International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 3 Issue 4 July 2013
Manuscript ID : 2249054XV3I4072013-05

AN EMPIRICAL REVIEW ON FACTORS AFFECTING REUSABILITY OF PROGRAMS IN SOFTWARE ENGINEERING

Neha Sadana, Surender Dhaiya, Manjot Singh Ahuja
Computer Science and Engineering Department
Shivalik Institute of Engineering and Technology, Aliyaspur (Ambala), Haryana, India

Abstract

Software engineering deals with the development of software systems and to reduce the cost and improve the development process. We make reuse of software in order to reduce effort, time and cost and thus it increases the productivity and quality of software programs. To check whether software components can be reused or not we measure reusability of the components. Reusability depends on number of factors. Using some of these factors reusability metric is formed to measure software component reusability. But software metric research area lacks standardization and this discourages metric usage and real applicability of these metrics in development and maintenance phase. This paper presents a survey on factors affecting reusability and reusability metrics to guide our researchers and developers for quality software development.

Keywords: Metrics, Complexity, Object-oriented, Software quality; Software Reuse and Software Metrics.

1 Introduction

As the size of the software system increased, new approaches of software development came into picture. These approaches include the object-oriented encoding, component-based development, aspect-based programming. But there is a need to reduce the effort, time and cost to build software so that productivity and quality of software programs can increase. Software reuse has been a cherished goal for software engineers and is viewed as a means to reduce development costs and to improve quality. Software reuse is use of existing software to build new software. Reusable software can be codes, templates, functions, procedures, objects, routines or framework including various documents. Initial investment is required to start the reuse process, but when reuse process is matured in the new software system it decreases the implementation time. Today, most of the applications are developed by using some existing libraries, codes, open sources etc. Software components are ready to use programming codes or controls that excel the code development. A software system is the collection of different software modules or the components that are integrated as the whole system. With the inclusion of software components the complete life cycle of the software is changed.

Reusability is the extent to which a segment of source code can be used again to add new functions with new slight or no modification. To measure reusability, we need software metric. Software metrics is concerned with measurements of reusability of software components. Software metric can be broadly classified into three categories-

Product Metric- It is used in documentation, design, performance, cyclomatic complexity for testability, coupling factor for maintainability.

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 3 Issue 4 July 2013
Manuscript ID : 2249054XV3I4072013-05

Process Metric- Its emphases are on software development process such as development time, methodology used and quality assurance techniques.

Resource Metric- Its emphasis are on human, hardware, software resource such as developers skill level, hardware reliability, software component quality.

1.1 Components Based Development

As the world of software development has evolved rapidly in the last decade, Component-Based Development has evolved from previous design and programming paradigms. This approach advocates the acquisition, adaptation, and integration of reusable software components, including commercial-off-the-shelf (COTS) products, to rapidly develop and deploy complex software systems with minimum engineering effort and reduced cost [1]. According to the Software Engineering Institute (SEI), the use of commercial off-the-shelf (COTS) products as elements of larger systems is becoming increasingly commonplace, due to shrinking budgets, accelerating rates of COTS enhancement, and expanding system requirements. COTS components also provide greater reliability as compared to custom-made components since they are refined by substantial field-testing. Voas [2] presented a summary of the advantages that can be gained by developing a system using COTS components:

Functionality is instantly accessible to the developer.
Components may be less costly than those developed in-house.

The component vendor may be an expert in the particular area of the component Functionality.

As COTS are used for reuse of components in software, now, we need to determine the degree of reusability of the component. For this many reusability metrics are proposed till date, out of these metrics some are used for object oriented and some are used for procedure oriented. In this work we are trying to overcome these problems by proposing metric for both procedure and object oriented approaches.

2. Factors affecting Reusability of components

After studying number of papers and documents we found that following are the factors which affect the reusability of components:

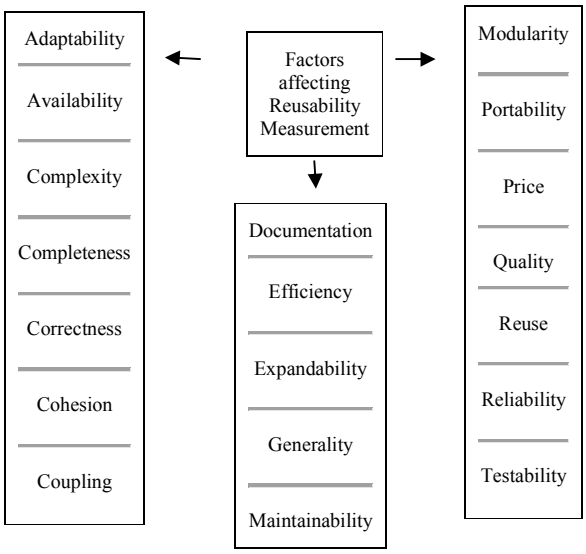

Fig1: Factors affecting Reusability Measurement

Availability – The availability of a software component can be determined that how easy and fast it is to retrieve. For measuring the availability of a component a generic, qualitative and subjective, metric can be used. The value obtained by the metric is placed on an ordinal scale and normalized to fit in the overall calculation of reusability of that component.

Documentation - A good documentation can make the software component more

reliable since it makes it easier to understand. Furthermore, it should contain the legal terms and conditions and thus make clear if it is licensed for reuse in the context of the developer or if any legal issues may arise. It can be determined by four different attributes: amount, quality, completeness of documentation and availability of appropriate legal terms and conditions.

Complexity - The complexity of a software component determines how usable it is (i.e., easy to understand and to maintain) and how easy it is to adapt the software component in the new context of use. The complexity of a component can be measured by the size of the component, no. of loops, nested computation etc. If the Complexity of the component increases, more difficult is to reuse that component. Complexity metric depends on classes, methods and parameters of the component.

Quality - The quality of a component describes how good it fulfills requirements and also how error-free and bug-free it is. If a component is error-free and bug-free, it can be used again. Quality of a component can be accessed via four attributes: the number of bugs, the number of tests performed, availability of test cases, and an independent rating and certification.

Maintainability - The maintainability of a software component directly determines how easy it is to adjust the component to a new context. A component must be able to fit its behavior according to changes in the environment or in parts of system to be used again.

Price - The price of the software component determines how expensive it is to reuse. A generic, objective and quantitative metric can be evaluated, expressed through a predefined currency.

Completeness - The degree to which the component implements all required capabilities.

Correctness - The ability of a component to produce specified outputs when given specified inputs, and the extent to which they match or satisfy the requirements.

Efficiency - The degree to which a component performs its designated functions with minimum consumption of resources and less delay in execution time. Higher the efficiency of component higher will be the reusability of components.

Generality - The degree to which a system or component performs a broad range of functions.

Modularity – The degree to which a system or computer program is composed of discrete components (modules) such that a change to one component has minimal impact on other components. Each component contains everything necessary to execute one aspect of desired function. High modularity leads to high reusability.

Portability- The extent to which a module originally developed on one computer or operating system can be used on another computer or operating system. If a component is portable, it can run on different platforms which increase the probability of component to be reusable.

Reliability - The ability of a component to perform its required functions under stated conditions for a specified period of time.

Cohesion - The degree to which the functions or processing elements within a module are related or bound together. Cohesion increases if the methods in the module perform same function. High cohesion leads to high reusability of component.

Coupling - The degree that modules are dependent upon each other in a computer program. Tight coupling between components reduces reusability of components. In tightly coupled modules, a change in one module forces a change in other modules which leads to extra effort and time to assemble these components.

Hence, these components cannot be used as reusable components.

Expandability - The extent to which a component allows new capabilities to be added and existing capabilities to be easily tailored to user needs. It measures the level of effort and cost required to implement the system with new capabilities. If the component is easily expandable, it can be easily reusable.

Testability - The ability to evaluate conformance with requirements. The testability of the component is critical when reusing the software. A well-defined set of test cases aids in quickly assessing the components use in a new environment. The testability of a component is defined in part by its complexity, as well as its size.

After studying various factors affecting reusability, it can be concluded that components must be well defined and understandable by the software engineers and encapsulate as much implementation detail as possible. Fault density, code-related problem counts, defect density, and efficiency are some of the metrics used for assessing existing components for reusability. The longer a component has been in actual use; higher will be the confidence of the component's correctness. Cohesion and Coupling are the most important factors to measure the reusability of component. Cohesion and Coupling are complementary to each other. High Cohesion leads to Low Coupling between modules, hence increases reusability. To measure the reusability of component we need to calculate the reusability metric.

4. Related Work

Reusability is the measure of extent upto which we can make reuse of component. For the sake of practicality, reusability metric can be separated into two categories: one for white-box, which allow to look into the code of the components and one for black-box (where merely interface and documentation of a component are available) reusability. This separation helps to distinguish the different nature of metrics for these two paradigms.

Prieto-Diaz and Freeman checked white-box reuse and identified some program attributes for evaluating reusability [3]. Attributes used are: Program Size, Program Structure, Program credentials, Programming Language, and Reuse Experience.

Caldiera and Basili [4] in 1991 defined three main attributes for assessing the reusability of components – reuse costs, functional usefulness and quality of components.

Seven years later, Barnard [5] suggested a composite metric for reusability of object-oriented software, which was derived from two empirical experiments. As foundation, again a variety of readily available software metrics have been used. Based on the experiments, those metrics that were related best to reusability have been selected. [5] Focused on the Simplicity, Generosity and Understandability of class interfaces, methods and attributes.

Around the same time, Mao et al. [6] investigated the effects of inheritance, coupling and complexity on the reusability of classes in object-oriented software. Two years later, Lee and Chang [7] proposed another set of metrics for measuring the reusability and maintainability of object-oriented software. In 2001, Cho et al. [8] suggested metrics for component complexity, customizability, reusability and reuse. Component Reusability is determined by the functionality that the software components provide for their domain: it is the ratio between the number of interface methods in the component that provide common functions in the domain, and the total number of interface methods in the component. The more the common functions a component provides, more its

reusability is considered. Also in 2001 Etzkorn et al. [9] have published a model capturing reusability of object-oriented legacy software. They suggested a comprehensive metric suite covering different aspects of the reusability of individual classes. It is defined as the sum of metrics for Modularity, Interface Size, Documentation and Complexity of a class, each equally weighted.

Four years later, Bhattacharya and Perry [10] suggested reusability metrics that measure how well a component fits in a predefined architectural context.

In 2008, Gui and Scott [11] suggested revised formulas for established coupling and cohesion metrics in order to measure the reusability of Java components. Gill and Sikka [12] proposed five new metrics for better assessing reuse and reusability in object-oriented software development. The metrics are Breadth of Inheritance Tree, Method Reuse per Inheritance Relation, Attribute Reuse per Inheritance Relation, Generality of Class and Reuse Probability.

Reusability of a component can be measured by Coupling or Cohesion metrics. Number of authors has proposed metrics for coupling and cohesion.

Coupling metrics

Page Jones [13] introduced the concept of tramp coupling, where data may flow between many intermediate modules from where data is defined to where data is used. This metric measures the coupling among many modules instead of just two modules.

Classification produced by Myers [14] is used by Fenton and Melton [15].They proposed the following metric as a measure of coupling between two components x and y:

$$\mathbf{C}(x,y) = i + n/(n+1)$$

Dhama [16] proposed a coupling metric that measures the coupling of an individual component C, which is equal to:

$$\mathbf{C} = 1/ (i1 + q612+u1+q2u2+g1+q8g2+w+r)$$

Cohesion metrics

Cohesion metrics measure how well the methods of a class are related to each other. A cohesive class performs one function.

LCOM1 was introduced by Chidamber & Kemerer [17], and it was calculated as follows:

Take each pair of methods in the class. If they access disjoint sets of instance variables, increase P by one. If they share at least one variable access, increase Q by one.

LCOM1=P-Q, if P>Q LCOM1=0 otherwise

LCOM1 = 0 indicates a cohesive class.

Further two additional metrics have been proposed: LCOM2 and LCOM3. A low value of LCOM2 or LCOM3 indicates high cohesion and a well-designed class likely to have high reusability.

The higher TCC and LCC, the more cohesive and thus better the class. For TCC and LCC we only consider visible methods. A method is visible unless it is Private. A method is visible also if it implements an interface or handles an event.

TCC = NDC/NP

LCC = (NDC+NIC) / NP

6. Limitations

Most of the metrics proposed till date considers direct coupling only. If coupling was considered, values were considered on the bases that coupling exists or not. No in-between values are taken, like upto what extent coupling is there. There is no metric considering both procedure oriented and object oriented approach. Many works

also face lack of validation, or week validation criteria.

Conclusion

This paper presents a survey on software code metrics, providing an overview on what has been done in recent years. It will also help researchers to get a comprehensive view of the direction of works in area of software reusability measurement. Most of the earlier metrics were validated in theory, metrics range definition. Reports do not show the success range of the metric. Experiments were done on few data set, poor details in software metrics reports, many reports have few information about the metric usage. In this survey we extracted number of factors on which reusability depend and based on this survey new metric will be constructed to analyze source code components quality in context of reusability. And we will try to construct a metric for both procedure and object oriented approach.

References

[1] Tran Vu N. and Liu Dar-Biau. Application of CBSE to Projects with Evolving Requirements – A Lesson-learned.

[2] Voas J., COTS Software: the Economical Choice?, IEEE Software, volume 15, issue 2,1998.

[3] Prieto-Diaz, Ruben Freeman, P., 1987 “Classifying Software for Reusability”, IEEE Software

[4] G. Caldiera and V.R. Basili, Identifying and qualifying reusable software components, IEEE Computer, vol.24, Feb.1991.

[5]J. Barnard, A new reusability metric for object-orientedsoftware, Software Quality Journal, vol. 7, Jan. 1998, pp.35-50.

[6] Y. Mao, H. Sahraoui, and H. Lounis, Reusability HypothesisVerification using Machine Learning Techniques: A CaseStudy, Proceedings of the International Conference onAutomated software engineering, IEEE, 1998, pp.84-93.

[7] Y. Lee and K.H. Chang, Reusability and maintainabilitymetrics for object-oriented software, Proceedings of the 38thannual on Southeast regional con-ference (ACM-SE 38),ACM, New York, NY, USA, 2000, pp.88-94.

[8] E.S. Cho, M.S. Kim, and S.D. Kim, Component Metrics toMeasure 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.

[9] L.H. Etzkorn, W.E. Hughes Jr., and C.G. Davis, Automatedreusability quality analysis of OO legacy software,Information and Software Techn., vol.43, 2001, pp. 295-308.

[10] S. Bhattacharya and D.E. Perry, Contextual reusabilitymetrics for event-based architectures, Intern. Symp. onEmpirical Software Engineering, 17-18 Nov. 2005, pp.459-468.

[11] G. Gui and P.D. Scott, New Coupling and Cohesion Metricsfor Evaluation of Software Component Reusability, Proc. of the Intern. Conf. for Young Computer Scientists, 2008,pp.1181-1186.

[12] N. Gill and S. Sikka, Inheritance Hierarchy Based Reuse &Reusability Metrics in OOSD, International Journal onComputer Science and Engineering (IJCSE), vol.3, June2011, pp.2300-2309.

[13] Page Jones , The Practical guide to structured System design, YOURDON press , newyork NY 1980.

[14] G. Myers, Composite/Structured Design. Van Nostrand Reinhold, 1978.

[15] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous & Practical Approach. 2nd edn. Reading, 1997.

[16] H. Dhama, ―Quantitative Models of Cohesion and Coupling in Software‖, Journal of System and Software, 9(1)(1995), pp. 65–74.

[17] Shyam R. Chidamber, and Chsis F. Kemerer, ―A Metrics Suite For Object Oriented Design,‖ IEEE Transactions On Software Engineering, 20(6)(1994), pp. 476– 493.

[18] Hutches and V. R. Basili, ―System Structure Analysis: Clustering with Data Bindings‖, IEEE Transactions on Software Engineering, 11(8)(1985), pp. 749–757.

[19] W. Li and S. Henry, ―Object-Oriented Metrics that Predict Maintainability‖, Journal of Systems and Software, 23(2) (1993), pp. 111–122.

[20] J. Chen, and J. Lu, ―A New Metric for Object-Oriented, Design‖, Information and Software Technology, 5(4)(1992), pp. 232–239.

[21] Brian Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity. New York: Prentice Hall PTR, 1996.

[22] Briand, L., Devanbu, W., Melo W.; (1997), "An investigation into coupling measures for C++", 19th International Conference on Software Engineering, page(s): 412-421, Boston, USA.