

Web Metrics: The way of improvement of quality of Non web-based systems

Shazia Arshad, Muhammad Shoaib and Prof.Dr.Abad Shah
Department of Computer Science and Engineering
University of Engineering and Technology,
Lahore, Pakistan

Abstract

Today, the Internet, web, computer and its applications are the most amazingly and dynamically growing technologies. To improve the quality of web-based software it is important that some measurement methodologies i.e. metrics must be defined those are unavailable till that time. The existing metrics for the different types of systems has been analyzed to take direction for defining web-based metrics. The research can be carried for defining and proving the list of new metrics for web-based systems.

Keywords: - Metrics, on-web based systems, design metrics

1.0 Introduction

Metrics is a word that is used in everyday life in many aspects. Metrics is usually taken as to compare, measure quantity or quality of something with the other. It is concerned with a sort of finding size, amount, quality etc of something in order to compare it with some other thing (i.e. according to standards) and to draw some conclusions on its basis. We draw our conclusions and results of measurements according to some well-defined set of rules. We measure so many things in our daily lives e.g, while buying clothes we measure prices, while doing any equation, we measure its variables, while traveling ,we measure distances etc. Similarly, software are needed to be measured in order to validate reliability, stability, usability, quality and applicability etc. Each and every type of software is measured according to some strategy. Norman Fanton in his article "Software Measurement: A Necessary Scientific Basis" discussed measurement and its features in detail. Measurement is defined as the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. There are two broad types of measurement: direct and indirect. Direct measurement of an attribute is measurement, which does not depend on the measurement of any other attribute. Indirect measurement of an attribute is measurement which involves the measurement of one or more other attributes [8]. Dumke in his article "An Object-Oriented software measurement and evaluation framework" stated that CAME is used to perform software measurement. CAME strategy stands for the following terms: Community: A group of people having enough knowledge of software measurement to install and work on software metrics. e.g German Interest Group on Software Metrics. Acceptance: The acceptance of top management is required to install the software metrics program for the applications. Motivation: In order to motivate all the personnel from top management to the customers of the organization, it is required to present one metrics as an example to get its better understanding for the software being used. Engagement: The acceptance required to implement the software measurement as the permanent metrics. According to Dumke, this

CAME strategy itself consists of the following four phases. Choice: It included the metrics chosen for the measurement. Adjustment: It is concerned with the attributes and characteristics of the metrics to be used. Migration: It shows the way in which all of the metrics are related to each other throughout the life cycle of the software. Efficiency: It is the implementation of the measurement objects

Kelvin defines metrics as “When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and of unsatisfactory kind” The word metrics has been evaluated by the Greek word “metron” which means measure. So, metrics is directly concerned with measurement. Metric system was actually made for measurements and calculation in different fields of life. This system was first introduced and adopted by law in France in 1970s. Later on, with the passage of time a majority of the countries adopted this system for weights and measures. Scientists of the whole world then adopted metric system. Similarly, computer scientists in their work then adopted this system as well and associated it with software which is now known as software metrics.. About 30 years ago, with the evolution and advancement of software engineering, software metrics also made its place in this world. In order to go through ”software metrics”, we have to get better understanding of the word ”software crisis”. According to estimation, by 1990, one half of American work force is relying on computers and software to do their daily work. As demands of new and more efficient applications of software are increasing because of declining rates of hardware, the effort of maintaining it continues to increase as well. On the other hand, there comes the shortage of computer experts and scientists to handle the whole situation. That could show a result of involvement of each and every American to be involved in maintenance and development of software. Software development is sometimes characterized as Schedule and cost estimates which grossly inaccurate.

Software of poor quality .A productivity rate which is increasing slowly than its demands. This situation has been referred to as “ Software Crisis” [Arthur, 1985] Software crisis is needed to be resolved in order to get better and efficient use of computer applications. This requires the software to be better estimated (in terms of cost and time), improved quality and increase performance. All these could be achieved through better maintenance and management of the software, which in turns depends upon proper usage of software metrics. This management needs better measures to improve performance of software. Currently available measures are not that much efficient that they must be. Those metrics does not support decision-making for managers of the organization during the whole software life cycle. Better support for decision-making is required in order to avoid and manage risk. Thus, better measures and parameters and required to ensure managers being able to predict, assess and manage different aspects of the software. This is the goal of software metrics. The identification and measurement of the essential parameters, which affect software development, is the main goal of software metrics. [16]. In fact, Software metrics is a collective term used to describe very wide range of activities concerned with measurement in software engineering[8]. Everal E.Mills in his article” Software Metrics”, defines software metrics as follows “ Software metrics deals with the measurement of the software product and the process by which it is developed”. Software metrics includes models, which are used to measure productivity and improve quality of the product. So, metrics must be accurate complete and well defined. Thus, ideal metrics should be: (a) Simple, precisely definable-so that it is clear how the metric can be evaluated. (b) Objective, to the greatest extent possible. Easily obtainable(i.e. at reasonable cost) (c) Valid-the metric should measure what is intended to measure; and Robust-relatively insensitive to insignificant in the process of product[10]. It has been observed that fundamental qualities required of any technical system are (a): Functionality-correctness, reliability etc. (b) Performance-response time, throughput, speed etc (c) Economy-cost effectiveness.

From above only “a” and “b” is included in software metrics. Performance is not generally included in its discussions. The evaluation of performance is often treated extensively in performance evaluation studies. So. It is not included in studies of software metrics [5]

Software metrics are related to the four important phases of software development phases. Planning - Metrics serve as a basis of cost estimating, training planning, resource planning, scheduling, and budgeting. Organizing - Size and schedule metrics influence a project's organization.

Controlling - Metrics are used to status and track software development activities for compliance to plans. Improving. - Metrics are used as a tool for process improvement and to identify where improvement efforts should be concentrated and measure the effects of process improvement efforts. There are two steps to follow for program metrics. The first component of a metrics program is an infrastructure which describes the metrics to collect, how to collect them, and how to use them. There are eight steps used to develop this infrastructure for the measurement of analysis and development of software product. The major two objectives of applying these eight steps are as follows. (a) The goals of establishing the metrics program must be documented. This is to facilitate the clear view of the main objective to be accomplished rather than just start using the resources for its development. (b) All of the needed information for the goal to be achieved and its framework must be identified. In order to accomplish these tasks the following eight steps are taken under consideration. (a) Document the Software Development Process (b) State the Goals (c) Define Metrics Required to Reach Goals (d) Identify Data to Collect (e) Define Data Collection Procedures (f) Assemble a Metrics Toolset (g) Create a Metrics Database (g) Define the Feedback Mechanism. The second component of metrics program is the method to follow to apply the metrics infrastructure (the first component) to projects in a company. This method is called as Project Measurement Cycle and is accomplished in the following parts. choose a project to measure, build team awareness, measure the project, prepare the results, present results and collect feedback, implement changes, measure again. There is some preliminary groundwork to cover before starting a metrics program: Taking the time to perform these tasks first may save a lot of time and trouble in the long run, identifying a sponsor, selling the program to senior management, creating a measurement team, determining how the metrics program will be documented and communicated to other staff members.

2.0 Types of Software Metrics

Based on articles of Bollmann and Cherniavsky, in 1984, in which measurement theory was applied to evaluation measures in the area of information retrieval systems, in 1985, Bollmann and Zuse transferred this approach towards software complexity measures. They used measurement theory that was being proposed by Roberts in 1979, giving conditions of measures. Later on, in 1991, by Zuse & Bollman, the conditions for software complexity measures were being presented. So, the work carried on upto now, where experts are working and trying to propose new strategies and methodologies for measurements in software industries. Dumke classified software metrics on the basis of measurement theory at three levels, given as follows: Product metrics: This metrics is classified into the following 3 types, each further having its own extension. 1: Architecture metrics: component metrics, configuration metrics, data-base metrics 2: Run-time metrics: task metrics, data handling metrics, human interface metrics 3: Documentation metrics: manual metrics, development metrics, marketing metrics Process metrics: This metrics also classified into 3 types each having extensions: 1: Management metrics: project metrics, configuration metrics, SQA metrics 2: Life cycle metrics: phases metrics, milestone metrics, workflow metrics 3: CASE metrics: method, metrics, paradigm metrics, tool metrics Resources metrics: 1: Personnel metrics: skill metrics, communication metrics, productivity metrics 2: Software metrics: paradigm metrics, performance metrics, replacement metrics 3: Hardware metrics: reliability metrics, performance metrics, availability metrics [11].

3.0 Different Approaches to Software Metrics

Software design metrics research has been ongoing at Ball State University since 1989. The main purpose of the whole research work was that software developers should be able to infer more about the software they are developing during the design process. Thus, during the development of design, computing various metrics helps software developers and managers to choose among several available designs. It also helps them to predict early in time the difficulties and complications, which may occur in software development.

Ivory, Sinha & Hearst in their article “Empirically Validated Web Page Design Metrics” stated that there is currently much debate about what constitutes good web site design . Many detailed usability guidelines have been developed for both general user interfaces and for web page design . However, designers have historically experienced difficulties following design guidelines. Design metrics are very important in order to validate your software against certain standards of software development. It helps you to detect where certain design rules are violated. Experts have divided design metrics into the following types:

3.1 Basic Metrics: These metrics are concerned with the language elements in which the program has been written.

3.2 Quality Metrics: Software engineering and other design principles are of major concern in these metrics. Stability Metrics: These metrics are concerned with stability of the packages.

Robert C. Martin in his article ”Design Principles and Design Patterns” discussed the problems associated with the software being implemented in companies and all other areas of technology. What goes wrong with software? The design of many software applications begins as a vital image in the minds of its designers. At this stage it is clean, elegant, and compelling. It has a simple beauty that makes the designers and implementers itch to see it working. But then something begins to happen. The software starts to rot. At first it isn’t so bad. An ugly wart here, a clumsy hack there, but the beauty of the design still shows through [10]. Robert C. Martin wrote the symptoms that are associated with the rotting design of software and are not orthogonal. These are given as follows: Rigidity It is the tendency of the software resistant to change. Every change causes a cascade of subsequent changes in dependent modules. A simple two day change is prolonged to many weeks to occur. In such situations, managers are reluctant to allow engineers to fix the non-critical problems because they don’t know about the completion of the engineers and thus they refuse any change to be occurred in the project. Fragility :It is much similar to rigidity because it causes the software to break down at many places where the change has been occurred. Often the breakage occurs in areas that have no conceptual relationship with the area that was changed. Such software is impossible to maintain. Every fix makes it worse, introducing more problems than are solved [10].

3.3 Immobility: It is the inability to reuse the part of the project by other projects or other parts of the same project. The same module is needed by more than one engineers at the same time. After much work, the engineers discover that the work and risk required to separate the desirable parts of the software from the undesirable parts are too great to tolerate. And so the software is simply rewritten instead of reused. Viscosity: It has two forms: Viscosity of design and viscosity of the environment. When faced with a change, engineers usually find more than one way to make the change. The viscosity of design is increased when the methods of design preserving are very complicated to be implemented. While when the environment is inefficient and slow then the viscosity of environment is increased. Changing Requirements: Things are changed in a way not previously defined which causes rapid changing problems and are made by engineers not properly trained for such tasks. This makes the improper change in design Dependency Management: Changes causing the rotting design are those that are new and unplanned for dependencies. Each of

the four symptoms mentioned above is either directly, or indirectly caused by improper dependencies between the modules of the software. This management consists of the creation of dependency firewalls. Principles of Object Oriented Design Metrics: The Open Closed Principle (OCP) It means simply this: We should write our modules so that they can be extended, without requiring them to be modified. It was first introduced by Bertrand Meyer. In other words, we want to be able to change what the modules do, without changing the source code of the modules. Many techniques are used for OCP and all are based upon abstraction and these are as Dynamic Polymorphism and Static Polymorphism. The main advantage of OCP is that it makes modules extensible without being changed. The Liskov Substitution Principle (LSP) It states that “Derived classes should be substitutable for their base classes. That is, a user of a base class should continue to function properly if a derivative of that base class is passed to it.” It was introduced by Barber Liskov from her work for abstraction. The Dependency Inversion Principle (DIP) If the OCP states the goal of OO architecture, the DIP states the primary mechanism. Dependency Inversion is the strategy of depending upon interfaces or abstract functions and classes, rather than upon concrete functions and classes[10]The Interface Segregation Principle (ISP) It is very simple. If you have a class that has several clients, rather than loading the class with all the methods that the clients need, create specific interfaces for each client and multiply inherit them into the class.

3.0 Need of Software Metrics in the Software Development Life Cycle

Just like all other fields of life, we need measurements in software engineering as well. Software engineering is the type of field which includes designing, analysis, costs, plans, implementation and testing. All these tasks are required to be done accurately in order to develop and implement a successful software. We need to measure our software in order to assess its status and quality. Tom Demarco, asserts that “ You cannot control what you cannot measure” [17]. Software engineering is a challenging as well as essential part of a high quality software engineering culture. Software needs measurements that must be clearly defined and easily understandable. Managers, engineers and other persons associated with the software project use measurement as follows. Software managers use measurement to find out the time and effort needed to complete the tasks during development of software. They measure the time taken by the staff to design, code, and test the system. They measure functionality of their system in order to validate it according to customer’s requirements. While measuring the software, different comparisons are made and these comparisons help afterwards to adopt changes required to improve the software. Software engineers measure the requirements in order to find out that whether they are testable or not. We can reach to the root causes of failures of code, design or implementations through effective measurements of these phases. Engineers compare software with other software in order to find out that whether their software achieves certain goals in the given amount of time or not. They can also use it to predict future actions of their software as well. Fenton & Pfleeger stated that, in short, measurement is helpful to understand, control and improve our software. Karl E. Wiegers in his article “ A Software Metrics Primer” stated that software measurement helps you to quantify many aspects of your project such as estimation of cost, time and effort schedules, product size and the important thing, the overall performance of your project. Experts have to measure many aspects of the software and that’s why Wiegers gave a healthy tip to make an easy decision. He suggested to chose a small and balanced set of metrics that will prove to be helpful to direct your measurement towards achieving goals. This set of metrics must include size of product and its quality, work effort, quality of the process, project status and the most important, customer satisfaction. Horst Zuse in his article “History of Software Measurement” discussed the need of measurement in future as well as present. He stated that we need software measures to drive : (a). A basis for estimates, (b). To track project progress, (c). To determine (relative) complexity,(d). To help us to understand when we have archived a desired state of quality,(e). To analyze or defects(f). and to experimentally

validate best practices(g). In short: they help us to make better decisions[16]. Karl E. Weigers_ in his article “Software Metrics: Ten Traps to Avoid” discussed the purpose of using measurement theory in software engineering. He gave the following reasons to make measurement necessary for the development and successful implementation of any project. (a) To collect objective information about the current state of a software product, project, or process. (b) To allow managers and practitioners to make timely, data-driven decisions. (c) To track your organization's progress toward its improvement goals. (d) To assess the impact of process changes.

5.0 Conclusion

In this paper, we investigated a number of design metrics and our study showed that the most significant role in design metrics for web based metrics is its complexity measures both for static as well as dynamic pages. We have reviewed object oriented and other design methodologies and concluded that these methodologies can be later on proved to be useful to make flexible, less rigid, less viscous and easily implemented web based design metrics.

6.0 Future Work

Research can be carried out to define and enhance the web-based metrics those should be capable to accommodate the web-based systems. An experiment bed or case study to prove the reliability of those metrics can be established.

7.0 References

[1] Anie Mitchell, James F. Power” Toward a Definition of run-time object-oriented metrics”, 7th ECOOP workshop on quantitative approach in object-oriented software engineering, 2003.

<http://www-ctp.di.fct.unl.pt/QUASAR/QAOOSE2003/papers/Mitchell.pdf>

[2] Fernando Brito e Abreu” Design Quality Metrics For Object-Oriented Software Systems”.

ERCIM news no.23,1995.

[3] G. Booch “Object Oriented Design with applications”, Redwood City, CABenjamin1991. <http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=82528>

[4] Horst Zuse in his article “History of Software Measurement” Published on 14th September 1995 <http://irb.cs.tu-berlin.de/~zuse/metrics/3-hist.html>

[5] Karl E. Wiegers “ A Software Metrics Primer”

http://www.processimpact.com/articles/metrics_primer.pdf

[6] Karl E. Weigers_ in his article “Software Metrics: Ten Traps to Avoid” published in *Software Development*, October 1997

[7] Linda H. Rosenberg & Lowerence E. Hyatt “ Software Metrics Program For Risk Assessment” presented at the 47th International Astronautical Congress & Exhibition at October 1996.

[8] Norman Fenton” Software Measurement: A Necessary Scientific Basis” IEEE Transactions on Software Engineering. Vol. 20, NO. 3, MARCH 1994

[9] Peter Barna, Flavius Frasinca, Geert-Jan Houben, and Richard Vdovjak Technische Universiteit Eindhoven “Methodologies for Web Information System Design”.

<http://wwwis.win.tue.nl:8080/~hera/papers/ITCC2003b/itcc2003b.pdf>

[10] Robert C. Martin_”Design Principles and Design Patterns” published in 2000. <http://www.objectmentor.com>

[11] R. Dumke "An Object-Oriented software measurement and evaluation framework” University of Magdeburg. Available on

<http://ivs.cs.uni-magdeburg.de/sw-eng/us/menue/frameworksA.html>

[12] R. Harrison, S. Councill and R. Nithi ” An Overview of Object-Oriented Design

Metrics”.

In proceedings of Empirical Assessment in Software Engineering (EASE)’, 1997.

[13] S. R Chidamber and C.F Kemerer “ A metrics suite for object oriented design”available on <http://ieeexplore.ieee.org> ,Vol 20, June 1994.

[14] Vili Podgorelec, “ Software Complexity” 14th April, 1997.

[15] W.N. Mills III, L. Krueger, W. Chiu, N. Halim, J.L. Hellerstein, M.S.Squillante
”Metrics for Performance Tunning of Web-Based Applications” IBM Corporation,
Proceedings of the 1999 Computer Measurement

[16]W.N. Mills, L. Krueger, W. Chiu, N. Halim, J.L. Hellerstein, M.S. Squillante “Metrics
For Performance Tuning Of Web-Based Applications”

[17] Y. Ivory , Rashmi R. Sinha, Marti A. Hearst “Empirically Validated Web Page Design
Metrics”Psychology Department/EECS Department, UC Berkeley. Appearing in ACM
SIGCHI'01,March 31-April 4, 2001, Seattle, WA, USA