The Applicability of Existing Metrics for Software Security

Sree Ram Kumar T
Research Scholar
Madurai Kamaraj University
Madurai, India.

Sumithra A
Research Scholar
Madurai Kamaraj University
Madurai, India.

Alagarsamy K
Associate Professor
Madurai Kamaraj University
Madurai, India.

ABSTRACT
With the increasing inclination of people to use software systems for most of the purposes, comes a major challenge for software engineers – the engineering of secure software systems. The concept of "Computer Security" is being heavily researched and this perfectly makes sense in a world where e-commerce and e-governance are becoming the norms of the day. Along with their potential for making life easier and smarter for people, these systems also carry with them the danger of insecurity. Because any software system is an outcome of some software engineering process it makes sense to incorporate security considerations during the software engineering processes. This is easier said than done because traditional software engineering approaches are requirements driven and pay very little, if any, attention to security. Tom DeMarco [1] stated, "You can't control what you can't measure." This clearly states the importance of metrics in software engineering. Traditional software metrics do not address the issue of security well and now with security becoming an imperative necessity of most software systems, these metrics have to be adapted to take into account the security aspect. The paper discusses the applicability of some established metrics for the security aspect.

Keywords
Metrics, Security, Security Metrics, Size Metrics, Complexity Metrics

1. INTRODUCTION

Tom DeMarco [1] stated, "You can't control what you can't measure." This clearly states the importance of metrics in software engineering. Since quantitative methods have proved so powerful in other sciences, computer science practitioners and theoreticians have worked hard to bring similar approaches to software development. Eventhough many software metrics are now available, most of the metrics have lacked a sound theoretical basis or a statistically significant experimental validation. Despite these problems, it appears that the judicious methodical application of software metrics can aid significantly in improving software quality and productivity. Engineering of secure software systems seems to be one of the most important challenges confronted by software practitioners today and hence it is worth exploring the possibility of using metrics to aid the software engineers in this regard.

1.1 The Need of Software Metrics

Current software management is ineffective because software development is extremely complex, and we have few well-defined, reliable measures of either the process or the product to guide and evaluate development. Thus, accurate and effective estimating, planning and control are nearly impossible to achieve [2]. Improvement of the management process depends upon improved ability to identify, measure and control essential parameters of the development process. This is the goal of software metrics—identification and measurement of the essential parameters that affect software development.

1.2 Software Metrics Defined

It is important to further define the term software metrics as used in this paper. Essentially, software metrics deals with the measurement of the software product and the process by which it is developed [3]. In this discussion, the software product should be viewed as an abstract object that evolves from an initial statement of need to a finished software system, including source and object code and the various forms of documentation produced during development. Ordinarily, these measurements of the software process and product are studied and developed for use in modeling the software development process. These metrics and models are then used to estimate/predict product costs and schedules and to measure productivity and product quality. Information gained from the metrics and the model can then be used in the management and control of the development process, leading, one hopes, to improved results.

1.3 Security Metrics Defined

We shall adopt the following definition of "Security Metrics"

"At a high-level, metrics are quantifiable measurements of some aspect of a system or enterprise. For an entity (system, product, or other) for which security is a meaningful concept, there are some identifiable attributes that collectively characterize the security of that entity. Further, a security metric (or combination of security metrics) is a quantitative measure of how much of that attribute the entity possesses. A security metric can be built from lower-level physical measures." [4]

1.4 Characteristics of ideal/good metrics

Ideal metrics should be :

- Simple, precisely definable – so that it is clear how the metric can be evaluated
- Objective, to the greatest extent possible;
- Easily obtainable
- Valid – the metric should measure what it is intended to measure
- Robust – relatively insensitive to insignificant changes in the process or product.

Ferrari [17] observes that the fundamental qualities required of any technical system are :

- functionality – correctness, reliability etc.
- performance – response time, throughput, speed etc.
- economy – cost effectiveness.

But in this era security of software systems has reached such importance that it also needs to be included in the list above. Software metrics as the term is most commonly used today, concerns itself almost exclusively with functionality and economy [3]. But there is now an imperative necessity to broaden the scope of software metrics to include the security characteristic as well.

1.5 Classification of Software Metrics

Software metrics may be broadly classified as either product metrics or process metrics. Product Metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure the complexity of the software design, the size of the final program, or the number of pages of documentation produced. In the context of security an example of a product metric may be the number of vulnerabilities identified in the software. Process metrics, on the other hand, are measures of the software development process such as overall development time, type of methodology used or the average level of experience of the programming staff.

Software metrics can also be classified into objective and subjective metrics. Objective metrics should always result in identical values for a given metric, as measured by two or more qualified observers. For subjective metrics even qualified observers may measure different values for a given metric, since their subjective judgment is involved in arriving at the measured value. An example of an objective security metric can be the number of invalidated input data and an example of a subjective metric can be the classification of a program as "highly secure", "secure" and "insecure". Although most programs might be easy to classify, those on the borderline might reasonably be classified in different ways by different knowledgeable observers.

Metrics can also be categorized as primitive or computed. Primitive metrics are those that can be directly observed, such as the number of security bugs observed in unit testing. Computed metrics are those that cannot be directly observed but are computed in some manner from other metrics. Examples of Computed Security metrics include the average number of vulnerabilities noticed per thousand lines of code.

2. SOURCE CODE ANALYSIS

Most of the available software metrics are centered on the source code, as this is the direct embodiment of the software system. But the major drawback of this approach is that the source code is typically available only late in the software development life cycle. Still, the utility of metrics derived from source code has been proven. It is highly likely that the source code will prove useful when measuring the security aspect as well. Many static analyzers of source code are now available and these typically analyze the source code without executing it. The output from the analyzers can be used for various purposes including identifying possible coding errors and in formal methods that mathematically prove properties about a given program. (that the program behavior matches its specification). In the context of security metrics these source code analyzers can provide valuable data that provide an insight into the various security aspects of the system. Many available static analyzers have to be enhanced to be used in this manner. Source Code Analyzers can be developed that identify potentially vulnerable code. [5] details the desirable properties of such "security aware" source code analyzers.

3. EXISTING METRICS AND THEIR SUITABILITY FOR MEASURING SECURITY

A brief overview of some of the available software metrics and their applicability for measuring "security" is presented below:

3.1 Size Metrics

A number of metrics attempt to quantify software "size". The metric that is most widely used , "Lines Of Code" or "LOC", suffers from the obvious deficiency that its value cannot be measured until after the coding process has been completed.

3.1.1 LOC

"Lines of Code" is possibly the most widely used metric for program size. It would seem to be easily and precisely definable; however, there are a number of different definitions for the number of lines of code in a particular program. These differences involve treatment of blank lines and comment lines, non-executable statements, multiple statements per line, and multiples lines per statement, as well as question of how to count reused lines of code. The most common definition of LOC seems to count any line that is not a blank or comment line, regardless of the number of statements per line [6,7]. LOC has been theorized to be useful as a predicator of program complexity, total development effort, and programmer performance.

In the context of Security, the utility of the LOC metric is at best questionable because there seems to be no relationship between the LOC of a program and its security. Whether a program with more LOC is more or less secure than a program with a fewer LOC is yet to be proved. LOC is also influenced by some other factors like the programming language used as some recent programming languages have the ability to deliver more functionality with fewer LOC.

3.1.2 Function Points

Albrecht has proposed a measure of software size that can be determined early in the development process. The approach is to compute the total function points (FP) value for the project, based upon the number of external user inputs, inquiries, outputs, and master files. The value of FP is the total of these individual values, with the following weights applied: inputs: 4, outputs: 5, inquiries: 4, and master files: 10. Function points are intended to be a measure of program size, and, thus, effort required for development.

3.1.3 Bang Metrics

DeMarco defines system Bang as a function metric, indicative of the size of the system. In effect, it measures the total functionality of the software system delivered to the user. Bang can be calculated from certain algorithm and data primitives available from a set of formal specifications for the software. The model provides different formulas and criteria for distinguishing between complex algorithmic versus heavily data oriented systems.

With regard to security, all the size metrics are of little, if any, utility, as the relationship between software size and software security is not yet established. Any attempt to reuse these metrics, for security, must first determine the relationship between size and security, if any exists.

3.2 Complexity Metrics

Numerous metrics have been proposed for measuring program complexity – probably more than for any other program characteristic. As is the case with size metrics, measures of complexity that can be computed early in the software development life cycle will be of greater value in managing the software process.

3.2.1 Cyclomatic Complexity – v(G)

Given any Computer Program, we can draw its control flow graph G, wherein each node corresponds to a block of sequential code and each arc corresponds to a branch or decision point in the program. The cyclomatic complexity of such a graph can be computed by a simple formula from graph theory, as v(G) = e – n + 2, where e is the number of edges, and n is the number of nodes in the graph.

3.2.2 Extensions to v(G)

Myers noted that McCabe's cyclomatic complexity measure v(G), provides a measure of program complexity but fails to differentiate the complexity of some rather simple cases involving single conditions (as opposed to multiple conditions) in conditional statements. As an improvement to the original formula, Myers suggests extending v(G) to v'(G)=[l:u], where l and u are lower and upper bounds, respectively, for the complexity. This formula gives more satisfactory results for the cases noted by Myers [8].

Stetter proposed that the program flow graph be expanded to include data declarations and data references, thus allowing the graph to depict the program complexity more completely. If H is the new program flow graph, it will generally contain multiple entry and exit nodes. A function f(H) can be computed as a measure of the flow complexity of program H. The deficiencies noted by Myers are also eliminated by f(H) [9]

3.2.3 Knots

The concept of program knots is related to drawing the program control flow graph with a node for every statement or block of sequential statements. A knot is then defined as a necessary crossing of directional lines in the graph. The number of knots in a program has been proposed as a measure of program complexity [10]

3.2.4 Information flow

The information flow within a program structure may also be used as a metric for program complexity. Henry and Kafura [11] have proposed such a measure. Basically, their method counts the number of information flows entering (fan-in) and exiting (fan-out) each procedure. The procedure's complexity is then defined as :

$$C = [\text{ procedure} - \text{length }] . [\text{ fan-in} . \text{fan-out }]^2$$

All the complexity metrics have generally been related to programming effort, debugging performance, and maintenance effort. In the context of security, these metrics may also serve as indicators of the strength of security mechanisms needed by the program. But for this, evidence needs to be established that a relationship exists between the complexity of the program and the strength of the security mechanisms needed by it. For example, a more complex program may require more effort for security.

3.3 Halstead's Product Metrics

Most of the product metrics proposed have applied to only one particular aspect of the software product. In Contrast, Halstead's software science proposed a unified set of metrics that apply to several aspects of programs, as well as to the overall software production effort. Thus, it is the first set of software metrics unified by a common theoretical basis.

3.3.1 Program Vocabulary

Halstead theorized that computer programs can be visualized as a sequence of tokens, each token being classified as either an operator or operand. He then defined the vocabulary, N, of the program as:

$n = n1 + n2$

Where n1 = the number of unique operators in the program and

n2 = the number of unique operands in the program.

Thus, n is the total number of unique tokens from which the program has been constructed. [12].

3.3.2 Program Length

Having identified the basic tokens used to construct the program, Halstead then defined the program length, N, as the count of the total number of operators and operands in the program. Specifically,

$N = N1 + N2$

Where N1 = the total number of operators in the program, and,

N2 = the total number of operands in the program.

Thus, N is clearly a measure of the program size, and one that is directly derivable from the program itself. In practice, however, the distinction between operators and operands may be non-trivial, thus complicating the counting process. [12]

Halstead theorized that an estimated value for N', designated , can be calculated from the values of n1 and n2 by using the following formula:

$$N' = n1 \log_2 n1 + n2 \log_2 n2.$$

Thus, N is a primitive metric, directly observable from the finished program, while N' is a computed metric, which can be calculated from the actual or estimated values of n1 and n2 before the final code is actually produced.

Some studies have attempted to relate N and N' to other software properties such as complexity and defect rates. Similar studies need to be done to explore any possibility of relationship between Halstead's metrics and software security.

3.4 Quality Metrics

One can generate long lists of quality characteristics for software – correctness, efficiency, portability, maintainability, reliability and perhaps even security. Early examples of work on quality metrics are discussed by Boehm. [13,14]. Unfortunately, the characteristics often overlap and conflict with one another; for example, increased portability may result in lowered efficiency.

Although a good deal of work has been done in this area, it exhibits less commonality of direction or definition that other areas of metric research, such as software size or complexity.

3.4.1 Defect Metrics

The number of defects in the software product should be readily derivable from the product itself; thus, it qualifies as a product metric. However, since there is no effective procedure for counting the defects in the program, the following alternative measures have been proposed:

- Number of design changes
- Number of errors detected by code inspections
- Number of errors detected in program inspections
- Number of code changes required

The number of defects observed in a software product provides, in itself, a metric of software quality. These metrics can be extended to include "security defects" as well. We may take into account for example the number of security design changes required, number of security related errors detected by code inspections and the number of code changes necessitated by security related aspects. These are likely to provide a good measure of the security mechanisms built into the program. But they have a major disadvantage: these metrics are available only lately in the software development life cycle.

3.4.2 Reliability Metrics

It would be useful to know the probability of software failure, or the rate at which software errors will occur. Again, although this information is inherent in the software product, it can only be estimated from the data collected on software defects as a function of time. If certain assumptions are made, these data can then be used to model and compute software reliability metrics. These metrics attempt to measure and predict the probability of failure during a particular time interval, or the mean time to failure (MTTF).

The parallels between software reliability metrics and software security metrics have been discussed by [15]. The paper also highlights some challenges that have to be overcome before we attempt to devise metrics for security based on the reliability metrics. The paper also discusses the usage of probability-based framework to model security breaches analogous to the modeling of faults in the context of reliability. Time cannot be a good criterion when it comes to security and hence the paper suggests the usage of "effort" in the place of "time". The paper also suggests having the Mean Time To Breach metric analogous to MTTF.

4. DISCUSSION

Of all the metrics, discussed above only the quality metrics seem to be the most likely candidates for consideration in the context of security metrics. But this too, is not without drawbacks. This indicates the need for the development of new metrics focused exclusively on security. Current measurements about security are highly subjective and need a high involvement of the human element. This needs to be reduced. Clearly, much more is to be done in the area of security metrics, as the currently available metrics do not address the security issue effectively. [16] can be a good starting point for the commencement of research in this area. The paper suggests the usage of Historical data Collection and data mining techniques and AI Assessment techniques in the development of security metrics. With the increasing demand for secure systems, the development of security metrics would be well worth the effort. Such metrics can be of immense help to the software engineers in the engineering of secure software systems. It needs to be pointed out here that even in the context of security, different systems may require different levels of security. Another point worth mentioning here is that, currently security seems to be taken into account and measured only for systems where "very high" security is demanded such as in military systems. Problems faced in this context are similar to that faced in the development of "ultra-high" reliable systems, the evaluation of which has been notably unsuccessful. This suggests that, in the development of security metrics, we should initially focus on systems where the security requirements are also modest. Once these attempts prove successful, work can be done for ultra-high secure systems as well.

5. CONCLUSION

The paper discussed the applicability of existing software metrics for measuring "software security". It is observed that the utility of most of the available metrics is at best questionable. And much effort is needed to justify the usage of these metrics for security. Metrics focused exclusively on security need to be developed and this requires a clear understanding of the applicability, utility and shortcomings of the existing metrics.

6. REFERENCES

[1] http://en.wikipedia.org/wiki/Software_metrics

[2] Rubin, H. A. "*Macro-Estimation of Software Development Parameters: The ESTIMACS System*." Proc. SOFTFAIR: A Conference on Software Development Tools, Techniques, and Alternatives. New York: IEEE, July 1983, 109-118.

[3] Mills, Everald E, "*Software Metrics SEI Curriculum module SEI – CM – 12 – 1.1*", Carnegie Mellon University, Software Engineering Institute, December, 1988.

[4] SSE-CMM: Systems Security Engineering Capability Maturity Model, International Systems Security Engineering Association (ISSEA), referenced on July 7, 2008, http://www.sse-cmm.org/metric/metric.asp

[5] Chess, Brian, "*Metrics That Matter – Quantifying Software Security Risk*", Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics, National Institute of Standards And Technology, February 2006.

[6] Boehm, B. W. "*Software Engineering Economics*", Englewood Cliffs, N. J.: Prentice-Hall, 1981.

[7] Jones, T. C. "*Programming Productivity*", New York: McGraw-Hill, 1986.

[8] Myers, G. J. "*An Extension to Cyclomatic Measure Of Program Complexity*", ACM SIGPLAN Notices 12, 10 (Oct. 1977), 61-64.

[9] Stetter, F. "*A Measure of Program Complexity*", Computer Languages 9, 3-4(1984), 203-208.

[10] Woodward, M. R., M. A. Hennell, and D. Hedley. "*A Measure of Control Flow Complexiy in Program Text*", IEEE Trans. Software Eng. SE-5, 1 (Jan. 1979), 45-50.

[11] Kafura, D. and S. Henry. "*Software Quality Metrics Based on Interconnectivity*", J. Syst. and Software 2, 2 (June 1981), 121-131

[12] Halstead, M. H. "*Elements of Software Science*", New York: Elsevier North-Holland, 1977.

[13] Boehm, B. W., J. R. Brown, and M. Lipow, "*Quantitative Evaluation of Software Quality*", Proc. 2nd Intl. Conf. On Software Engineering, Long Beach, Calif.: IEEE Computer Society, Oct. 1976, 592-605.

[14] McCall, J. A., P. K. Richards, and G. F. Walters, "*Factors in Software Quality, Vol. I, II, III: Final Tech. Report*.", RADC-TR-77-369, Rome Air Development Center, Air Force Systems Command, Griffiss Air Force Base, N. Y., 1977.

[15] Littlewood Bev, Brocklehurst Sarah, Fenton Norman, Mellor Peter, Wright David, Dobson John, McDermid John, Gollmann Dieter, "*Towards Operational Measures of Computer Security*", http://www.csr.city.ac.uk/people/bev.littlewood/bl_public_papers/Measurement_of_security/Quantitative_security.pdf

[16] Jansen, Wayne, "*Directions in Security Metrics Research*", Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, April 2009.

[17] Ferrari, D. "*Considerations on the Insularity of Performance Evaluation*", IEEE Trans. Software Eng. SE-12, 6 (June 1986), 678-683.