Volume 3, Issue 11, November 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Software Quality Improvement through High Cohesion (HC) & Low Coupling (LC)

S.V. Achuta Rao
Research Scholar, CSE, Krishna University
Machilipatnam , India.

Kiran Kumar Reddi
Department of CSE, Krishna University
Machilipatnam, India.

Abstract: Software Quality is an important factor for any Software Organization. Demand for Software has undergone with rapid growth during the last few years. Software Development team tries to increase the software quality by decreasing the number of defects as much as possible. In this paper we explain the importance of cohesion and coupling in object oriented metrics, help to deliver a defect free software and improve the quality.

Keywords: Defect Prediction, Object Oriented Metrics, Software Quality, Software Development.

I. INTRODUCTION

A Software defect is an error, flaw, mistake, failure, or fault in a computer program or system that produces an incorrect or unexpected result, or causes it or behave in unintended ways. Software defects are expensive in terms of quality and cost. Moreover, the cost of capturing and correcting defects is one of the most expensive software development activities. It may not be possible to eliminate all defects, but it is possible to minimize the number of defects and their severe impact on the projects. Nowadays, High Technologies are taking more and more important role in the process of taking the most important decisions. It is very important to deliver Quality Software with Conformance [12].

Conformance of Quality : " User satisfaction + Confirmation of functional & performance requirements + delivery within budget and time"

Software Quality in particular is a dynamic collection of ideas, best practices and procedures. Human mistakes resulting in faults are mostly associated with design or Implementation such mistakes may originate at a number of levels - Inception, Understanding, Implementation and Execution [2].

Object Oriented Software Design Metrics play an important role in software defect prediction If it use properly, software engineering metrics allow us to quantitatively define the degree of success or failure of a product, process, or project. These can also used to take meaningful and useful managerial and technical decisions related to cost, effort, time and quality etc., Design Metrics play an important role in helping developers to understand design aspects in Coupling, Cohesion, Inheritance and Size are the independent variables in the Development. Software Quality must be addressed during the whole process of Software Development. Measuring Software Quality in the early stages of Software Development is the key to develop High Quality Software. Product Quality has some attribute such as Functionality, Usability, Effectiveness, and Understandability, Reusability and Maintainability. Large Number of Software Metrics have been proposed in software engineering to measure the quality attributes of the software in early stages [1, 3, 5]. It is imperative to predict and fix the defects as many as possible before the product delivers to the customer. Much effort is spent within the software engineering community on proposing technologies (processes, methods, techniques, principles) that are believed to improve the efficacy of developing and maintaining Quality Products. The relative strengths and weaknesses of the proposed technologies need to be evaluated in such a way that the interplay between tasks, technology, developers and their organization can be better understood. Empirical studies in realistic settings are thus required to understand *when*, *how* and *why* the proposed technologies might be beneficial. In turn, the scientific knowledge obtained from the empirical studies is fundamental to improve the proposed technologies and to ensure that they are used in a cost-effective manner in an industrial context.

II. RESEARCH QUESTIONS AT SOFTWARE DEVELOPMENT TO IMPROVE QUALITY

Software development can be seen as a sequence of changes a constant stream of activities that add new value to software, adapt it to a changing environment, delete features no longer required, or improve its structure for better maintenance. All of these activities are ultimately conducted by humans, and as humans make mistakes, it is unavoidable that some of these changes will induce defects.

The 5 common questions in roots of software development. We segregating these problems based on different stages of an SDLC [9, 14 , 15]

Question #1: Requirements gathering

Many projects still follow Waterfall SDLC. If the requirements are not very clear and exhaustive, then the team will find it difficult to proceed on schedule. The team will find it especially difficult to provide deliverables as per customer's expectations since the requirements were unclear or incomplete at the beginning. You may be saved if you decide to go for Agile SDLC, but then that again requires that the end-user involvement will high, Else, requirements gathering will suffer.

Question #2 Scheduling & Estimation

Many a time project managers cram too much work within too less man-hours either because of inexperience or because of other constraints such as limited budget or lack of clarity in customer's business. Bad scheduling is bound to have a ripple effect on development, testing and deployment stages of an SDLC.

Question #3 Development

Mistakes committed during the above two stages cause problems that are inevitable during the development stage. Unclear requirements at the beginning may mean that you have to be ready for lot of feature creep-in. The business side of the customer may consider adding features during development stage as minor changes. But it may throw the whole development team very much off the schedule and cost estimates if the team has to keep revising the code to add features often. Similarly, bad project schedule estimation will build pressure on the developers, which means more scope for bugs.

Question #4 Testing & How To Improve Software Testing?

Due to bad scheduling it may be possible that there is not enough time for testing, bug fixing and re-testing. It is not possible to have bug free software development. So an overly optimistic schedule (one of the mistakes considered to have serious to catastrophic impact on software development projects) will most certainly create problems as there will not be adequate testing done before deployment.

The application of process improvement tools to the software development life cycle is becoming popular in the software community. These techniques have already been successfully leveraged by manufacturers which have encouraged software professionals to apply such tools to the SDLC. It can help identify the potential causes of a problem, suggest suitable corrective action and offer insight into preparing test case scenarios.

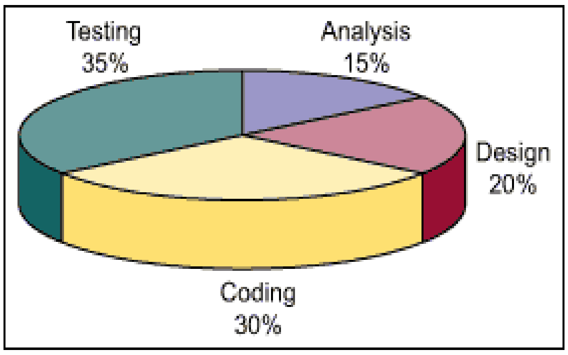
2

Challenges in Software Testing

4.1. Inherent challenge: It is next to impossible to test a software product of average complexity to all of its specifications and features. The number of test cases required to test every aspect of a software application would be so large that it would be economically impossible to prepare and execute. For example, a simple program to analyze a string of 10 alphabetic characters could have 26^{10} combinations. At a rate of one combination per microsecond, testing the string would take 4.5 million years, according to author Watts S. Humphrey in his book *Managing the Software Process*.

4.2. Laborious process of test case preparation and documentation: Test case preparation is labor intensive and has to fit into what is normally an already tight schedule. Often project teams are tempted to pay less attention to this activity. Considering the large number of test cases to be developed, it takes considerable effort to document and maintain the documentation. The project team seldom documents all the test cases and has to conduct testing with additional undocumented test cases.

4.3. Effectiveness of test cases: Identifying a test case is as important as writing a line of code. Lack of proper methods makes this task more challenging. Software engineering researchers, such as Glenford J. Myers in his book *Software Reliability, Principles and Practices*, have observed that it is impossible to test one's own programs. Test cases for a module created by the software developer tend to have an ingrained bias toward an application's functionality. Such test cases often are prepared to prove what is being developed, instead of to reveal defects – the proper objective of test cases.



SDLC Effort Distribution by Phases

4.4. Resource crunch: More effort is spent in software testing than in any other phase of software development.

Figure 1 shows the distribution of efforts in the software development life cycle. The percent distributions are typical industry averages. While specific amounts of effort are scheduled for testing, projects often end up with less testing time than planned because the design and construction phases consume more effort than estimated. Tools that may increase effectiveness of testing are unavailable or unnoticed. Even if tools are available, a project team faced with a new learning curve may not be inclined to use them.

Question #5 Communications and Collaboration

Though communication and collaboration are not the stages of an SDLC, communication is the backbone for collaborating and getting things done. Whether communication is done through documentation, stand-up meetings and virtual brainstorming or through web-based SDLC management tools, information sharing and knowledge exchange is required for the project to meet quality standards and deliver on schedule.

III. TRADITIONAL METRICS

In an object-oriented system, traditional metrics are generally applied to the methods that comprise the operations of a class. A method is a component of an object that operates on data in response to messages and is defined as part of the declaration of a class [14,15]. Methods reflect how a problem is broken up and the capabilities other classes expect of a given class.

Three traditional metrics are discussed here:

1. Cyclomatic complexity
2. Line counts (size).
3. Comment Percentage

IV. OBJECT ORIENTED DESIGN METRICS

Object Oriented Design and Development is an interesting area of current research and many authors have done great deal of work in recent years. In fact Object Oriented Development requires not only a different approach to design and implementation, but also a different approach to software metrics. Software metrics make it possible for software engineers to measure and predict software processes, necessary resources for a project and products relevant for a software development effort. A software measure provides software engineers with a means of quantifying the assessment of a software product[19]. Measurement can be used throughout a software project to assist in estimation, productivity assessment, quality control and project control. The design of complex software based systems often proceeds with virtually no measurement. The introduction of Object Oriented Methods to software development has changed the process of building and managing software in a profound way. Changes in design and implementation techniques also require new ways of measuring software systems. Design Metrics (Albert Dieter Ritzhaupt, 2004) play an important role in helping developers to understand design aspects of software techniques and, hence, improve software quality and developers productivity [13, 14]. In addition, the focus on process has increased the demand for software measures or metrics with which one can manage the process. The need for such metrics is particularly rare, when an organization is adopting a new technology for which a suitable metric design plays a crucial role. We present existing and new software metrics useful in the different phases of the Object Oriented Software Development cycle[10,11]. The importance of properly defined metrics is of immense importance in the Object Oriented design. If the metrics are properly defined, we can avoid problems that will be more expensive to correct during the latter phases of Object Oriented Software Development [11,12]. This helps researchers and practitioners better in Understanding and selects software metrics suitable for their purposes. In today's software industry, the aim is to deliver high quality software product to the customers. No doubt that the software quality can make or break a company. So software Quality plays a vital in the software industry

V. ROLE OF COHESION IN OBJECT ORIENTED SYSTEMS

Cohesion is an important attribute corresponding to the quality of the abstraction captured by the class under consideration. Good abstractions typically exhibit high cohesion. The original object oriented cohesion metric as given by Chidamber and Kemerer (and clarified by the same authors) represents an inverse measure for cohesion. They define Lack of Cohesion in Methods (LCOM) as the number of pairs of methods operating on disjoint sets of instance variables, reduced by the number of method pairs acting on at least one shared instance variable [12, 13].

Chidamber and Kemerer also define RFC (Response for a Class) as the union of the protocol a class offers to its clients and the protocols it requests from other classes. Measuring the total communication potential, this measure is obviously related to coupling and is not independent of CBO.

Strength 1:

Accessing the interface of any server class SC, provided SC is a stable class or features at least a stable interface, the most harmless type of Class coupling occurs, as no change dependencies are introduced.

Strength 2:

Changing the interface of an SC method called via an object local to one of CC's methods, only this latter method needs to be changed correspondingly. The same argument applies to the case where SC is the type of a parameter of a CC method.

Strength 3:

Changing the interface of an SC method invoked via a message sent to one of CC's instance variables of class SC, due to the class scope of instance variables, potentially all methods of CC are affected. This is why this case is less favourable than the above. Similarly, changing the interface of a method of the super class SC of CC affects all methods of CC calling this super- class method. Thus, again potentially all methods of CC may be affected. As a global variable is accessible from all methods of a class, the same argument applies for global variables, too.

Strengths 4 & 5: Following the same arguments as for strengths 2 and 3 and noticing that change dependencies are generally stronger when breaching the information hiding principle, these assignments result.

For object oriented software, the coupling has not been considered with similar priorities.
There are two main reasons for this negligence:
1.In structured design, there were few semantic guidelines to decompose a system into smaller subsystem. Consequently, syntactic aspects like size, coupling etc. played a major role. In contrast, in the object-oriented paradigm, the main criterion for systems decomposition is the mapping of objects of the problem domain into classes or subsystems in the analysis/design model, thus reducing the relative importance of syntactic criteria.
2. Object-oriented analysis and design strive to incorporate data and related functionality into objects. This strategy in itself certainly reduces coupling between objects. However, since employing object-oriented mechanisms in it-self does not guarantee to really achieve minimum coupling.

Role of Coupling in object-oriented systems:
1. In many cases, data or operations cannot be unambiguously assigned to one or another class on the grounds of semantic aspects, thus designers does need some kind of additional criteria
for such assignments.

2. Although introduction of classes as a powerful means for data abstraction reduces the data flow between abstraction units and therefore reduces also total coupling within a system, the number of variants of interdependency rises in comparison to conventional systems.
3. The principles of encapsulation and data abstraction, although fundamental to object-orientation, may be violated to different extents via the underlying programming language. This leads to different strength of de-facto coupling which should be taken into account.

VI. CONCLUSIONS

In this paper, we have discussed that how high cohesion & low coupling helps in Object Oriented Design Software Metrics to improve software Quality. We have also studied in previous papers that how these techniques have performed better results when performed on different source code data sets. In future, we will compare the results of the techniques with important methods for reliable software systems based on Fault prevention, Fault removal, Fault tolerance and Fault forecasting – to estimate the presence of faults and the consequence of failures.

REFERENCES

[1] ESPQ: A New Object Oriented Design Metric for Software Quality Measurement – International Journal of Engineering Science and Technology, Vol. 2(3), 2010, 194-208
[2] The Prediction of faulty classes using Object-Oriented Design Metrics – ELSEVIER – The Journal of Systems and Software - 56- (2001) – 63-75
[3] Software Fault Prediction: A Software Fault Prediction Model by Hybrid Feature Selection and Hybrid Classifier Approach By Akalya Devi, http://www.biblio.com
[4] A Thesis on " "Design Metrics for Object-Oriented Software Systems" by Fernando Brito e Abreu, PORTUGAL K.El Emam, W. Melo, J. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", *Journal of Systems and Software*, vol. 56, 63-75, 2001.
[5] S. Chidamber and C.Kemerer, "A metrics Suite for Object-Oriented Design *",IEEE Trans. Software Engineering*, vol. SE-20, no.6, 476-493, 1994.
[6] M-H. Tang, M-H. Kao, and M-H. Chen, ªAn Empirical Study on Object Oriented Metrics, Proc. Sixth Int'l Software Metrics Symp., pp. 242-249, 1999.
[7] L. Briand, J. Wuest, J. Daly, and V. Porter, ªExploring the Relationships Between Design Measures and Software Quality in Object-Oriented Systems,º J. Systems and Software, vol. 51, pp. 245- 273, 2000.
[8] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, ªA Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study, Technical Report ISERN-98-29, Int'l Software Eng. Research Network, 1998.
[9] Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi. K" An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems" International Journal of Emerging Trends in Engineering and Applied Sciences (IJETEAS) 1 (2): 216-224 (ISSN: 2141-7016).
[10] Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi," Metrics for Object Oriented Design Software Systems: A Survey "International Journal of Emerging Trends in Engineering and Applied Sciences (IJETEAS) 1 (2): 190-198 (ISSN: 2141-7016).
[11] R.S. Pressman; "*Software Engineering - A practitioner's approach",* sixth edition, Mc Graw Hill, 2005.
[12] I. Sommerville; *"Software Engineering",* 7th edition, Addison Wesley, 2004.
[13] El Emam, K.; Benlarbi, S.; Goel, N. and Rai, S. N; *" The confounding effect of class size on the validity of object-oriented metrics",* IEEE Transactions on Software Engineering, 27(7) - 630-650, 2001.
[14] N. Fenton et al.; *"Software Metrics - A Rigorous and Practical Approach",* International Thomson Computer Press, 1996.

Authors

Dr. Kiran Kumar Reddi has received PhD in Computer Science and Engineering from Acharya Nagarjuna University,Guntur,Andhra Pradesh,India. He is working as Assistant Professor in the Department of Computer Science,Krishna University,Machilipatnam, Andhra Pradesh, India. His research interest includes Bioinformatics, Data Mining, Image Processing Software Engineering and Network Security. He is a member of professional societies like IETE and ISTE.

S.V. Achuta Rao is Research *Scholar* under the guidance of Dr. Kiran Kumar Reddi in Krishna University, Machilipatnam, India. He has received M.Tech.(Computer Science and Engineering) from JNTU, Kakinada, India. He has been published and presented more than 15 numbers of Research and technical papers in International and National Conferences & Journals. His main research interests are Software Engineering and Software Metrics.