IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 6, August 2014.
www.ijiset.com
ISSN 2348 – 7968

A Note on Complexity and Understandability as Attributes for Assessing the Reusability of Software Components.

Sammy Nyasente [1], Prof. Waweru Mwangi [2] and Dr. Stephen Kimani[3]

[1] School of Computing and IT, Jomo Kenyatta University of Agriculture and Technology, P O Box 62000-00200 Nairobi Kenya

[2] School of Computing and IT, Jomo Kenyatta University of Agriculture and Technology, P O Box 62000-00200 Nairobi Kenya

[3] School of Computing and IT, Jomo Kenyatta University of Agriculture and Technology, P O Box 62000-00200 Nairobi Kenya

Abstract

Using software metrics is one of the ways of evaluating and improving reusability of software components. Among other reusability aspects of a component, the Complexity/understandability aspect is crucial and needs to be assessed and managed. This reusability aspect can be assessed by measuring one of two overlapping reusability attributes: Complexity and understandability. We closely examine these two reusability attributes in this article, and issues that must be considered when making a choice between them are presented thereto.

Keywords: *Reusability, Reusability attributes, Complexity, Understandability.*

1. Introduction

Software reuse, the use of existing software artifacts to construct new software [1], is often pursued in the quest of bridging the growing gap between the demand and supply of new software applications that need to be developed [2]. However, the concept of reuse is not straightforward, because mere availability of components does not guarantee their reuse—as components must possess the required degree of reusability for them to be reused in other contexts other than the one they were initially developed for [3]. This means that, the potential of any component's reuse is dependent on its degree of reusability; therefore, efforts should converge at developing components with a high degree of reusability when developing for reuse [4]. One of the ways of achieving this objective is using metrics to ascertain if the software being developed possesses the required degree of reusability, and if not, strategies for improvement should be devised. A metric is a quantitative indicator of an attribute of a thing [5].

The Reusability of software components is influenced by certain measurable attributes, thus it can be determined by relating each of these attributes with appropriate metrics and finding out how these metrics collectively determine the reusability of a component [6]. One of the major challenges in coming up with an effective reusability measurement framework is to determine the attributes to be included in the framework. This is because; (i) an effective reusability assessment framework should have as few attributes as possible, but at the same time be sufficient in assessing all aspects of reusability [4], and (ii) software quality attributes are not independent, but overlap—with some impacting on one another in a positive sense, while others doing so negatively—requiring a tradeoff between attributes [2].

Among other reusability aspects of a component, the Complexity/understandability aspect is crucial and needs to be assessed and managed [4,7-9]. This aspect of reusability can be assessed by considering two factors viz., Complexity, and understandability. These factors are not independent, but overlap; therefore, a choice has to be made between them. This article focuses on these two attributes. Issues that must be considered when making a choice between them are highlighted thereto.

2. Complexity vs. Understandability

Complexity is defined in IEEE Standard 610.12 [10], as the degree to which a system or component has a design or implementation that is difficult to understand and verify. Laird and Brennan [11] distinguish three different aspects of complexity viz., structural, conceptual, and computational. Structural complexity is concerned with the design and structure of the software, and it can be measured using existing structural complexity metrics such as Lines of Code (LOC), Function Points (FP), Cyclomatic Complexity metrics, etc. Conceptual complexity refers to difficulty in understanding a software artifact. Conceptual

ISSN 2348 – 7968

complexity tends to be more psychological, based on the mental capacity, and thought processes of programmers. It could be based on how difficult it is to specify a solution, or in some cases the amount of effort required to understand the code itself. This type of complexity is difficult to quantify, and there are no specific metrics known to measure it. Lastly, Computational complexity refers to the complexity of the computation being performed, and it can be measured by determining the amount of time and space required for the calculations.

Understandability on the other hand, is concerned with the estimated effort needed by a user to recognize the concept behind a software component and its applicability [7]. A system that has a design that is more understandable (easy to understand) is said to have high understandability [12]. That is to say, a module that requires less effort to understand has a high degree of understandability. According to [12], Understandability can be viewed from two perspectives viz., internal and external. Internal understandability is an internal product quality, which helps in achieving many other qualities, such as evolvability and verifiability, whereas external understandability is a factor of a product's usability concerned with the predictability of a system to the user.

3. Complexity and Understandability in Reusability Measurement

Reusability has the best return on investment of any software technology, if it is pursued in a proper manner [13]. However, reusability comes with a cost: it adds complexity to a component, hence reducing its understandability [14]. This is because, components that are developed for reuse need to possess the Generality property, for them to be reused in other contexts than the one it was initially developed for [3]. According to [15], Generality can be described as a state or quality of being not limited to one particular case, and assets must possess this characteristic for them to be reusable. According to [14], Generality increases the reusability of a component, as well as its complexity: as we add generality to a component, its reusability increases, and so does its complexity. This phenomenon calls for the management of complexity, in order to achieve high understandability, which is extremely important because; the decision of reusing a component in a given context is dependent on how well we understand what the component does—in order to decide whether it meets new requirements or not [3,7].

Understanding becomes even more important when a component does not perfectly fit the required needs within the reuse context—(which is more often the case), and the component has to be modified in order to fit the new reuse context [3]. This means that, complexity may impede reuse of a component. In addition, unnecessary complexity is associated with problems such as additional defects and lower productivity [11]. Laird and Brennan [11] could not describe it any better when they state that; "Unnecessary complexity is the Typhoid Mary of software development" (p. 54). Since software is inherently complex [11,12], and generality adds complexity to software, we need to manage complexity when developing *for* reuse, so that we may end up with components that are more understandable.

According to [12], certain guidelines can be followed to produce more understandable systems, regardless of the tasks they perform. That is, given two systems that perform similar tasks (tasks of inherently similar difficulty), the system that has been developed according to the said guidelines will be more understandable (less complex), than the one that is developed without following these guidelines. For example, concepts like abstraction and modularity enhance a system's understandability. This is to say that, software complexity can be diminished by enhancing understandability (i.e. complexity decreases when understandability increases and vice versa). Using laws of mathematical proportionality, Complexity and Understandability can be said to be inversely proportional. This can be restated mathematically as follows:
Let C_c be the complexity of a software component, and U_c be its understandability, then:

$$C_c \propto \frac{1}{U_c} \quad (1)$$

Equation 1 is the same as:

$$C_c = \frac{k}{U_c} \quad (2)$$

Where:
$C_c \equiv$ component's complexity
$U_c \equiv$ component's understandability
$k \equiv$ coefficient of proportionality (i.e. a quantity that relates the two variables: C_c and U_c).

Since Complexity and understandability can be expressed as an inverse proportion as in Eq (2), it then follows that, we only need to measure the value of only one variable (either C_c or U_c); because the value of one variable will tend to decrease at the same rate that the other variable's value increases. When the value of one variable has been determined, the value of the other variable can then be determined mathematically—if the coefficient of proportionality (value of k) is known. The values for C_c

ISSN 2348 – 7968

and U_c can be found by using appropriate metrics to measure the software, and then k can be found by multiplying the initial values of C_c and U_c. Therefore, a reusability assessment framework should only include either of the two attributes.

Two possible approaches for managing complexity/understandability aspect of a component—through measurement are evident. The first approach is by measuring the complexity of the component—using appropriate metrics then use the metrics information as the basis for devising ways of diminishing complexity, if it is too high. For instance, measures of a component's complexity can be useful in identifying complex designs and code; that ought to be simplified [11]. In this case, high values for complexity metrics would indicate flaws in the design and implementation of the component. Some of the criteria for measuring Complexity have been presented in [8,16,17].

The second approach is by measuring the understandability of the component; then devise ways of improvint it, if it is too low. This can be achieved by defining understandability guidelines, then use metrics to determine how well these guidelines have been adhered to. The Metrics information should form the basis for devising mechanisms for improving understandability—diminishing complexity as a result. Understandability can be determined by different criteria such as the ones presented in [4,7,9].

Although the two approaches are supposedly equal in terms of functionality; considering complexity as a reusability attribute (i.e. adopting the first approach)—like in the case of [8] is not straightforward as it seems. This is based on the fact that; high complexity of components is not always unjustifiable—i.e. components may be justifiably complex [2]. According to van Vliet [2] there may be good reasons why components have high complexity, and further decomposition of those components—to achieve understandability may be unjustifiable. For example, Complexity brought about by other inevitable reusability attributes such as generality is justifiable. In fact, a Component's complexity can be used as an indirect measure of a component's ability to perform many functions (Generality), assuming that the component was developed in a non-redundant way [18]. Therefore, putting an upper bound on the allowable value for a component's complexity is too simple an approach [2]. One possible consequence for having an upper bound for a component's complexity is that; we may unnecessarily diminish the degree of other desirable reusability attributes (such as generality), which may be the cause for the additional complexity of the component. The decision of whether to further simplify a component—to achieve understandability is not a trivial task, and may require expert opinion [2]. This means that, the task of determining whether complexity of a component is justifiable or not is difficult. One logical explanation to this phenomenon is that; complexity of a component is caused by multiple factors, which must be well understood and holistically considered, in order to make the determination.

In our view, considering understandability as a reusability attribute like in the case of [4,7], is a more straightforward and effective way of producing more understandable components. As aforementioned, it is possible to produce bigger and complex systems that are more understandable, by following certain guidelines. This means that, if these guidelines are followed to the latter, then we will end up with components that are justifiably complex. This approach eliminates the possibility of wanting to further simplify a component that is justifiably complex. Some of the guidelines for producing understandable components include: modularization, abstraction [12]; high cohesiveness and low coupling [12,16]; sufficient documentation [8,19], etc. We can relate these guidelines with certain factors that can be quantified using appropriate metrics, and then the metrics information can be used to examine how well the design guidelines were adhered to.

4. Conclusion and future Direction

This article puts into perspective two measurable—but overlapping reusability attributes, i.e. Complexity and Understandability. These attributes are helpful in assessing and managing the complexity/understandability aspect of reusable components. The definition and different aspects of understandability and complexity have been given. The relationship between these two attributes has been restated using laws of mathematical proportionality; for better understanding. The analysis of the relationship between these two attributes (Complexity and Understandability) shows that either of them is sufficient in assessing the complexity/understandability aspect of reusable components, thus only one of them should be included in a reusability assessment framework. Although using any of the two attributes may accomplish the same objective, we highlight some issues that arise when using measures of complexity in managing a component's complexity. We hold the view that, considering understandability as a reusability attribute addresses most of the issues that arise when Complexity is considered.

ISSN 2348 – 7968

References

[1] W.B. Frakes and K. Kang, Software reuse research: status and future, IEEE Transactions on Software Engineering, vol.31,July 2005, pp.529-536.
[2] H. van Vliet, Software Engineering: Principles and Practice (2nd ed.), John Wiley & Sons Ltd, 2000.
[3] J. Sametinger, Software Engineering with Reusable Components, Berlin: Springer-Verlag, (1997).
[4] S. O. Nyasente, W. Mwangi, and S. Kimani, A Metrics-based Framework for Measuring the Reusability of Object-Oriented Software Components, Journal of Information Engineering and Applications , Vol. 4, April 2014, pp. 71-84.
[5] W.B. Frakes and C. Terry, Software reuse: metrics and models, ACM Computing Surveys, vol. 28, June 1996, pp.415-435.
[6] P. S. Sandhu, H. Kaur, and A. Singh, Modeling of Reusability of Object Oriented Software System, World Academy of Science, Engineering and Technology, 2009, pp. 162-165.
[7] H. Washizaki, H. Yamamoto, and Y. Fukazawa, A Metrics Suite for Measuring Reusability of Software Components, Proceedings of the 9th International Symposium on Software Metrics (METRICS '03), IEEE Computer Society, Washington, DC, USA, 2003, pp.211-223.
[8] D. Hristov, O. Hummel, M. Huq, and W. Janjic, Structuring Software Reusability Metrics, Proceedings of the 7th International Conference on Software Engineering Advances. IARIA, 2012, pp. 421-429
[9] M. Ilyas, and M. Abbas, Role of Formalism in Software Reusability's Effectiveness, International Journal of Database Theory and Application , Vol. 6 No. 4, August 2013, pp. 119-130. (2013).
[10] IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.
[11] L. M. Laird, and M. C. Brennan, Software Measurement and Estimation A Practical Approach, Hoboken, New Jersey: John Wiley & Sons, Inc, 2006.
[12] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of Software Engineering (2nd ed.), New Jersey: Prentice-Hall, 2003.
[13] S. H. Kan, Metrics and Models In Software Quality Engineering, PEARSON, 2003.
[14] I. Sommerville, Software Engineering (9th ed.), Boston: Pearson Education, 2011.
[15] P. Návrat, and R. Filkorn, A Note on the Role of Abstraction and Generality in Software Development, Journal of Computer Science , Vol. 1 No. 1, 2005, pp. 98-102.
[16] E.S. Cho, M.S. Kim, and S.D. Kim, Component Metrics to Measure Component Quality, Proceedings of the Eighth Asia-Pacific on Software Engineering Con-ference (APSEC '01), IEEE Computer Society, Washington, DC, USA, 2001,pp.419-426.
[17] S. Chawla, and R. Nath, Evaluating Inheritance and Coupling Metrics, International Journal of Engineering Trends and Technology (IJETT) , Vol. 4 No. 7, 2013, pp. 2903-2908.
[18] G. Caldiera and V.R. Basili, Identifying and qualifying reusable software components, IEEE Computer, vol.24, Feb. 1991.
[19] A. B. AL-Badareen, M. H. Selamat, M. A. Jabar,J. Din, and S. Turaev, Reusable Software Components Framework, in proc. Advances Communications, Computers, Systems, Circuits and Devices, Puerto De La Cruz, Tenerife, 2010, pp. 126-130