Software Metrics: An Essential Tool for determining software success

Ruchi Yadav & Pramod Kumar
Dept. of Information & technology, Dronacharya College of Engineering
Farruhknagar, Gurgaon, India
Email:ruchiyadav477@gmail.com

Abstract

This paper describes software metrics as an important and essential tool for measuring success of various software's in various organizations. Software Metrics are tools for anyone involved in software engineering to understand varying aspects of the code base, and the project progress. In regards to software project cost and underestimation, it is a problem that has not diminished in the last 70 years. The Standish Chaos Report (2004) found only 29% of project met their criteria for project success: projects that were on budget, on schedule, and with the expected functionality. The Standish Chaos Report also estimated that the annual cost of cancelled projects was $55 billion. It helps predict defects in code and can be used to determine code quality. The process of software development, including documentation, design, program, test, and maintenance can be measured statistically. Therefore the quality of software can be monitored efficiently. Software metrics is very important in research of software engineering and it has developed gradually. In this paper, software metrics definition were given and the history of and the types of software metrics were overviewed. Software complexity measuring is the important constituent of software metrics and it is concerning the cost of software development and maintenance. In order to improve the software quality and the project controllability, it is necessary to control the software complexity by measuring the related aspects.

1. Introduction

"What is not measurable, make measurable", the great Galileo Galilee had said. Measurement has always been fundamental to any engineering discipline and software engineering is no exception. This is how Pressman [8], introduces metrics. So what kind of measurement is he talking about? Obviously it should be something that gives us the ability to evaluate software process – the design, the code, the testing, etc. But does it end there? Probably not. Metrics also cover the aspect of evaluating the final software product and a lot more. Several companies have implemented metrics programs to support the managers in their decisions. However the benefits from the implementation are not as great as expected. Nearly 80% of software metrics programs fail within the first two years (Dekker's, 1999).Most currently used metrics concentrate on the latter stages of development-coding and testing. The author proposes a shift in focus toward using metrics during the high-payoff phases of software development-requirements definition and design. A general methodology that can assist the software manager and developer in creating an effective software metrics program is offered. This will increase understanding and improvement of the software development process at an early stage in the software life cycle. It is noted that early use of software metrics will allow the manager

to spend more time on error prevention and less on error correction. This should yield cost and time savings, and an improvement in software product quality.

Software metric is a measure of some property of a piece of software or its specifications. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. The goal is obtaining objective, reproducible and quantifiable measurements, which may have numerous valuable applications in schedule and budget planning, cost estimation, quality assurance testing, software debugging, software performance optimization, and optimal personnel task assignments.

2. Software Metrics

2.1 Definition of Software Metrics

The definition of software metrics has taken various forms since its inception. Metrics are quantitative measures that enable software people to gain insight into the efficacy of software process and also pinpoint problem areas . They provide requisite information for quantitative managerial decision making as well as support for risk assessment and reduction. They can be considered an objective mathematical measure of software that is sensitive to difference in software characteristics. According to the IEEE standard glossary of Software Engineering Terms ,they are "a quantitative measure of the degree to which a system, component, or process possesses a given attribute.".

Software Metrics are tools for anyone involved in software engineering to understand varying aspects of the code base, and the project progress. They are different from just testing for errors because they can provide a wider variety of information about the following aspects of software systems:

- Quality of the software, different metrics look at different aspects of quality, but this aspect deals with the code.
- Schedule of the software project on the whole. I.e. some metrics look at functionality and some look at documents produced.
- Cost of the software project. Includes maintenance, research and typical costs associated with a project.
- Size Complexity of the software system. This can be either based on the code or at the macro-level of the project and its dependency on other projects.

Software metrics are used to obtain objective reproducible measurements that can be useful for quality assurance, performance, debugging, management, and estimating costs. Finding defects in code (post release and prior to release), predicting defective code, predicting project success, and predicting project risk. There is still some debate around which metrics matter and what they mean, the utility of metrics is limited to quantifying one of the following goals: Schedule of a software project, Size/complexity of development involved, cost of project, and quality of software. Almost every metric has one thing in common – the motivation behind it; this could be assessing cost and effort to be put in or assessing the quality of the software. For example, one of the earliest metrics to measure code efficiency – the Lines of Code (LOC) metric has been used in a model as simple as:

Effort = f (LOC)

There have been various other attempts to use, for example, LOC (and other naive metrics) as a metric to measure aspects such as effort, complexity, etc. It is pretty obvious that with the advent of newer programming languages these models were not going to work because it does not make sense, for instance, to compare LOC values of an assembly language program with a high-level language program. Thus researchers started working on coming up

with metrics that were independent of the language used.

2.2 Evaluation of Software Structure Metrics

Structured design methodologies provide a disciplined and organized guide to the construction of software systems. However, while the methodology structures and documents the points at which design decisions are made, it does not provide a specific, quantitative basis for making these decisions. Typically, the designers' only guidelines are qualitative, perhaps even vague, principles such as "functionality," "data transparency," or "clarity." This paper, like several recent publications, defines and validates a set of software metrics which are appropriate for evaluating the structure of large-scale systems. These metrics are based on the measurement of information flow between system components. Specific metrics are defined for procedure complexity, module complexity, and module coupling. The validation, using the source code for the UNIX operating system, shows that the complexity measures are strongly correlated with the occurrence of changes. Further, the metrics for procedures and modules can be interpreted to reveal various types of structural flaws in the design and implementation.

2.3 Need for Software Metrics

Having looked at some basics of software metrics, the next question that arises in many a budding software engineering's mind is why do we need metrics? And if they are indeed useful to their cause? The answer to the latter question in short is yes. Software metrics have been proven to be useful if applied in the right way – this needs to be stressed. Application of wrong methods might lead to failure of a metric but it is not really the metric which is to be blamed! Here we state some arguments as to why we need software metrics:

- Without measuring software process or the quality of an end product, only subjective evaluation is possible.
- Not desirable.
- With robust measurements.
- Requirements can be assessed better.
- Error prone components can be identified at early stages.
- Quality assurance can be improved.
- Predicting resource requirement is another important use of software metrics.

The issue of integration problem is discussed by Sedigh-Ali et al., where the complexity of interfaces and their integration is interpreted as quality metrics. Cho et al define a metrics for complexity, customizability and reusability. They count the complexity of metrics by using the combination of the number of classes, interfaces, and relationship among classes. They also combine the calculation of cyclometric complexity with the sum of classes and interfaces.

Our work in this area indicates that the suite of metrics that we have defined could be of substantial use in estimating the effectiveness of the overall integration process, during the specification and design stages. As a consequence, a software developer does not have to wait until the programming stage to get any estimation. We intend to consolidate our work and validate the metrics suite through actual experimentation and analysis of the results. The suite will also integrate existing metrics available in the literature.

3. Classification of Software Metrics

There are three types of software metrics: process metrics, project metrics and product metrics.

1) Process Metrics:

Process metrics highlights the process of

software development. It mainly aims at process duration, cost incurred and type of methodology used. Process metrics can be used to augment software development and maintenance. Examples include the efficacy of defect removal during development, the patterning of testing defect arrival, and the response time of the fix process.

2) Project Metrics:

Project metrics are used to monitor project situation and status. Project metrics preclude the problems or potential risks by calibrating the project and help to optimize the software development plan. Project metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

3) Product Metrics:

Product metrics describe the attributes of the software product at any phase of its development. Product metrics may measure the size of the program, complexity of the software design, performance, portability, maintainability, and product scale. Product metrics are used to presume and invent the quality of the product. Product metrics are used to measure the medium or the final product. We can find more efficient ways of improving software project, product and process management.

3.1 Mathematical Analysis

A metric has a very explicit meaning in mathematical analysis .It is a rule used to determine distance between two points. More formally, a metric is a function‘ d ’ defined on pairs of objects p and q such that d (p, q) expresses the distance between p and q. Such metrics must satisfy certain properties:

d (p,p) = 0 for all p : that is, the distance from point p to itself is zero;

d (p, q) = m (q, p) for all p and q: that is, the distance from p to q is similar to the distance from q to p;

d (p, r) ≤ d (p, q)+d (q, r) for all p, q and r: that is, the distance from p to r is no larger than the distance measured by stopping through an intermediate point.

A prediction system comprise of a mathematical model along with a set of prediction processes for determining unknown parameters and depicting the results. The model should not be complicated for use. Suppose we want to predict the number of pages, P that will print out as a source code program, so that we can bring sufficient paper or calculate the time the program will take for printing. We can use a simple model,

$$P = x/a \quad (1)$$

Where x is a variable, acts as a measure i.e. length of source

Code program in LOC (line of code), and ‘a’ is a constant that represents the average number of lines per page. There are number of models to determine effort estimation; from analogy based estimation to parametric models. A generic model can be used to estimate effort predication.

$$E = aS^{b} \quad (2)$$

Where a and b are constants. E is effort in person-months.
S is the size of source code in Line of code.

3.2 Importance of Software Quality

In recent times the importance of software quality has come to light when random errors on a say a telephone bill, or on a bank statement were randomly attributed to a bug in the “computer code” or using the ignorant adage of “the computer does things” without making an effort to undermine the cause of the problem or even separating it by hardware or

software. The problem arises when "computer errors" creep into highly critical aspects of our lives involving situations where a small error can lead to a cataclysmic chain of events. Bearing all this in mind, the importance of enforcing software quality in computer practices has become highly important. Seeing the penetration of computer code into everyday objects like washing machines, automobiles, refrigerators, toys and even things like the mars rover, any system be it a large one or a small system running embedded IC technology, ensuring the highest levels of software quality is paramount.

Software quality, as stated earlier, depends on a number of factors. Also as theorized by David & Garwin, quality is a complex as well as multifaceted concept, which can be viewed according to different points of view as follows

1) User View:

The user viewpoint of software quality tends to be a lot more concrete and can be highly subjective depending upon the User. This view evaluates the software product against the user's needs. In certain types of software products like reliability performance modeling and operational products, the user is monitored according to how they use the product.

2) Manufacturing View:

This viewpoint looks at the production aspect of the software product. It basically stresses on enforcing building the product without any defects and getting it right the first time rather than subsequently making a defective product and spending valuable project time and more importantly costs ironing out the defects or bugs at a later stage. Being process based, this viewpoint focuses on conformity to the process, which will eventually lead to a better product.

3) Product View:

The product viewpoint looks at the internal features as well as the characteristics of the product. The idea behind this Viewpoint is that in case a product is sound in terms of the features and functionality it offers, and then it will also be favorable when viewed from a user viewpoint in terms of software quality. The idea is that controlling the internal product quality indicators will influence positively the external product behavior (user quality) There are models trying to link both the views of software quality but more work is needed is this area.

4) Value based view:

The value-based view becomes important when there are lots of contrasting views, which are held by different Departments in an organization. For example, the marketing department generally takes a user view and the technical department will generally take a product-based view. Though initially these contrasting viewpoints help to develop 360-Degree product with the different viewpoints complementing each other, the later stages of the software product development might have issues.

4. Validation of Software Metrics

With the plethora of metrics proposed it is critical that these metrics are thoroughly validated it the help of past experiences and new test data. There are many views on how this validation should be carried out. For example, Ejiogu suggests that since metrics touch both structured programming and mathematical measure theory, the two need to be combined when validating software metrics. He goes on to expand on this point in the paper. This usually metrics are validated using simple regression or linear rank correlation techniques. These have proved to be effective in a lot of simple cases.

But as the complexity of software grows

and also as more and more metrics are proposed, these simple techniques have been looked at, with more skepticism. This has led to more research in this area of validating software metrics, some of which we will discuss in this paper. The reason why regression based models may fail in validation of software metrics can be explained by Fenton all's analogy. They argue that regression models lead to misleading results and cite a road accident analogy – just claiming that winter is the best time to drive is flawed if the decision is just based on lower number of accidents; the fact that lesser number of people actually drive in winter conditions is very crucial.

5. Comparison of Software Metrics-Strengths and Weaknesses

The software industry does not have standard metric and measurement practices. Most of the software metric has multiple definitions and ambiguous rules for counting. There are also important subject issues that do not have specific metrics, such as quantifying the volume or quality levels of databases, web sites and data warehouses. There is a lack of strong empirical data on software costs, schedules, effort, quality, and other tangible elements, which results in metric problems.

5.1 Source Code Metrics

"Source lines of code" or SLOC was the first metric developed for quantifying the outcome of a software project. The divergent "lines of code" or LOC has similar meaning and is also widely acceptable. "Lines of code" could be defined either:

- A physical line of code.
- A logical line of code.

Physical lines of code are sets of coded instructions terminated by hitting the enter key of a keyboard. Physical lines of code and logical lines of code are almost identical for some languages, but for some languages there can be considerable differences. Generally, the difference between physical lines of code and logical lines of code is often excluded from the software metrics literature.

Strengths of physical lines of code (LOC) are:

- It is easy to measure.
- There is a scope for automation of counting.
- It is used in a verity of software project estimation tools.

Weaknesses of physical LOC are:

It may include significant "dead code." It may include white spaces and comments. This metric is vague for software reuse.

6. Conclusions

With the rapid advancement in software industries, software metrics have also developed fast. Software metrics become the basis of the software management and crucial to the accomplishment of software development. It can be anticipated that by using software metrics the overall rate of progress in software productivity and software quality will improve. If relative changes in productivity and quality can be determined and studied over time, then focus can be put upon an organization's strengths and weaknesses. Although people appreciate the significance of software metrics, the metrics field still needs to mature. Each of the key software metrics candidates has broken into many competing alternatives, often following national restrictions. There is no adequate international standard for any of the extensively used software metrics. Absence of firm theoretic background and t h e assurance of methods, software metrics are still young in comparison of other software theories.

7. References

[1] Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, 1996

[2] "Software Quality Metrics for Object Oriented System Environments", June 1995, National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt Maryland

[3] Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, A Min Tjoa, Stefan Biff,"Defect Prediction using Combined Product and Project Metrics ", March 2008.

[4] J.E. Gaffney, Metrics in software quality assurance, Proceedings of the ACM CSC-ER '81 conference, pp 126-130, 1981