Software semantic metrics: A Survey

Rasha Gaffer M. Helali

Abstract— **Software metrics are quantities estimates for software product attributes which guide us in taking managerial and technical decisions [1]. The current used software metrics depend on syntactical source code at the same time they ignore its semantic aspects. This fact motivates us to focus on semantic metrics instead of traditional used metrics. Moreover, semantic metrics are more precise than syntactic ones, they able to capture the semantic defects of the software system development. In this paper, the researcher will make survey of an existing research on software metrics along with potential research challenges and opportunities.**

Index Terms—**Metrics, Semantic, Synthetic, quality, data mining.**

I. Introduction

The quality of software systems become an important issue in continuation of business, i.e. The vast amount of software used in markets and their role in managing precise and dangerous tasks [1]. The assertion by Ward Cunningham in 1992, that quick and careless development with poor quality leads to many years of expensive maintenance and enhancements. So, software measurement paradigm becomes more and more important to monitor quality during all software development stages.

The field of software metrics is a relatively young one [2], whose origins can be found in the work by Halstead published in 1972. From then on, the interest in software metrics has increased because they have been recognized as a useful instrument for managing software process effectively.

Software metrics allow to use a real engineering approach to software development, providing the quantitative and objective base that software engineering was lacking. In fact, their use in industry is becoming more and more widespread.[2] As regards the research in software metrics [2], it has undergone a great evolution, in the first period the focus was very much on inventing new metrics to measure different software attributes, without concerning the scientific validity of these metrics. Recently, much work has been done on how to apply measurement theory to software metrics and ensure their validity. [2]

The remainder of this paper is organized as follows. In the next section characteristics and challenges in software measurement are discussed. Current used Metrics including syntactic and semantic metrics are described in section 3. Section 4 examines the current Challenges and finally conclusion is presented.

II. Characteristics and Challenges in Software Measurement

A principal objective of software engineering is to improve the quality of software products. Many models spot distinction between tow types of software attributes internal and external ones [1]. Since internal attributes can be measured directly from source code, external attributes measured indirectly based on another internal attributes. Much research has been done on identifying quality models to support software quality improvements. Such quality models concentrated on mapping between internal and external attributes i.e. McCall, Boehm [4], FURPS(+) and ISO9126 (ISO 1991) models. The main objective of these models is to focus on formal products and identify key attributes of quality from the user prospective [5]. Boehm and MaCall model identify 3 key attributes called "quality factors" such as reliability, usability, maintainability (High level external attributes) but it related to many internal attributes called quality criteria [5] . Dromey and FURPS(+) [6] are focusing on the relationship between the quality attributes and the sub-attributes, as well as attempting to connect software product properties with software quality attributes. In 1992, derivation of McCall model was proposed as a basis for international standard software quality ISO9126 (ISO 1991). The model decomposes the quality into six factors as follow: functionality, efficiency, usability, maintainability and portability. Each of these factors is defined as a set of attributes. [5]

Software Metrics are used to measure specific quality attributes using relationship identified by standard quality models. Current used metrics are succeed in measuring internal attributes but still there is shortcomings in measuring external attributes accurately such as reliability, availability …etc. The following section focus on current used metrics and their limitations.

II. Software Metrics

In this section, we will discuss software metrics concept and classification along with their advantages and shortcomings each. According to the IEEE standard glossary of software engineering terms (adapted from [6]), metrics are a quantitative measure of the degree to which a system, component, or process possesses a given attribute."

To this effect, researchers have long been interested in defining and analyzing metrics that capture properties of software products and software processes, metrics are now the subject of many textbooks [7, 8 , 2].

The importance of software metrics comes from the need for assessing requirements, identifying error prone components at early stages of software development, improving software quality attributes such as reliability, maintainability, availability …etc. Different classes of metrics measure different aspects of system quality, so a single or multiple metrics belong to a class cannot be successfully used to measure all quality related facets of a system.

G. Eason.et.al. in[9] Identify the characteristics of an optimal metric to be simple, objective, easily obtainable, valid and robust. Generally software metrics are classified as follow:

1. Process metric which highlights the process of software development. It mainly aims at process duration, cost incurred and type of methodology used.
2. Project Metrics that are used to monitor project situation and status. And identify risk. E.g.. Staff number and their patterns, cost, etc…
3. Product metrics describe the attributes of the software product at any phase of its development. [10] Metrics are further classified into static and dynamic. [10]

Metrics are further classified into static and dynamic. Static measures are obviously simpler to collect because there is no need to run the software. In contrast, to obtain dynamic measure of code, simulation models of the software system are needed, which are available very late in the software development lifecycle [2]. Static metrics are widely used due to the fact that they are easier to obtain, especially at the early stages of software development. However, the potential benefits of dynamic metrics collected by executing the program outweigh the complexity and cost of measuring them.[2] Next section discuses research in syntactic metrics in details.

A. Syntactic metrics

In this section, we will discuss the traditional metrics and their limitations. Most widely used software metrics nowadays are based on syntactical aspects of software; as such, they reflect how a program is represented, but not what a program does [3]. The most used software metrics are LOC (lines of code) and Cyclamate Complexity [11]. These measures were originally defined for procedural programs and later incorporated for object-oriented systems. The LOC metric measure software size, while cyclamate complexity measures logical complexity of a module [3]. Software metrics proposed and used for procedural paradigm have been found inadequate for object-oriented software products [1]. The word syntactic reflects to what extend we can use source code syntax to estimate some quality attributes. In [12] quality is defined as the degree to which a product is bug-free.

In this point it is important to say that the use of pre-release defects as an indicator of quality is questioned. Knowing that there were a large number of defects during the coding stage does not mean there will be a lot of bugs in the post release version too. In contrast, other researchers defined quality by using different concepts such as reliability, availability, lower maintainability cost and sometimes the ability to perform tasks as it should.

Some current studies move through different direction and see the quality as a weighted combination of different software attributes. For example, source code length is related to complexity and a lower software complexity could lead to a greater software reliability (Fenton & Pfleeger, 1997)[1]. It is not wrong to say that there is a relationship between complexity and the length of the program. However, all studies agree that when measuring complexity one should take into account something more than length. This approach was followed by Törn et al. (1999) [13]. A Number of studies move through the same direction and investigate complexity measure using code syntax. In [7] researchers divide complexity of software into three separate classes' the first class is the essential complexity which is determined by the problems that the software tries to solve. The second class the selecting complexity is determined by the program languages, the problem modeling methods and the software design methods. Lastly, The incidental complexity is determined by the quality of the involved implementer.[7] The most common used complexity metrics are Halstead metrics which introduced in 1977 by Maurice Halstead. One of it is main advantegs that it doesn't require deep knowlegde of prgram logical strcuter so it is easy to calculate but in the other hand, it desont give accurate measure because it doesn't consider program flow control.[7]

Another metric is WHCM which try to overcome HCM limitations. WHCM adds wieght of the code instrcutions such as loops or branches. In [14] Thomas J. McCabe introduced a software complexity metric named McCabe Cyclamate Complexity Metric. It's main limitation it doesn't distinguish between code lengths and it ignores the complexity added by the nesting codes. A significant drawback of syntactic metrics is that different structural aspects of code can result in different metrics value, even when the code is performing the same task [15]. Syntactic metrics are not always accurate quality descriptors. Metrics that provide a better mapping between software and its associated quality factors thus has the potential to be used in improving software quality, including quality of newly developed software as well as currently maintained software. Such metrics can help in identification of good reusable software components. [3]

Another important point should be highlighted is that desirable quality attributes like reliability and maintainability can not be measured until some operational version of the code is available. Yet we wish to be able to predict which part of the software is less reliable, more difficult to test, or require more maintenance than others, even before system is completed [2]

B. Semantic metrics

Traditional syntactic metrics discussed above reflect how program are represented, but not what a program does [3]. Yet, many important program attributes may have more to do with the latter than the former. In the following we introduce a number of software metrics that reflect semantic properties of software products. Gall, C. S In 2008 [16] suggests an approach using semantic metrics to provide insight into software quality early in the design phase of software development by automatically analyzing natural language (NL) design specifications for object-oriented systems is presented. Semantic metrics are based on the meaning of software within the problem domain [17]. New trend has introduced by Selim et.al. They tried to extend the concept of using semantic information to improve software quality [18]. In [19], researchers extend semantic metrics to analyze design specifications. Since semantic metrics can now be calculated from early in design through software maintenance, they provide a consistent and seamless type of metric that can be collected through the entire lifecycle. Results indicate semantic metrics calculated from design specifications can

give insight into the quality of the source code based on that design.

Previously in 1992 [20] some researches make spot on software faults that infrequently affect output cause problems in most software and are dangerous in safety-critical systems. When a software fault causes frequent software failures, testing are likely to reveal the fault before the software is released; when the fault "hides" from testing, the hidden fault can cause disaster after the software is installed. A simple metric, derivable from semantic information found in software specifications, indicates software sub functions that tend to hide faults. Potential for hidden faults can be further explored using empirical methods .A study [21] in 2008 was empirically investigated the suite of object-oriented (OO) design metrics introduced in (Chidamber and Kemerer, 1994). More specifically, their goal is to assess these metrics as predictors of fault-prone classes. Norman in [22] found relationship between faults density and module size and analysis time thorough his study. He confirmed that the number of faults discovered in pre-release testing is an order of magnitude greater than the number discovered in 12 months of operational use. Marcus etal. In [23] try to improve this study by suggesting a way for predicting software faults in OO programs.

Some studies shed light on how to integrate entropy concept with semantic aspects of software as quality measure. Such concept dates back to 1997 when D. Melamed define semantic entropy as the measure of semantic -ambiguity and uninformative Hess [24]. It is a graded lexical feature which may play a role anywhere lexical semantics plays a role. Brown et al. (BD+91)[25] present a word-sense disambiguation algorithm involving minimization of semantic entropy weighted by word frequency. Yarowsky (Yar93)[26] compares the semantic entropy of homographs conditioned on different contexts. Salwa K and Abd-El-Hafiz, in 2004 [27] also address entropy as a means to measure software information content. They use the entropy metrics to study the evolution of the modules within the system. in 2004 [28] also address entropy as a means to measure software information content. Oleksandr et.al. [29] Proposes a novel interpretation of an entropy-based metric to assess the design of a software system in terms of interface quality and understandability. The research in this area is still too young.

c. Advantages of semantic over syntactic metrics

In this section paper will discus the benefits of having semantic metrics in comparison to static syntactic one.
Syntactic metrics are simpler to collect and easy to understand due to, the way to measure it directly from source code. In contrast, Semantic metrics is more difficult to measure because it relay on deep understanding to the function of software without considering its programming language. In the other hand, as mentioned earlier, one of the main drawbacks of syntactic metrics is the fact that most metrics address only one aspect of the multifaceted software development process. Thus these metrics might fail to take into consideration, for example, the trade of between two resources because its dependent variable is only one of those resources. Researchers have tried various methods to overcome this problem. One of those methods which are more used than any other is combining metrics from various classes to give a more general picture. Another mentioned point is that using of syntactical code defects only as indicator of quality is questioned. Semantic metrics try to overcome these limitations by considering new semantic aspects of source code. The major differences between semantic and syntactic metrics are listed in table 1.

Table 1: Comparison between syntactic and semantic metrics

Syntactic metrics	Semantic metrics
Simpler to collect	Difficult to obtain
Deal only with structural aspects of source code	Deal more with semantic and functional aspects
Dependant on programming language	programming language independent
Inefficient to deal with OO features.	Applicable to both procedural and object oriented software
Less precise in some quality attributes	More precise that syntactic metrics
Available after coding stage	Can be measured in different software development stages

D. Complementary Enhancing Techniques

Software measurement can be classified in two parts prediction and estimation. The above mentioned techniques fall in the area of estimation. Much research is also done in field of prediction. The simplest example of software quality prediction is to use available attribute to predict another different quality feature. Studies in the area classify the software into different parts such as evolution-critical, evolution-prone and evolution-sensitive parts. The first term refers to parts of software that need to be evolved due to lack of quality. The second term refers to parts that are likely to be evolved. The third term refers to parts which can lead to problems as it evolves. Recently, new analysis techniques are used to enhance quality prediction. The article [17], presents the use of data miner tool called Multi method for prediction, experiments on three datasets in the Metrics Data Project (MDP) of the NASA has been done. Results confirms their efficient as defect prediction tool. In the same direction Salwa K [30] also conducts the use of data mining in detection of function clones in software systems. T Menzies [31] touch a hot issue by investigating the use of data mining to generate defect predictor from static code attributes. Many researchers use static attributes to guide software quality predictions (see [32], [33]).

Using multiple metrics in the prediction or detection process may increase the accuracy and thus increase software quality. Much research is required here to integrate new analytical methods that grantee prediction accuracy.

IV. CURRENT CHALLENGES

From the above detailed discussions of different types of metrics, it can easily be said that the field of semantic metrics is wide open for researchers.

Some of the possible research directions are listed below:

- As mentioned previously in context of complexity metrics latest researches go though merging different technologies to improve quality. In addition to that, integrate new analytical technologies such as data mining for software evaluation.
- Semantic metrics is a new trend in software measurement. Most of studies in this area agreed upon evaluating software in early stages in its development life cycle are better for quality assurance. Since much studies starts suggesting metrics to work in design phase or even after implantation and a little address to extract knowledge from requirements.
- New suggested metrics lacks the mathematical support. So researches encouraged to move toward this direction.
- No much research in semantic metrics instead much focuses now a day is on semantic web and ontology. So many researches still required in this area.

V. Conclusion

The paper is an attempt to surveys the literature that has been published to date, on software metrics generally and semantic metrics particularly. The main objective of the paper is to explore a new research direction in the field. There is still much to be done in the area. We expect that this area of research will get more attention in future due to its importance in business market.

Acknowledgment

I wish to thank Prof. Ali Mili for his helpful advices.

References

[1] *Norman E. Fenton, 1991, Software Metrics, A Rigorous Approach, Chapman & Hall, London*

[2] *A Survey of Software Metrics, Fabrizio Riguzzi, July 1996, DEIS Technical Report no. DEIS-LIA-96-010.*

[3] *A.Mili .et.al. Semantic software metrics.2013. unpublished.*

[4] *Bohem, B.W, Brown,j.R. and Lipow, M, "Quantitve evaluation of software quality" proceedings of the second international conference on software engineering, 1976.*

[5] *Norman E.Fenton, shari Lawerance Pfleeger. Software metrics Arigorous and practical approach. Second edition.PWS Publishing company. 20 park plaza Boston.*

[6] *Dromy. R.G, "cornering the chimera" , IEEE software ,31(1),33-34,1996.*

[7] *Christof Ebert and Reiner Dumke. Software Measurement: Establish, Extract, Evaluate, Execute. Springer Verlag, 2007*

[8] *Norman E. Fenton and Shari Lawrence Pfleeger. Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company, 1997.*

[9] *G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lips Software Metrics SEI Curriculum Module, SEI-CM-12-1.1, December 1988.*

[10] *A Survey on Metric of Software Complexity Sheng Yu, Shijie Zhou.*

[11] *Survey on Impact of Software Metrics on Software Quality - (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No. 1, 2012*

[12] *McCabe T. A complexity metric. IEEE Transactions on Software Engineering, 1976, 2(4): 308-320.*

[13] *A Whitepaper on Metrics, Andreas Rau, Steinbeis Transferzentrum Softwaretechnik, 1998, 1999, 2001.Last Change: 2001-08-06*

[14] *Thomas J McCabe, "A Complexity Measure", IEEE Transactions On Software Engineering, IEEE, Washington, Oct 1976, pp. 308-320*

[15] *Torn, A., T. Andersson and K. Enholm, 1999. A complexity metrics model for software. South Afr. Comput. J., 24: 40-48.*

[16] *Gall, C. S. Inf. Technol. & Syst. Center, Univ. of Alabama in Huntsville, Huntsville, AL Lukins, Stacy K.; Etzkorn, Letha H.; Gholston, Sampson; Farrington, Phillip A.; Utley, Dawn R.; Fortune, J.; Virani, Shamsnaz Semantic Metrics, Conceptual Metrics, and Ontology Metrics: Volume: 2 , Issue: 1 Page(s): 17 – 26.*

[17] *Eric S. Raymond, the Art of Unix Programming, Addison-Wesley, New York, 2004.*

[18] *Semantic Metrics, Conceptual Metrics, and Ontology Metrics: Letha H. Etzkorn*

[19] *Bo Hu, Yannis Kalfoglou, Harith Alani, David Dupplaw, Paul Lewis, Nigel Shadbolt Managing Knowledge in a World of Networks Lecture Notes in Computer ScienceVolume 4248, 2006, pp 166-181. Semantic Metrics.*

[20] *Jeffrey M. Voa. Keith W. Miller Semantic metrics for software testability. Journal of Systems and Software. Volume 20, Issue 3, March 1993, Pages 207–216 Oregon Metric Workshop on Software Metrics, 1992.*

[21] *Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems Software Engineering, IEEE Transactions on Date of Publication: March-April 2008, Marcus, Andrian. Wayne State Univ., Detroit. Poshyvanyk, Denys; Ferenc, Rudolf Volume: 34, Issue: 2 . Page(s): 287 - 300*

[22] *Software Metrics and Risk (1999) by Norman Fenton, Martin Neil.*

[23] *A validation of object-oriented design metrics as quality indicators 06 August 2002, Issue Date : Oct 1996, Sponsored by : IEEE Computer Society*

[24] *D. Melamed, "Measuring Semantic Entropy," Proceedings of the SIGLEX Workshop on Tagging Text with Lexical Semantics, Washington, DC, 1997*

[25] *P. F. Brown, S. Della Pietra, V. Della Pietra, R. Mercer, "Word Sense Disarnbiguation using Statistical Methods", Proceedings of the ~9th Annual Meeting of the Association for Computational Linguistics, Berkeley, Ca., 1991.*

[26] *D. Yarowsky, "One Sense Per Collocation," DARPA Workshop on Human Language Technology, Princeton, N J, 1993*

[27] *Abd-El-Hafiz, Salwa K. An information theory approach to studying software evolution[J]. AEJ - Alexandria Engineering Journal, v 43, n 2, March, 2004, p 275-284.*

[28] *Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems Software Engineering, IEEE Transactions on Date of Publication: March-April 2008, Marcus, Andrian. Wayne State Univ., Detroit. Poshyvanyk, Denys; Ferenc, Rudolf Volume: 34, Issue: 2 . Page(s): 287 - 300*

[29] *Panchenko, O. ; Hasso Plattner Inst. for Software Syst. Eng., Potsdam, Germany ; Mueller, S.H. ; Zeier, A. Measuring the quality of interfaces using source code entropy, Industrial Engineering and Engineering Management, 2009. IE&EM '09. 16th International Conference, Oct. 2009, Page(s): 1108 - 1111*

[30] *Salwa K, A Metrics-Based Data Mining Approach for Software Clone Detection. Pages 35-41 , COMPSAC '12 Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference IEEE Computer Society Washington, DC, USA ©2012.*

[31] *N. Nagappan and T. Ball, "Static Analysis Tools as Early Indicators of Pre-Release Defect Density," Proc. Int'l Conf. Software Eng., pp. 580-586, 2005.*

[32] *T. Khoshgoftaar, "An Application of Zero-Inflated Poisson Regression for Software Fault Prediction," Proc. 12th Int'l Symp. Software Reliability Eng., pp. 66-73, Nov. 2001.*

[33] *W. Tang and T.M. Khoshgoftaar, "Noise Identification with the KMeans Algorithm," Proc. Int'l Conf. Tools with Artificial Intelligence (ICTAI), pp. 373-378, 2004.*