Product Based Software Metrics: A Review

P.Jha¹, K.S. Patnaik²

¹Department of Computer Science and Engineering
Birla Institute of Technology, Mesra, Ranchi (INDIA)
pooja.jha.ism@gmail.com

²Department of Computer Science and Engineering
Birla Institute of Technology, Mesra, Ranchi (INDIA)
kspatnaik@bitmesra.ac.in

Abstract - This article introduces some of the research carried out in the area of software product metrics. The practices of metrics in software industry and the experiences of some organizations that have developed, promoted, and utilized variety of software metrics is presented here. In order to improve the software quality and the project controllability, it is necessary to control the software complexity by measuring the related aspects. McCabe methods and C&K metric method are used to expound examples of complexity metrics. A two-dimension metrics suite based on complex networks parameters from the perspective of software engineering has been discussed that not only uncover the features of software structure such as scale free, small world, but also can get some latent rules about design and structure. Further, a quality model, distinct metrics and their implementation into a measurement tool for quality management in agile software development is discussed. Also, CCMETRICS, a tool for calculating the coupling and cohesion metrics of object oriented software and its implementation have been discussed. A structured unifying framework for software metrics for use in both instructional and practitioner environments based on the three “primary metrics” has also been discussed. Effective refactoring requires proper metrics to quantitatively ascertain the improvement in the quality after refactoring. An effort to quantitatively evaluate the quality enhancement by refactoring using dependency oriented complexity metrics is shown. Using a neural network, the use of cross-correlations to predict the impact that a given modification on the software solution will have on embedded software physical metrics has been shown. This paper also covers the Taguchi-based software metrics and focuses on the smaller-the-best loss function for estimating the loss to society. Software Code Restructuring is one of pivotal application of software metrics that can insure reliable development of software system. Given a specific target reliability estimate by the customer, a model has been discussed in this paper to software professionals and researchers to identify the phase in which corrective action need to be preformed. This paper also discusses the different strategies on software metrics estimation executed within a large worldwide IT company.

Keywords – Software metrics, complexity metrics, Software Code restructuring, unifying framework, Program Slicing, CCMetrics

I. INTRODUCTION

As per IEEE “standard of software quality metrics methodology”, software metrics is a function with software data as input and output as a value, which can decide on how the given attribute affects the software. Software metrics is very important in research of software engineering and it has developed gradually. The role of metrics in software quality is well recognized. However, software metrics are yet to be standardized and integrated into development practices across software industry. While process, project, and product metrics share a common goal of contributing to software quality and reliability, utilization of metrics has been at minimum. The work by Mauricio J. Ordonez and Hisham M. Haddad [1] is an effort to bring more attention to software metrics. It examines the practices of metrics in software industry and the experiences of some organizations that have developed, promoted, and utilized variety of software metrics. The practice of software measurement is lacking behind and yet to mature enough to be prompted widely across software industry. Experiences indicate when metrics are used early in the development cycle they help detect and correct requirement faults and prevent errors later in the life cycle. Software metrics can be used at each phase of the development. In addition to time, cost, and resource factors, developers are often reluctant to collect and archive project data. Software measurement started in the early 1970s in the US and Canada. The SEI at Carnegie Mellon University helped establish many measurement programs giving a platform to help increase the use of software metrics in software industry. Organizations such as HP, Motorola, NASA, Boeing, AT&T, and others use software metrics extensively. Grady and Caswell [2] categorized metrics as primitive and computed types. Motorola based its metrics program on the paradigm by Basili and Weiss [4]. NASA implements software metrics with emphasis on improving reliability in software requirements specification and source code. Software metrics and error prevention techniques can be applied throughout life cycle phases (requirements, coding and testing) to help improve reliability. Work by Ulisses Brisolara Corrêa, Luis Lamb and Luigi Carro [5], focus on measuring the correlation between object-oriented
quality metrics for traditional software design and physical metrics for embedded systems design. An approach to determine individual suites that fit the projects and do not correlate too highly with each other is cited in [7]. Kiumi Akingbehin, [15] presented some software quality metrics based on the Taguchi definition of quality as “loss imparted to society” after a product is delivered to the end user. Shaikh Mohsin and Zeeshan Kaleem [16] suggested an approach, which explores effective code comprehension by combining Software metrics and technique called Program Slicing. This analytical approach can insure reliable development of software system. The motivation of their research was to develop an approach for automating a large part of program evaluation process by combining the application of program slicing and software metrics. Stevens’ Myers [44] considered coupling as fundamental attribute that significantly impact quality of system. Alghamdi [45] redefined coupling as binary relations on pair of system modules K. Saravanan Kumar and R. B. Misra [17] emphasized it is important to predict the number of faults at each and every phase of software life cycle to know about the reliability. Tu Honglei, Sun Wei and Zhang Yanan [18] discussed the importance of software complexity metrics. They pointed out how these metrics can predict and maintain projects. They stressed that the workload of programming and development cost can be evaluated using these metrics. Results show that dependency oriented complexity metrics have acted as quality indicators with respect to design, considered under ripple effects, before and after refactoring. Metrics as quality indicators help to estimate the required maintenance cost, which is needed for any successful software project. Zhang Haohua, Zhao Hai, Cai Wei, Zhao Ming and Luo Guilan [23] in their paper have proposed an integrated and common metrics suite for large-scale software systems based on complex networks and help developers to improve comprehension, evaluation and quality control. Martin kunz, Reiner R. Dumke and Niko Zenker [27] in their paper aimed to combine refactoring with software measurement. Rodrigo C. Barros, Duncan D. Ruiz, Márcio P. Basgalupp and Karin Becker [29] referred in his paper different strategies on software metrics estimation, presenting a case study executed within a large worldwide IT company. They presented an empirical study on software estimation dealing with three distinct approaches: expert judgment, least square linear regression based models and data mining regression tree models. Work by Sukainah Husein,1, Alan Oxley [49] is based on Briand et al whose framework is one of the few that mentions relationships relevant to cohesion. Briand et al defines two types of cohesion, Data-Data interaction (DD-interaction) and Data-Method interaction (DM-interaction) [37], [50]. A three-layer framework based on function points (FP), person months (PM), and lines of code (LOC) as three primary (or primitive) metrics have been proposed by Kiumi Akingbehin [39]. Earlier work done was on layered aspects of this framework [40], [51]. At the lowest layer are the primary software metrics (plus time) upon which other software metrics are built. At the next higher layer are general-purpose software metrics that are based on the primary metrics. At the highest layer are special purpose metrics. Computed metrics are sometimes used to refer non-primary metrics [41]. An important and useful property of the primary metrics, discussed in the paper is the "convertibility" property, which states that each primary metric can easily be converted to another primary metric. General-purpose metrics or Layer 2 metrics include productivity, cost, documentation, and several others. The special purpose metrics in layer 3 are specific to various areas of software engineering such as software quality, software reliability, time-critical software, mission-critical software, and so on. The unified treatment of software metrics makes it suitable for both instructional and practitioner use. This article discusses the overall view of Software Product Metrics in Introduction Section I. Section II focuses on the methods and techniques applied in the area of software product metrics. Section III and IV deals with the conclusion and future work.

II. METHODS AND TECHNIQUES ADOPTED SO FAR

1. HP software Metrics: HP’s software metrics program served as a model for many organizations and prompted a wide interest among organizations seeking to improve the quality of their products and software development processes. In HP, most widely used computed metrics are:

(i) Metrics for project scheduling cost of defects, workload, and project control for example: Average fixed defects/working day, Average engineering hours/fixed defect, Average reported defects/working day, Defects/testing time, Percent overtime: Average overtime per week. Phase: engineering months/total engineering months.

(ii) End product quality metrics: For example: Defects/KNCSS (Thousand Non-Comment Source Statements), Defects/Lines of Documentation (LOD) not included in the program source code

(iii) Testing effectiveness metrics: Example indicator is Defects/testing time.

(iv) Testing coverage metrics: Example indicator is Branches covered / total branches. This indicates what percentage of the decision points in the program was actually executed.

(v) Useable functions metrics: Example indicator is Bang [3], which is "a quantitative indicator of net usable functions from the user's point of view".

(vi) Productivity metrics: Example indicator is NCCS/engineering month.

2. Motorola software Metrics: Motorola identified a set of goals to increase productivity, improve software reliability, enhance defect containment, and improvement customer service and project planning activities. Reliability, effectiveness throughout the process, delivered defects, accuracy of estimates, and adherences to schedule were among the most important characteristics to measure. A question was
first formulated and a corresponding metric for each goal that aims to improve process quality was done later. Process quality metrics were also developed, including fault type tracking metric, remaining defect metric, and review effectiveness metric

3. **NASA software metrics**: NASA implements software metrics with emphasis on improving reliability in software requirements specification and source code. Software metrics and error prevention techniques can be applied throughout life cycle phases (requirements, coding and testing) to help improve reliability. NASA’S metric tool named ARM (Automated Requirements Measurement) parses requirements document file line by line searching for certain words and phrases also evaluates the structure of the requirements document. It identifies number of requirements at each level of the hierarchical numbering structure. This information helps indicate potential lack of structure that may impact software reliability by increasing the difficulty to make changes. It may also indicate unsuitable levels of details that may constrain software design. NASA’s Software Assurance Technology Center (SATC) uses Weighted Methods per Class (WMC), □ Response For a Class (RFC), □ Coupling Between Objects (CBO), □ Depth In Tree (DIT), Number Of Children (NOC) for object-oriented quality analysis.

4. **Boeing’s software metrics**: Boeing’s 777 program earned the company recognition for achievements through its metrics program, among other related software development initiatives. Experiences with Boeing’s shows that metrics were invaluable for indicting where program risk points are, thus allowing early corrective actions. Development metrics (included software size and number of tests) were used to track progress against the plans for design, code, and testing.

5. **Neural Network Based Metrics**: Ulisses Brisolara Corrêa, Luis Lamb and Luigi, Carro, Lisane Brisolara and Júlio Mattos [5] proposed a neural network to learn from correlations between metrics and estimate the impact of software design decisions on the final system physical metrics. This guides design decisions for improving physical properties of embedded systems, while maintaining an adequate trade-off regarding software quality. Based on the results by Redin [6], and in the capacity of Neural Networks (NN) to learn complex non-linear patterns, their work proposes the usage of NN to accurately predict physical metrics from software metrics. Basically, two concepts i.e. refactoring and the confidence interval of function approximation methods served as the baseline of their study. Assumptions were made that if modifications were made in the source code, and the metrics variation do not exceeds the confidence interval we can expect that the prediction made by these methods is a reliable prediction. The experiments done shows decisions made in the software design phase can impact on physical properties of the final system. The Neural Networks prediction were compared with the statistics multivariate regression prediction, in order to demonstrates that even in cases where the data exceeds the regression confidence limits the Neural Network was able to accurately predict. In the industrial practice, compact metrics suites with high defect predictability are needed, and for the sake of comparability it is convenient to use the same suite for all similar projects in large-scale business architecture.

6. **Metrics for complex software architecture**: Georgios Lajios used a set of candidate metrics containing basic quantitative metrics (like LoC), Chidamber-Kemerer metrics [8], coupling and complexity metrics [9], [12] to determine firstly, which subset of metrics is highly correlated with defect occurrence in a module, but shows only low intercorrelation and secondly, can such a subset be chosen uniformly for structurally similar projects within a company’s architecture? Techniques from machine learning [10], [11] were used for an investigation of the first question, comparing the results provided insight on the second one. The five projects covering the insurance branches having similar functionality were developed under the same technical conditions by different teams. The defects associated with the modules were taken from the bug tracker database during 12 months. The data was analyzed by the machine-learning environment Weka [52] that provides standard implementations for attribute selection. Correlation-based Feature Selection (CFS), a technique aiming at finding a subset of features that are highly correlated with the target attribute, yet uncorrelated with each other was used [13]. A small subset of 4-5 metrics can be found which have good correlation with defect occurrence.

7. **Taguchi quality metrics**: Taguchi defines quality as “loss imparted to society” after a product is delivered to the end user [14][52]. Taguchi most commonly used loss function is the quadratic function commonly known as the nominal-the-best loss function. The loss function can also be determined using a sample from a population:

\[ I_{ave} = k \left[ z^2 + (y_{mean} - t)^2 \right] \]

where:

- \( I_{ave} \) = average (or expected) loss
- \( k \) = quality loss coefficient
- \( z \) = sigma, population standard deviation, not usually known
- \( y_{mean} \) = performance characteristic
- \( t \) = target value

The mean and the standard deviations are given as:

- mean, \( x_{mean} = \frac{\text{Sum } x_i}{n} \)
- std dev, \( z = \frac{\text{Sum } (x_i - x_{mean})^2}{(n - 1)} \)

The smaller-the-best loss function is for a property whose value should be as small as possible. It is obtained from the nominal-the-best loss function as \( L = \frac{k}{y} \). The popularity and widespread use of the nominal-the-best loss function, causes derivation of the
smaller-the-best loss function from the nominal-the-best capitalizing on the existing utilization and effectiveness of the nominal-the-best loss function.

8. **Slice-based metrics:** Shaikh Mohsin and Zeeshan Kaleem [21] work was based on cognitive domains of knowledge, comprehension, Application and Analysis. Program slicing guides the debuggers to understand the program by capturing the computation of a chosen set of variables/functions at some point (static slicing) in the original program or at a particular execution position (dynamic slicing), consequently developing syntactically and semantically simplified version of original program. Program slicing is extensively applied for testing, debugging, maintenance phases of software engineering and reduces the program to identified code at given criteria of function or code. The capacity of Slice based metrics guides the effects of reverse engineering. Program slicing apparently reduces the coupling between modules in the software system. Slice-based metrics can be used to quantify the deterioration that accompanies software evolution. The return on investment in this process can be measured in low cost, fewer test cases per module, and increased overall comprehension and quality. Software systems designs produced by program slicing have low coupling and high cohesion, consequently enhancing quality and reliability.

9. **Reliability based software metrics:** Values of software engineering metrics can help the software professionals to decide whether the necessary amount of corrective actions is achieved or not in order to attain the required target reliability estimate. The enhanced model developed by K. Saravana Kumar and R. B. Misra [17] had a provision to reduce the phase containment of errors in all the early phases of software life cycle. In order to improve the software quality and the project controllability, it is necessary to control the software complexity by measuring the related aspects. Software complexity metrics is the measure of the cost consumed by developing, maintaining, and usage of the software.

10. **Software complexity metrics:** Tu Honglei, Sun Wei and Zhang Yanan [18] related complexity of software directly in affecting the eligibility, reliability of the software. The maintainability of object-oriented software depends on the quality of software. Design quality depends on modifications and enhancements, which are performed during development (agile, iterative) and maintenance because of changing requirements. As part of corrective, perfective, and adaptive maintenance, the software becomes more complex and deviates from its original design in the process of corrective, perfective, and adaptive maintenance, which, in turn lowers the quality of software. A design defect causes the system to exhibit high complexity, improper behavior and poor maintainability [19]. Higher the complexity of software, lower the reliability of software [20]. Object oriented software complexity can be reduced by incrementally improving the internal software quality without affecting the external behavior come under refactoring. The maintainability enhancement by refactoring using different coupling metrics is quantitatively evaluated. After refactoring the coupling values have come down which indicated the improvement in maintainability of software. Using some of the guidelines in the refactoring suggested in [21] refactoring was performed and improvement in coupling and cohesion was evident. Metric based detection strategies for capturing flaws of object-oriented design are defined in the papers [46], [22].

11. **Metrics based on Complex Networks:** The complex network aims to uncover the principles of complex system. Zhang Haohua, Zhao Hai, Cai Wei, Zhao Ming and Luo Guilian [23] in their paper referred that large-scale software systems are treated as SNM (Software Network Model) graphs. According to the experience of complex networks [24], [25], the main parameters of complex networks can be introduced to software engineering, such as k (degree), γ (degree distribution exponent), D(avg shortest path length), C(clustering coefficient), B (betweeness) and Coreness. They proposed a two-dimension metrics suite, which integrates with these metrics and parameters of complex networks from a new point of view. Through experiments of real systems, the feasibility of metrics suite was proved, revealing potential relations between design and implementation. To support agile software development and especially refactoring, mainly source-code based product metrics are beneficial to increase quality and productivity [26].

12. **Unit Metrics:** The metrics should deliver indices for distinct refactoring steps and they should be easily interpretable. The measurement results should be a trigger or activator for useful refactoring steps and they should avoid quality loss through refactoring steps. Martin Kunz, Reiner R. Dumke and Niko Zenker [27] carried out an empirical analysis about the usage and impacts of the tool can be useful to identify the influence of measurement values for the agile software development process. To support agile software development at the origin a tool was implemented as a Plug-In for an Integrated Development Environment (IDE). Eclipse, because of the preconditions and its open source character [28] was selected. Another important implemented feature is the snapshot concept. UnitMetrics Plug-In provides four major functionalities: presentation of metric values normalized metric values and mapping according to threshold of distinct quality models, visualization of package and type dependencies and package distance diagram. The metrics are calculated at method level and without normalization. Without normalization the interpretation is very difficult due to less clarity.

13. **Software Metrics Estimation:** Rodrigo C. Barros, Duncan D. Ruiz, Márcio P. Basgalupp and Karin Becker [29] in their work divided the existent software estimation work in four different classes: Expertise-
based Models, Parametric Empirical Models, Regression Models and Data Mining and Machine Learning Approaches. Decision and regression trees were also explored on software metrics prediction. Their paper describes experiments that compare three different approaches to software metrics prediction: human expertise, linear regression and a data mining regression tree algorithm. HP-EAS was the largest software maintenance project on which the experiment was conducted. The estimation strategy for this project is based on the expert judgment only. They accessed the data repository from ClearQuest, RequisitePro, and the locally developed software tool. They employed the two most commonly used [30], which were also used in [31], [32] and [33]: Mean Magnitude of Relative Error (MMRE) and Prediction at level l. while executing the experiments they identified three hypotheses: H1, H2, H3. Hypothesis H1 and H2 refer to the LSLR statistic strategy, where H1 considers a single-variable LSLR (least square linear Regression), and H2 considers a multi-variable approach. Hypothesis H3 refers to the use of a regression tree algorithm called M5P [34],[35], which is a powerful solution because it combines decision trees and linear regression for predicting a continuous variable, and has the ability of using both continuous and categorical attributes as inputs. The optimal model in hypothesis H1 generated an equation:

\[
\text{Effort} = 95 + 1.61 \times \text{NDOCs}
\]

Where NDOCs = number of documents (NDOCs).

The LSLR NDOCs model did not follow the tendency of the actual effort values, H1 hypothesis was discarded. In hypothesis H2 they observed that the models which included the Baseline Effort were indeed better; because they provided lower MMRE values and more than 40% of the estimates fell within the actual values. With H2 hypothesis it was concluded that it was fully validated, since two different multi-variable LSLR approaches were able to outperform the human expert estimates. With hypothesis H3 it was concluded that the M5P results in terms of effort estimation with more than 45% of its estimates falling within 25% of actual effort values, and MMRE of around 50%.

14. **CCMETRICS**: Coupling between classes exists because one class is relying on the other. Coupling between classes can be reduced by promoting cohesiveness of the class [47]. Coupling and cohesion need to be automatically measured to reduce the measurement effort, subjectivity, and possible errors. Therefore, coupling and cohesion metrics were introduced ranging from syntactic to semantic analysis [36],[37],[48] and software metrics tools were developed e.g. VizzAnalyzer, Analyst[4]; Understand, and Ckjm. Cohesion framework introduced by Eder et al [38], provides a qualitative list that defines the degree of cohesion in an object-oriented environment. Based on the relationships for both coupling and cohesion, metrics were evaluated and chosen. Sukainah Husein and Alan Oxley [49] developed CCMETRICS, a tool for computing coupling and cohesion metrics of object-oriented software. Presently, CCMETRICS handles source code. CCMETRICS allows users to evaluate source codes on package and class level. On package level, metrics of all classes are evaluated and summed together to get package level metrics.

15. **Software metrics framework**: A structured framework for software metrics that was based on the three primary metrics has been proposed Kiiumi Akingbehin [39]. At the lowest layer are the primary software metrics (plus time) upon which other software metrics are built. At the next higher layer are general-purpose software metrics that are based on the primary metrics. At the highest layer are special purpose metrics. Computed metrics are sometimes used to refer non-primary metrics [41]. General-purpose metrics or Layer 2 metrics include productivity, cost, documentation, and several others. The special purpose metrics in layer 3 are specific to various areas of software engineering such as software quality, software reliability, time-critical software, mission-critical software, and so on. The unified treatment of software metrics makes it suitable for both instructional and practitioner use.

### III. CONCLUSION

The various work done on product based software metrics were studied and analyzed. When metrics are used early in the development cycle they help to detect and correct requirement faults and prevent errors later in the life cycle. Metrics can identify potential problems that may lead to errors in the system and decreases over all development cost. The complexity of software will directly affect the eligibility, reliability of the software. Developing an adequate metrics suite for large-scale software systems is still a challenge. Feasibility of the two-dimension metrics suite reveals potential relations between design and implementation, and can facilitate calculating and applying in practical projects. The existing Implementation enhanced the Eclipse IDE to support agile software development. The current version of CCMETRICS handles source code. A structured framework and a three-layer model for software metrics were studied and one of the important property known as convertibility of primary metrics was studied metrics upon which most high level and more complex metrics are built... Elimination of defect and reduction in structure complexity indicate the improvement in quality of software due to refactoring. Dependency oriented complexity metrics have acted as quality indicators. The underuse of the organization metrics repository and the lack of constant feedback from the project team leaders made task not very simpler. The Neural Networks prediction were compared with the statistics multivariate regression prediction demonstrated that even in cases where the data exceeds the regression confidence limits the Neural Network was able to accurately predict physical metrics for embedded systems. To determine custom sets of metrics for quality assessment of complex software architecture, it was found that out that a small
subset of 4-5 metrics could be found to have good correlation with defect occurrence. Experiences with both the nominal-the-best and smaller-the-best loss functions have shown good correlation to existing software quality metrics. Designs of software systems produced by program slicing have low coupling and high cohesion, consequently enhancing quality. The enhanced model by K. Saravana Kumar and R. B. Misra offers a provision to reduce the phase containmment of errors in all the early phases of software life cycle.

IV. FUTURE WORK

More works need to be carried out, including the modeling and evolution of software structure, and the study of how to construct evaluating system about software development. The Eclipse IDE presently supports JAVA as programming language, but enhancement to other programming languages will have more practical benefit. In future, the next version of CCMetrics is expected to calculate the UML representations of software. In continuation of neural network to predict physical metrics for embedded systems, in future aim is to develop an automated tool for design space exploration and automatically applying different refactoring methods in order to find versions of the same application that can present better performance or power consumption, and lower memory footprint. In determining the custom set of metrics for quality assessment of complex system, in future these metrics will be used to pick modules for peer review. Future scope also lies in a detailed empirical study to further validate application of slice-based metrics and promote its implementation on commercial systems. After presenting different strategies on software metrics estimation further work is to study different machine learning and data mining techniques for software metrics estimation that encompass both maintenance and development projects, and to combine different techniques with the aim of generating optimum models.

REFERENCES

Proceedings of ACIS 4th


