeBlanc, R. (1988) A Study of the Applicability of Complexity Measures," IEEE Transactions on Software Engineering, 14, pp. 366-372

[28] Etzkorn, L., Gholston, S., Hughes, W. E. Jr. (2002) A Semantic Entropy Metric, Journal of Software Maintenance And Evolution, 14(4), pp. 293-310

[29] Gaffney, J. (1984) Instruction Entropy, a Possible Measure of Program/Architecture Compatibility, ACM SIGMETRICS Performance Evaluation Review, 12(4), pp. 13-18

[30] Basci, D., Misra, S. (2011) Entropy as a Measure of Complexity of XML Schema Documents' Int. A. Journal Of Information Technology, 8(1), 16-25

[31] Serrano, M., Trujillo, J., Calero, C., Piattini, M. (2007) Metrics for Data Warehouse Conceptual Models Understandability. Information and Software Technology, 49(8), pp. 851-870

[32] Weyuker, E. J. (1988) Evaluating Software Complexity Measure. IEEE Transaction on Software Engineering, 14(9) 1357-1365

[33] Pold, I. (1989) C++ for C Programmers, pp. 156-157, The Benjamin Cummings Publishing Company, Inc.

[34] Douglas A. Y., (1992) Object-Oriented Programming with C++ and OSFMotif, Prentice Hall

[35] Robert L. S. (1992), C++ Component and Algorithm", MNT Publishing Inc.,

[36] Stevens, A. I. (1992) C++ Database Development, MIS Press

[37] Subramanyam R., Krishnan M. S. (2003), Empirical Analysis of CK metrics for Object-Oriented Design Complexity: Implications for Software Defects," IEEE Trans. on Software Engineering, 29(4),297-310

[38] Wang Y., Shao, J. (2009) On the Cognitive Complexity of Software and its Quantification and formal methods. Int. Jour. Of Software Science and Computer Intelligence, 1(2), pp. 31-53

[39] Gupta V, Chhabra J. K. (2009) Package Coupling Measurement In Object-Oriented Software. Journal of Computer Science and Technology 24(2), pp. 1-12

[40] Sharma, A., Kushwaha, D. S. (2010) A Complexity Measure based on Requirement Engineering Document, Journal of Computer Science Engineering, 1(1), pp. 112-117

[41] Gupta, V., Chhabra, J. K. (2009) Object-oriented Cognitive-Spatial Complexity Measures, International Journal of Computer Science and Engineering, 3(2), pp. 122-129

[42] Benjapol, A., Limpiyakorn, Y. (2009) Towards Structured Software Cognitive Complexity Measurement with Granular Computing Strategies, In, Proc. 8th IEEE International Conference on Cognitive Informatics, pp. 365-370

[43] IEEE Computer Society (1994) IEEE Recommended Practice for Software Requirement Specifications, New York

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

[45] Basili V. (1993) The Experimental Paradigm in Software Engineering, Lecture Notes in Compputer Science, 706, pp. 1-7