Measuring Reliability of an Aspect Oriented Software using Fuzzy Logic Approach

Reena Dadhich, Bhavesh Mathur

Abstract— Aspect Oriented Programming (AOP) refers to the programming approach which isolates secondary and supporting functions from the main program's business logic. The application's modularity increased in that way and its maintenance becomes significantly easier. Apart from the functional requirements while developing any software, we should also consider some non functional requirements like Reliability, Adaptability, and Suitability etc. In this paper an attempt has been made to quantifying the reliability of aspect oriented software using ISO/IEC 9126 Model. Due to the unpredictable nature of software quality attributes, the fuzzy multi criteria approach has been used to evolve the quality of the software.

Index Terms— Aspect Oriented Programming (AOP), Cross Cutting Concerns, ISO/IEC9126 Model, Reliability

I. INTRODUCTION

Aspect oriented programming defines a new program construct, called an *aspect*, which is used to capture cross-cutting aspects of a software application in separate program entities..Aspect Oriented Programming comes into picture because of some limitation of Object Oriented Programming. In Object Oriented Programming there are parts of a system that cannot be viewed as being the responsibility of only one class, they cross-cut the complete system and affect parts of many classes. Examples might be locking in a distributed application, exception handling, or logging method calls. Of course, the code that handles these parts can be added to each class separately, but that would violate the principle that each class has well-defined responsibilities. This problem is solved by using Aspect Oriented Approach of Software Development [17].

Software quality is a very important aspect for developers, users, and project managers. Various researchers have worked in developing suitable models that define software quality in different perspectives as described in ISO/IEC 9126 Model [1], Boehm's Model [3], Dromey's Model [4] and the FURPS Model [5]. Quality, not only describes and measures the functional aspects of the software (what a system does), but also describes extra functional properties (how the system is built and performs). Different software quality models were proposed by various researchers in [2-5]. These models are proposed for generic software applications. Out of these models, ISO/IEC 9126 model [2] is the most prominent model, which includes the findings of almost all other models. This is widely accepted and recognized in the industry and research community. Researchers made several efforts to implement this model for component based systems with minor modifications. This present work attempts to quantify the software reliability using the ISO/IEC 9126 Model [1] as the base model with appropriate modifications to it. In order to deal with the fuzziness or uncertainty in quantifying the actual software parameters, the f*uzzy multi criteria* approach has been used. We used fuzzy logic approach to measure the reliability of aspect oriented java application.

Manuscript Received on June 15, 2012.

Dr. Reena Dadhich, Department of MCA, Government Engineering College, Ajmer, India.

Bhavesh Mathur, Department of Computer Engineering, Government Engineering College, Ajmer, India.

II. RELATED WORK

Currently, one of the important aspects of research in the field of Software Engineering is the "Quantification of Parameters Affecting the Software Quality." Various researchers have made attempts to quantify the software quality criteria [6-8]. Sharma et al. [8] had considered the Component Based Software Development Model to quantify the software quality criteria mentioned in the ISO/IEC 9126 model [1] with minor modifications. They used the Analytical Hierarchy Process (AHP) model and assigned weights to the software quality criteria to get the actual software quality quantified. P. R. Srivastava et al. [6] have also considered quantifying the software quality parameters in developer's, user's, and project manager's perspectives and then took the weighted average for all of these factors to get the actual software quality .S.A. Slaughter et al. [9]has made an attempt to evaluate the cost of software quality . M. Agarwal and K. Chari [10] had considered the software quality in terms of quality, effort, and cycle time . O. Maryoly, M.A. Perez, and T. Rojas [11] developed a systematic quality model for developing and evaluating the software product . Various characteristics and sub characteristics affecting the software quality have been quantified by using metrics to evaluate the software quality. Lamouchi Olfa Amar R. Cherif, and Nicole Lévyalso[12] attempted to quantify the software quality factors by subdividing the factors into criteria and sub criteria and by quantifying the metrics that are affecting them. They have elucidated their approach clearly by showing an example of quantifying *portability*. Y. Kanellopoulos et al [13] evaluated the code quality using various metrics with the help of the Analytical Hierarchy process model. They tried to evaluate the internal quality, which includes the characteristics - functionality, efficiency, maintainability and portability. I. Heitlager et al. [14] emphasized estimating software quality based on maintainability and R.Fitzpatrick et al. [15] have tried to estimate software quality by mainly emphasizing usability.

III. ASPECT ORIENTED PROGRAMMING

Beyond modularizing crosscutting concerns with the design and implementation perspective, AOP enables specific applications that improve the quality of software. Implementing software application using AOP improves software quality in many ways,

ike in term of better understanding, higher productivity and cost savings. Software developer can apply AOP approach for improving quality in design and implementation perspective as discussed by Ramnivas Laddad [2].AOP enables specific application that improve the quality of software.

System-wide policy enforcement- With AOP, you can create reusable aspects that enforce a variety of contracts and provide guidance in following "best" practices. For example, the Enterprise JavaBeans specification is about 600 pages long and describes many restrictions programmers should adhere to. Developer diligence alone can seldom achieve the required enforcement level [2].

Logging-fortified quality assurance- The inability to reproduce a bug is one of the biggest frustrations for every software engineer. By enabling logging functionality, you can enable a better QA process; the QA person can augment the bug report with the associated log and thus give developers a better chance at reproducing the behavior. Nonintrusive logging and tracing functionality is a helpful tool in understanding system behavior. The conventional implementation of the equivalent functionality is so cumbersome that systematic implementation rarely happens [2].

Better simulation of the real world through virtual mock objects- Using mock objects is a valuable AOP application that helps to test software quality. Complex scenarios often aren't tested because of the effort required to simulate faults such as a network failure. When you combine mock objects with AOP, you can introduce the mock objects without making any changes to the core system. You can then easily inject faults into the system and test how it responds. AOP makes such sophisticated testing easy and practical without requiring complex test harnesses or compromising the core design for testability [2].

Nonintrusive what-if analysis-_A pooling and caching optimization isa time and space trade-off that often needs to run what-if scenarios before deciding whether functionality is necessary. AOP provides a nonintrusive way to perform such an analysis without worrying about accidental changes to core behavior.[2].

IV. SOFTWARE QUALITY

The study of software quality involves a planned and systematic set of activities to ensure the effectiveness of software. It consists of various sub topics like software quality assurance, quality control, and quality engineering. According to the IEEE 610.12 standard [16], software quality is a set of attributes of a software system and is defined as:

1. The degree to which a system, component, or process meets specified requirements.
2. The degree to which a system, component, or process meets customer or user needs or expectations.
3. Quality also comprises of the factors leading to the satisfaction of its requirements.

The quality of the software is measured in terms of its capability to fulfill the needs of the users and also its ability to achieve the developer's goals. Quality is mainly studied by quality models. The quality model describes the set of characteristics, which are the basis for establishing the quality requirements and for evaluating software quality. In the present paper, the ISO/IEC 9126 Model [2] has been considered as the base model.

A The ISO/IEC 9126 Model

ISO (International Standard Organization) proposed a standard, known as the ISO/IEC 9126 Model [1], which provides a generic definition of software quality in terms of six main characteristics for software evaluation. These characteristics are functionality, efficiency, maintainability, portability, reliability and usability. The model covers almost all of the aspects covered in previously proposed models such as Boehm's model [3], McCall's model [4], Dromey's model [5], etc. It covers both the internal and external quality characteristics of a software product. It does not however describe how these characteristics and sub characteristics can be quantified.

B Reliability

Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability [18].

In this paper we have measured Reliability, Adaptability and Suitability for Aspect oriented Application.

Reliability has measured using its sub characteristics which are

(i) Maturity
(ii) Fault Tolerance
(iii) Recoverability
(iv) Reliability compliance

V.ASSUMPTIONS

- The values of all the parameters or characteristics have been quantified in the range 0 to 1. The overall quality of the software after quantification also appears in the range of 0 to 1.
- Various characteristics and sub characteristics have been prioritized appropriately to calculate the total quality of the software. The weights considered vary from case to case.
- Both ratings and weights have been quantified in terms of fuzzy, which are then converted into crisp numeric values using the Centroid Formula.
- The fuzzy weighted average of all the quantified criteria and sub criteria is taken in order to arrive at the final quality. This has been done to maintain consistency so that the range of final values lies between 0 and 1.

VI. PROCEDURE

The exact procedure to quantify the software quality has been described in this section. As it has already been discussed in Section 3, the software quality is evaluated on the basis of the Software Quality Model that has been derived from the ISO/IEC 9126 Quality Model [1].

The procedure to quantify the software quality is as follows:

Step 1: Assign fuzzy ratings (r_i) to each and every metric that exists in the software model.

Step 2: Assign fuzzy weights (w_i) to the sub characteristics, characteristics and perspectives.

Step 3: Take the weighted average of the metrics (using their weights and ratings).

Step 4: Take the weighted average of the sub characteristics for Reliability (using their weights and ratings) under the corresponding characteristics to evaluate the fuzzy rating.

VII. MEASURING RELIABILTY FOR ASPECT ORIENTED JAVA APPLICATION

Let us consider a Library application developed in Java which provides web access to College Library. A user can browse the various categories of books, add some books to a cart and finally checkout, do payment and get the books. For this app we might receive the requirements from a business analyst as follows:

- A login/registration screen to enter into Library.
- Users should be able to browse through various categories of books
- Users should be able to search books by name, author name, publisher
- Users should be able to add/remove the books to/from their cart
- Users should be able to see what items currently exist in their cart
- Users should be able to checkout and pay the corresponding amount through some payment gateway
- A successful message should be shown to users with all the details of their purchases.
- A failure message should be shown to users with the cause of failure.
- A Library administrator/manager should be granted access to add/remove/update book details.

All the above requirements fall under the "Functional Requirements" category. While implementing the above, we should also take care of the following things even though they are not explicitly mentioned:

- **Role based access to the UI**. Here, only Administrators/Managers should have access to add/remove/update book details. [Role based Authorization]
- **Atomicity in Purchasing.** Suppose a user logged into the Library and added 5 books into his cart, checked out and completed his payment. In the back-end implementation we may need to enter this purchase details in 3 tables. If after inserting the data into 2 tables the system crashed, the whole operation should be rolled-back. [Transaction Management].

• **No one is perfect and no system is flawless.** So if something went wrong and the development team has to figure it out what went wrong, logging will be useful. So, logging should be implemented in such a way that developer should be able to figure out where exactly the application failed and fix it. [Logging]The above implicit requirements are called Non-Functional Requirements. In addition to the above, performance should obviously be a crucial non-functional requirement for all public facing websites.

A. Calculation of Reliability for Aspect Oriented Library application

Table 6.1.1 shows the real time values of the metrics related to *reliability*. The values of these metrics have been acquired from five different users on the basis of a questionnaire.

Table 6.1.2 shows the ratings of the metrics corresponding to the *reliability* characteristic, after they have been fuzzified on the basis of the criteria discussed in Sections 6. After classifying the metrics in the corresponding fuzzy sets, they have been assigned appropriate triangular fuzzy numbers as shown in the table 6.1.2.Table 6.1.3 shows the values of the weights that have been taken from five users. These weights have also been acquired via a questionnaire. This table also shows the fuzzified value of the weights after taking their average. Now the ratings (r_i) of the metrics (belonging to *reliability*) have been multiplied by corresponding weights (w_i) and then added together to get the ratings of the corresponding sub characteristics.

Table 6.1.1 Values of Real time Metrics for the Reliability Characteristic

Sub Characteristics (Reliability)	Questions (Metrics)	U1	U2	U3	Ratings
Maturity	No. of version released so far.	L			(0.0,0.1,0.3)
Fault Tolerance	Exception handling provided or not.	VH			(0.8,0.9,1.0)
	Total Number of Functionalities successfully met/total number of Functionalities Available	VH			(0.85,0.9,1.0)
Recoverability	Availability of data backup	VH			(0.8,0.9,1.0)
Reliability Compliance	Whether software adheres to reliability compliance standard or not	H			(0.6,0.8,0.9)

Table 6.1.2 Fuzzy Ratings of the Metrics Belonging to Reliability Characteristic

Sub Characteristic (Reliability)	Questions (Metrics)	U1	U2	U3	Ratings
Maturity	No. of version released so far.	L			(0.0,0.1,0.3)
Fault Tolerance	Exception handling provided or not.	VH			(0.8,0.9,1.0)
	Total Number of Functionalities successfully met/total number of Functionalities Available	VH			(0.85,0.9,1.0)
Recoverability	Availability of data backup	VH			(0.8,0.9,1.0)
Reliability Compliance	Whether software adheres to reliability compliance standard or not	H			(0.6,0.8,0.9)

Table 6.1.3. Fuzzy Weights of the Metrics Belonging to the Reliability Characteristic

Sub Characteristic (Reliability)	Metrics (weight)	U1	U2	U3	Weights
Fault Tolerance	Relative importance for Exceptional Handling	VH	H	M	(0.75,1.0, 1.0)
	Relative importance for Functionalities that have been successfully met	H	H	M	(0.50,0.75 ,1.0)

$$r_{Reliability} = r_1 \times w_1 + r_2 \times w_2 + \ldots r_n \times w_n = \Sigma\, r_i \times w_i$$

Now these ratings and weights of the sub characteristics such as maturity, fault tolerance, recoverability, and reliability compliance have to be combined by taking the weighted average to get the exact fuzzy rating of reliability. This calculation is based on the formula: -

where i belongs to the set to the set {maturity, fault tolerance, recoverability, and reliability compliance}.

Here r_1 is rating by user 1, r_2 is rating by user2.Similarly w_1, w_2 are weights by different user.

Table 6.1.4 Fuzzy Ratings (calculated) of the Sub Characteristics Belonging to the Reliability Characteristic

Sub Characteristic (Reliability)	Rating	Metrics	Average Rating	Average Weight
Maturity	(0.0,0.1,0.3)	No. of Version Released	(0.1,0.3,0.5)	NA
Fault Tolerance	(0.75,1.0,1.0)	Exception andling	(0.3,0.5,0.7)	(0.3,0.5,0.7)
	(0.50,0.75,1.0)	Percentage of functionalities successfully met	(0.3,0.5,0.7)	(0.0,0.20,0.45)
Recoverability	(0.8,0.9,1.0)	Availability of Data backup	(0.1,0.3,0.5)	NA
Reliability Complaince	(0.6,0.8,0.9)	Adherence to ReliabilityCompliance	(0.7,0.9,1.0)	NA

VIII. CONCLUSION

In this paper we have measured the reliability of Library application using fuzzy logic. By our calculation we can say that the Library software which is developed using Aspect oriented Approach is having high level fault tolerance, good recoverability and high level reliability compliance standard. In future we can also compare these results with the same characteristics of any non aspect oriented application and we can also measure the different quality factors.

REREENCES

1. ISO/IEC 9126-1:2001, "Software Engineering-Product Quality—Part 1: Quality Model", Int'l Organization for Standardization, 2001, Available at www.iso.org
2. R. Laddad, "Aspect Oriemted Programming will improve Quality",2003,published by IEEE Computer Society 0740-745
3. B. W. Boehm, J. R. Brown and M. L. Lipow, "Quantitative Evaluation of Software Quality," Proceedings of the 2nd International Conference on Software Engineering, San Francisco, CA, USA, October,1976, pp.592-605.
4. J. A. McCall, P. K. Richards, and G. F. Walters, Factors in Software Quality, 1977, Vol.I, II, and III, US Rome Air Development Center Reports - NTIS AD/A-049 014, NTIS AD/A-049 015 and NTIS AD/A-049 016, U. S. Department of Commerce.
5. R. G. Dromey, "A model for software product quality," IEEE Transactions on Software Engineering, Vol.21, No.2, February, 1995, pp.146-162.
6. P. R. Srivastava and K. Kumar, "An Approach towards Software Quality Assessment," Communications in Computer and Information Systems Series (CCIS Springer Verlag), Vol.31, No.6, 2009, pp.345-346.
7. P. R. Srivastava, A. P. Singh, K.V. Vageesh, "Assessment of Software Quality: A Fuzzy Multi – Criteria Approach," Evolution of Computationand Optimization Algorithms in Software Engineering:
8. Applications and Techniques, IGI Global USA, 2010, chapter - 11, pp.200-219.
9. A. Sharma, R. Kumar and P.S. Grover, "Estimation of Quality for Software Components - an Empirical
10. Approach," ACM SIGSOFT Software Engineering Notes, Vol.33, No.5, November, 2008, pp.1-10.
11. S.A. Slaughter, D. E. Harter, & M. S. Krishnan, "Evaluating the Cost of Software Quality,"

Communications of the ACM, Vol.41, No.8, August, 1998, pp.67-73.
12. M. Agarwal, & K. Chari, "Software Effort, Quality, and Cycle Time: A Study of CMM Level 5 Projects," IEEE Transactions on Software Engineering, Vol.33, No.3, March, 2007, pp.145-156.
13. O. Maryoly, M.A. Perez and T. Rojas, "Construction of a Systemic Quality Model for Evaluating Software Product," Software Quality Journal, Vol.11, No.3, July, 2003, pp.219-242.
14. O. Lamouchi, A.R. Cherif, and N. Lévy, "A framework based measurements for evaluating an IS quality," Proceedings of the fifth on Asia-Pacific conference on conceptual modelling, Wollongong, NSW, Australia, January, 2008, pp.39-47.
15. Y.Kanellopoulos, P.Antonellis, D. Antoniou, C.Makris, E.Theodoridis, C. Tjortjis and N.Tsirakis, "Code Quality Evaluation Methodology Using The Iso/Iec 9126 Standard," International Journal of Software Engineering & Applications (IJSEA), Vol.1, No.3, July, 2010, pp.17-36.
16. I.Heitlager, T.Kuipers, J.Visser, "A Practical Model for Measuring Maintainability - a preliminary report," 6th International Conference on Quality of Information and Communications Technology (QUATIC), September, 2007, pp.30-39.
17. R. Fitzpatrick and C. Higgins, "Usable Software and its Attributes:A synthesis of Software Quality European Community Law and Human-Computer Interaction", Proceedings of the HCI'98 Conference, Springer, London, United Kingdom. 1998, pp.1-19.
18. IEEE Standard Glossary of Software Engineering terminology, IEEE Std 610.12-1990.
19. Markus Voelter, voelter at acm dot org ,"Aspect oriented Programming in Java".
20. Jiantao Pan, "Software Reliability" Carnegie Mellon University, 18-849b ,Dependable Embedded Systems Spring 1999