VOL. 9, NO. 9, SEPTEMBER 2014 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

AN ESSENCE OF SOFTWARE MAINTENANCE PREDICTION USING THE FUZZY MODEL FOR ASPECT ORIENTED SOFTWARE

Pradeep Kumar Singh[1], Om Prakash Sangwan[2] and Abhishek Srivastava[3]
[1]Department of CSE, ASET, AMITY University, Noida, India
[2]School of ICT, Gautam Buddha University, Gr. Noida, India
[3]Department of IT, ASET, AMITY University, Noida, India
E-Mail: pradeep_84cs@yahoo.com

ABSTRACT

Software maintenance is generally used to refer the changes that are made to software after its initial release, installation and operation. In several research it has proven that maintenance involve more than 40 percent of the total cost of the software. External quality factors assessments were always in light from the beginning of the software engineering research and related to internal quality attributes. Several research papers used the internal attributes to derive the external attributes and their relationship have been discussed and validated in several quality models related research papers. This paper considered the major factors that affect software maintenance for Aspect Oriented Software's and divide them into four categories: Separation of Concern, Cohesion, Coupling and Size. Based on the identified factors, a fuzzy model to predict the software maintenance have been proposed and validated for aspect oriented software. Automated software maintainability examination to guide software related decision's was always in great demand and has been applied from procedural, object oriented to component based software engineering. In this paper a model to predict the maintainability has been proposed and validated using the fuzzy logic for automation of maintainability prediction for AO software.

Keywords: software maintenance, aspect oriented software (AOS), cohesion, coupling, line of code (LOC), separation of concerns (SOC), fuzzy logic, maintainability, aspect oriented programming (AOP).

1. INTRODUCTION

Software maintenance and maintainability is defined in several research studies, but they are impartially consistent in purview and objective. As per the IEEE standard definitions maintenance is "the process of changing a software system or component after delivery to correct bugs, enhance efficiency or other factors, or suitable to a new environment" [1]. It can redefined as ''the intention with which a software system or component can be changed to correct faults, improve efficiency or other attributes, or adapt to a changed environment''.

Based on the fundamental principles of software engineering, software maintenance process can be divided into three areas: **(i) Corrective Maintenance:** maintenance carried out to fix faults in hardware or software **(ii) Adaptive Maintenance:** maintenance carried to make a computer program usable in a new environment **(iii) Perfective Maintenance:** maintenance carried to improve the efficiency, maintainability, or other attributes of a computer program.

Software maintainability has become one of the most important issues of the software industry from more than two decades. In various books and research papers on software engineering several well known software professionals such as Fred Brooks, R.S. Pressman, Ibrahim *et al.*, K.K. Aggarwal and Yogesh Singh mentioned that, maintainability involved the maximum cost i.e. 40 to 60 percent of the software cost [2, 3, 4, 5, 6].

The software development advancement has upgraded the ability of software professionals to achieve clear SOC, or "the potential to locate, encapsulate, and modify only the parts of software that are appropriate to a particular objective, goal, or purpose" [7]. Problem of synchronization as described in [8, 9] and logging among object oriented software's have been over come through the implementation of separation of concerns in Aspect Oriented Programming (AOP).

Kiczales *et al.* discussed the concept of aspect oriented programming in early nineteenth century [10]. Prime objective of AOP is the development of the code that is easier to understand and evolve the principle of SOC. AOP is a new methodology for separation of crosscutting concerns into single units called aspects. An aspect is a modular unit of crosscutting implementation. It encapsulates concerns that affect multiple classes into reusable modules. There are various examples of aspects in AOP i.e. synchronization, logging, security etc. In AOP, each aspect can be conveyed in a separate and natural form, and automatically merged together into a final executable form by an aspect weaver. As an outcome, each aspect can lead to the implementation of a number of modules, or objects, increasing reusability of the source code.

AOP languages have three essential elements for separating crosscutting concerns: a join point model, identifying join points, and implementation at join points [11, 12]. Aspect J is AO extension to Java and become one of the most dominant languages taken for research purpose among researcher of aspect oriented software development community, introduced by Xerox Palo Alto Research Center, which permits plug-and-play implementations of crosscutting in Java.

In several research papers, number of soft computing based techniques have been proposed to estimate the external quality factors (such as reusability, maintainability, reliability), test cases optimization and fault prediction for software being developed using the modular and object oriented programming languages [13,

14, 15, 16, 17]. There are number of studies in which automatic computation of maintainability has been proposed and validated for procedural to object oriented and component based software's [18]. In this paper efforts have put to estimate the maintainability for aspect oriented software using fuzzy logic approach.

In this paper introduction is followed by Section 2 describes the maintainability for aspect oriented software and the factors affecting the maintainability. Section 3 is about the proposed fuzzy model for aspect oriented software's with the experimental design and results are shown in Section 4, Section 5 respectively followed by conclusion.

2. MAINTAINABILITY IN AOS

In this section, firstly the research work related to aspect oriented software maintenance prediction is considered and later, the factors affecting the software maintenance have been identified for aspect oriented software's.

2.1. Related work on the software maintenance for aspect oriented software's

Initially some of the studies in context to AOSD have been carried out to investigate the qualitative and quantitative assessment [19, 20, 21, 22]. Hannemann *et al.* estimate Java and AspectJ implementations of the GoF design patterns in context of weakly defined evaluating criteria, such as pluggability and composability [21].

Sant'Anna *et al.* presents an assessment framework in terms of reusability and maintainability for AO software's, which comprises of two units: a metrics suite and a quality model for AOP [23]. The suggested framework has been measured in the context of two empirical studies. Metrics were combined according to the attributes they calculate: (i) SoC, (ii) Coupling, (iii) Cohesion and (iv) Size. They have pointed that calculating the structural design properties of software artifacts, such as coupling, cohesion, and SOC, is an effective approach towards early assessment of AO software quality.

Li *et al*. considered a case study to examine whether AOP can assist to build an easy-to-modify COTS-based system [24]. This study measures the modifications when adding and eliminating components deployed using Object-Oriented Programming (OOP) and AOP for a COTS system. It has been identified that the integrating COTS components using AOP may assist to improve the changeability of the COTS-based system if the cross-cutting concerns in the source code are homogeneous. Extracting heterogeneous or relatively homogeneous cross-cutting concerns in glue-code as aspects does not contribute remarkable advantages.

Burrows *et al.* compute the efficacy of coupling metrics as benchmark of fault-proneness in AO systems [25]. Their results show that Base-Aspect Coupling (BAC) and Crosscutting Degree of an Aspect (CDA) are two metrics that signify the strongest correlation with faults.

Shen *et al.* derived a fine-grained coupling metrics suite for AO systems, to estimate software changes during system evolution [26]. They also proposed a correlation model in context of intermediate processes, for better computing the relation between coupling metrics and maintainability.

Eaddy *et al.* established a correlation among concern metrics and an external quality indicator i.e. defects with few internal and external threats to analysis [27]. Main objective of this study was to investigate the effect of crosscutting concerns on quality in terms of defects. They also mentioned that more empirical validation is required to draw the general conclusion between the defects and crosscutting concerns.

Walker *et al.* consider the patch data of the Mozilla project for a decade to consider whether crosscutting concerns exist therein and whether the facts of problems arising from them can be find [28]. But this analysis reveal the general assumption of "crosscutting concern lead to defects".

Garcia *et al.* proposed a quantitative analysis of AO and OO solutions for the 23 Gang-of-Four patterns [29]. They have considered rigorous software engineering attributes as the evaluation parameter. They concluded that most AO solutions improve separation of pattern related concerns, although only 4 AO implementations have revealed consequential reuse.

Greenwood *et al.* proposal included an examination of the application in context of modularity, change propagation, concern interaction, identification of ripple-effects and stick to well-defined design principles for AO software's [30].

Kulesza *et al.* advocate a quantitative analysis to reveal the positive and negative impact of AOP on maintenance tasks of a Web information system [31]. This analysis considered a systematic comparison among the OO and the AO versions of the same application in order to estimate to what extent each solution deliver maintainable software decompositions. Authors analyzed that the AO design has shown better stability and reusability through the modifications, as it has evolved in fewer lines of code, enhanced SOC, weaker coupling, and lower intra-component complexity.

Kumar *et al.* estimate the correlation among changeability and WOM metric [32]. They analyzed the WOM can be considered as an indicator of maintainability but it is a weak indicator. Change impact is smaller in AO compared to OO systems. Maintenance effort was computed in context of the number of modules modified.

Based on the literature review, several metric suites for AOP programs for measuring the SOC, Cohesion, Coupling and Size is mentioned. These four factors have been identified that affect the maintainability for aspect oriented software's and also used by Shen *et al.* and Sant'Anna *et al.* [33, 34] respectively in their studies. Previously, we have proposed maintainability model for AOS based on the AO software metrics [33]. We have taken into account these factors for the proposed fuzzy system as input to predict the maintainability.

2.2. Factors affecting software maintainability for aspect oriented software

There are several factors which directly and indirectly affects the software maintainability. In case of aspect oriented systems maintainability factors are slightly different as compare to object oriented and modular approaches. Aspect oriented software's are basically extension to object oriented systems with some additional features such as aspects, join points, point cut, advices and identification of concerns. It has been identified in several studies that SOC improves design modularity and proven to be better solutions in case of logging, exceptions handling, and tracing [23]. Based on the related work mainly four factors have been identified which affect the maintainability for aspect oriented software i.e. (i) SOC (ii) Coupling (iii) Cohesion (iv) Size.

There are number of research papers and studies which have considered the design metrics as quality indicator for software [28, 34]. Our study is taken on account the design metrics to calculate the external quality factor, i.e. there is always a relationship between the software metrics and quality attributes such as maintainability, reusability etc.

Two sets of metrics were discussed in most of the research papers for aspect oriented software, Some of them are based on object oriented metrics and others by considering unique features of aspect oriented systems as following: (i) Metrics taken from the OO software's published by Chidamber and Kerner [35], Zakaria and Hosny [36], Santanna *et al.* [23], Ceccato *et al.* [37] i.e. line of Code (LOC), weighted operation in module (WOM), depth of inheritance (DIT) and number of children (NOC) etc. (ii) Metrics precise for AO software i.e. crosscutting degree of aspect (CDA) and coupling on advice execution (CAE) published by Ceccato and Tonella [37].

In our proposed work, studies discussed in literature review are considered as the benchmark to automate the maintainability prediction for aspect oriented softwares. We have identified mainly four factors that affect the maintainability for aspect oriented software: (a) Separation of Concern (b) Coupling (c) Cohesion (d) Size.

(a) Separation of concerns (SOC): SOC introduce identification, encapsulation and manipulation of those parts of software that are appropriate to a particular objective or cause [38]. Concerns may be defined in many forms and at different stages of abstraction: (i) Features from a feature list (ii) Requirements from a SRS (iii) Design patterns and design elements from a UML design document (iv) Low-level programming concerns such as language used, coding pattern, programming idioms, code reuse, information hiding, and algorithms. There are numerous metrics identified for SOC such as Concern Diffusion over Components (CDC), Concern Diffusion over Operations(CDO), Concern Diffusion over LOC (CDLOC), Cross cutting degree of an aspect (CDA), coupling on advice execution (CAE) [23, 37].

(b) Coupling: It is an indication of the degree of interconnections among the components or modules in considered system. According to fundamental principles of software engineering low coupling is desirable for high quality of design and best maintainable systems. Coupling between Components *(*CBC), Depth of Inheritance (DIT) and Coupling on Intercepted Module (CIM), CMC (Coupling on Method Calls), CFA (Coupling on Field Access) are the metrics for coupling measurement for AOSD [23, 37].

(c) Cohesion: It is defined as the degree to which elements of a module belong to each other. Lack of cohesion in operations i.e. LCOO, Aspect Cohesion (ACOH) [23, 29].

(d) Size: Size metrics measures the length of a software system's and static source code. There are number of size metrics derived from CKs metric suites for AOSD such as line of code (LOCC), weighted operation in module (WOM) or weighted operation per components (WOC), number of children (NOC), number of attributes (NOA) and vocabulary size (VS) [23, 35, 37].

As shown in Figure-1, proposed model for AO software maintainability, considers the maintainability as the integrated measure of the four factors: (a) SOC (b) Coupling (c) Cohesion (d) Size.

3. PROPOSED FUZZY MODEL FOR ASPECT ORIENTED SOFTWARE'S

In 1965, Zadeh *et al*. proposed fuzzy logic based fuzzy set theory [39, 40]. Fuzzy logic has been implemented at many places like mechanical engineering, production and many mores areas, as well as for software engineering problems. It emerges as an interesting area of research to calculate the precision values that human was managing from long time and managing imprecise information. Fuzzy logic is always beneficial in case of mapping and input to output data or values. This paper proposed a fuzzy model to predict the maintainability of AO software's based on four inputs, namely (i) SOC (ii) Coupling (iii) Cohesion (iv) Size as shown in Figure-1 by assuming the equal weight to each input.

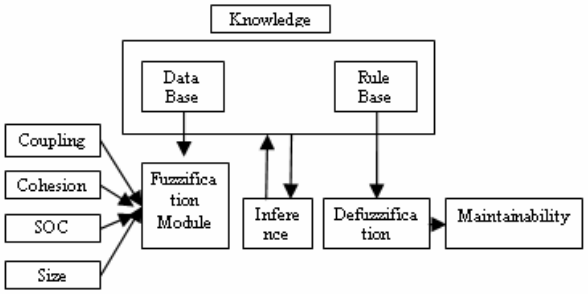

Figure-1. Fuzzy model for aspect oriented software maintainability.

In this proposed model all four factors are considered as inputs to provide a crisp value of maintainability for aspect oriented software using rule base. Fuzzy Inference System (FIS) is the technique to compose the mapping from a given input values to an output values using fuzzy logic. Mamdani's fuzzy inference technique is used in the proposed scheme as shown in Figure-2.

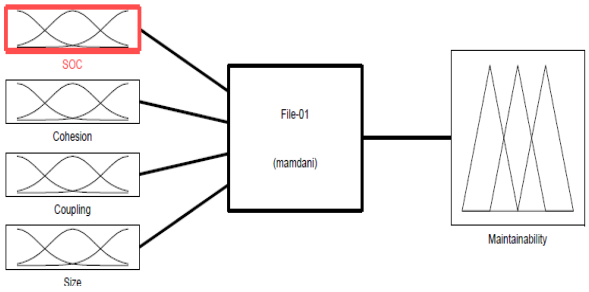

Figure-2. Fuzzy inference system: Maintainability model.

The linguistic variables for the proposed models have been identified in the following manner: (i) Linguistic variables for all inputs: {low, medium, high} and Linguistic variables for the output: {very low, low, medium, high, very high}. Defuzzification is require for fuzzy sets of each output variable after applying the fuzzification. The input to the defuzzification is a fuzzy set (the aggregate output fuzzy set) and the outcome is single numeric value. Centroid method is used for defuzzification, which calculates the center of area under curve [41].

4. EXPERIMENTAL DESIGN

In order to fuzzify the inputs, we have taken the cohesion, coupling, LOC and SOC. Input variables, SOC, cohesion, coupling and size of the source code for AO software's is classified among three categories i.e. low, medium and high as shown in Figure-3, for SOC only. Based on the suitability of membership functions "trapmf" is used as the membership function for the input and output.

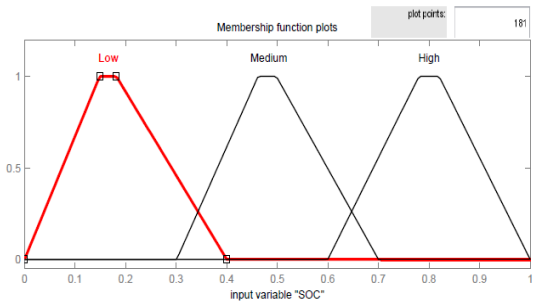

Figure-3. Fuzzification of input Variable SOC.

The output variable maintainability for AO software is classified as very high, high, medium, low and very low as shown in Figure-4.

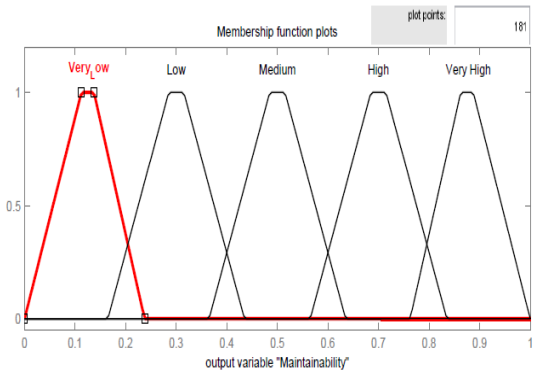

Figure-4. Fuzzification of output variable - maintainability for aspect oriented software.

4.1. Rule base and evaluation process

After fuzzyfying the input data, processing is carried out in fuzzy domain. The fuzzy model integrates the effect of Cohesion, Coupling, SOC and Size into a single measurable parameter, termed as software maintainability for aspect oriented software, based on the following knowledge/fuzzy rule base. Depend upon the usages, however the knowledge base can further be further refined to more ranges (fuzzy sets) for the input variables. All inputs and outputs have been fuzzified for proposed systems as shown in Figures 3 and 4, respectively. All possible combinations of inputs were considered which leads to 3^4 i.e. 81 possible sets for input. The maintainability in case of all 81 combinations is identified as very low, low, medium, high and very high with the help expert opinion from the software engineering domain. Total 81 rules for the fuzzy model have been derived and some of them are shown below:

1. If (SOC is Low) and (Cohesion is Low) and (Coupling is Low) and (Size is Low) then (Maintainability is Medium).
2. If (SOC is Low) and (Cohesion is Low) and (Coupling is Low) and (Size is Medium) then (Maintainability is High)
3. :
27. If (SOC is Low) and (Cohesion is High) and (Coupling is High) and (Size is High) then (Maintainability is High)
:
81. If (SOC is High) and (Cohesion is High) and (Coupling is High) and (Size is High) then (Maintainability is Medium)

All 81 rules are inserted and rule base is created in MATLAB Fuzzy Toolbox. All rules are fired based on set of inputs. There are three commonly used inference mechanism-mamdani style, larsen style, and sugeno style, we used ''mamdani'' style inference for this type of models. Using the rule viewer, output i.e. software maintainability is observed for a particular set of inputs using the Fuzzy Tool box as shown in Figure-5.

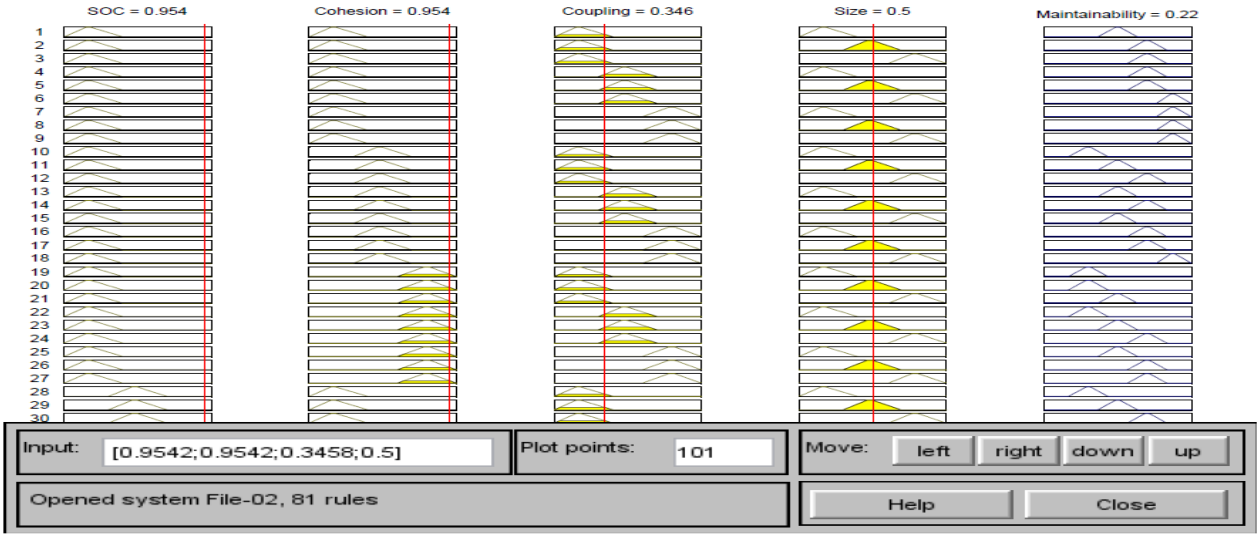

Figure-5. Rule viewer for the maintainability model.

5. EXPERIMENTAL RESULTS

This proposed system considers all the input sets into the range of [0-1], in case of different values same can be normalized to the specific range. Suppose we have the following crisp value inputs to the fuzzy model: SOC=0.1742, Cohesion=0.1769, Coupling=0.8258 and Size=0.7923. These inputs are fed to the fuzzification module and after fuzzification of the given values we find that Maintainability = Very High belongs to fuzzy set with membership grade 0.879. Here low value is desirable for better quality software and require less maintenance efforts. With all set of possible inputs values we find 81 rules, some of them are also shown in Table-1.

Table-1. Software Maintainability for AOSD calculation for a given sets of input.

Rules	SOC	Cohesion	Coupling	Size	Maintenance Level	Membership grade of software maintainability
9	Low	Low	High	High	Very High	[0.1742;0.1769;0.8258;0.7923]=[0.879]
77	High	High	Medium	Medium	Low	[0.9542;0.9542;0.3458;0.5]=[0.225]

5.1. Defuzzification

After obtaining the fuzzified output as mentioned above, we defuzzify them to get crisp value of the output variable maintainability for AO software [6, 41]. Transformation of the output from fuzzy domain to crisp domain is called defuzzification. In our proposed model we obtain this by considering the Centroid method of the aggregated output of rule number 77 as shown in Table-1 and same can be derived for rest of the rules to validate the result.

$X^* = \sum A\bar{x} / \sum A$ ……Equation no.1 [6]

$X^* = \sum A\bar{x} / \sum A = \sum A_1 \bar{x} + A_2 \bar{x} + A_3 \bar{x} / \sum A_1 + A_2 + A_3$

$X^* = 0.000085 + 0.027676 + 0.004551 / 0.00925 + 0.1258 + 0.011$

$X^* = 0.0323125 / 0.14615$

$X^* = 0.2210913$

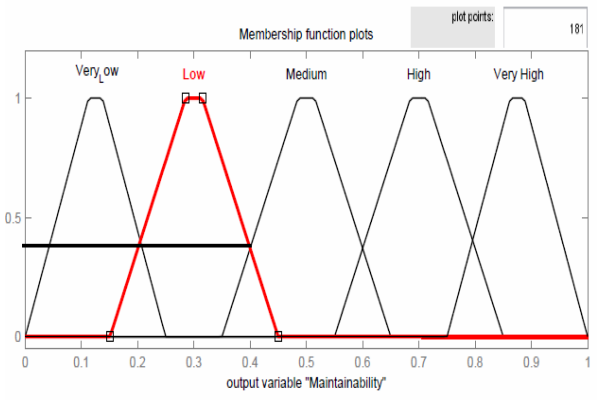

Figure-6. Defuzzification of output variable - maintainability.

The aggregated output has been divided into three parts for better understanding. The aggregated fuzzy set of A_1, A_2, A_3 area is computed as shown in Figure-6 using the centroid equation. X^* is the centroid, computed using the aggregated fuzzy set and $\bar{x}$ shows the corresponding

centroid. The effects of these rules were also observed by simulating the model using the Fuzzy Logic Tool Box of MATLAB. The maintainability for the above mentioned inputs comes out to be 0.225 which is very close to the calculated value using the centroid method i.e. 0.2210. Overall response of the system is shown in Figures 7, 8 and 9. The proposed fuzzy model is shown the suitability for identification of software maintenance level for the aspect oriented software's.

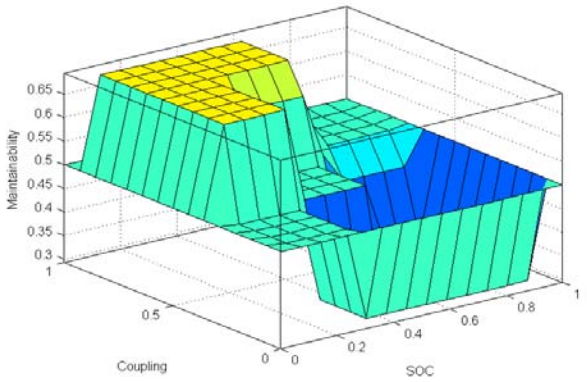

Figure-7. Surface view with SOC as input on x axis and Coupling on y axis and Maintainability on z axis.

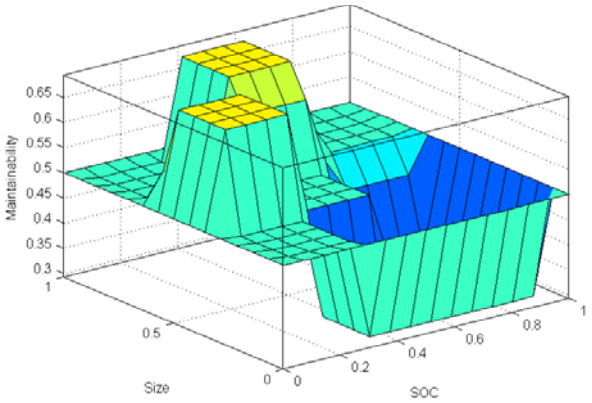

Figure-8. Surface view with SOC as input on x axis and Size on y axis and Maintainability on z axis.

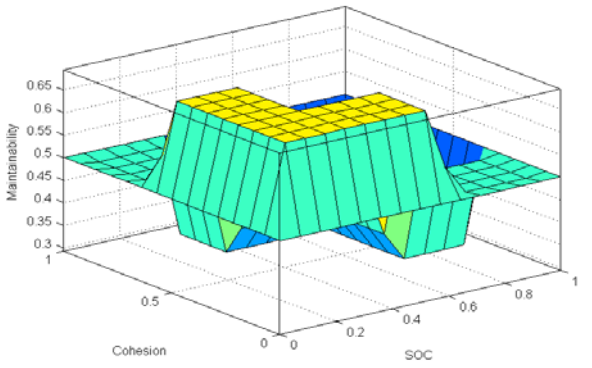

Figure-9. Surface view with SOC as input on x axis and Cohesion on y axis and Maintainability on z axis.

6. CONCLUSIONS

In this paper rule based fuzzy model to predict software maintenance level of the aspect oriented software has been presented. Fist we have identified the factors affecting the software maintainability for aspect oriented software, a relationship between the identified factors and maintainability has been derived. This relationship is used to predict the external quality factor such as maintainability based on input factors i.e. internal quality attributes. The used input factors already discussed and validated using several studies to derive quality models for aspect oriented software [23, 35, 36, 37].

In this paper we automate the process of the measurement of software maintainability using four important factors derived from the design metrics i.e. SOC, Cohesion, Coupling and Size as inputs. These factors (internal attributes) can be measured using software metrics through metrics tools available for the aspect oriented software i.e. AOP metric, Metric 1.3. Based on these input factors, the proposed maintainability measure could be evaluated and normalized. The proposed model will provide automatic prediction of software maintainability for a module as well for the whole software and would help in evaluation of software maintenance level based on the available set of design metrics for AO software. It would encourage software organization to develop less maintainable software. In future sub attributes of maintainability, like testability can be examined using metrics and other attributes for AO software. In [42], testability has been analyzed theoretically for AO software. Further, we are planning to apply the proposed model on the AO software dataset and cross validate using the neural network, support vector machine techniques for maintainability assessment.

REFERENCES

[1] IEEE Std. 610.12-1990. Glossary of Software Engineering Terminology. In Software Engineering Standards Collection, IEEE CS Press, Los Alamitos, Calif., Order No. 1048-06T, 1993.

[2] Aggarwal K. K. and Singh Y. 2001. Software Engineering: Programs, documentation, operating procedures. New Age International Publishers.

[3] Brooks F. P. 1995. The mythical man-month. Essays on Software Engineering. Pearson Education.

[4] Ibrahim S., Idris N.B., Munro M. and Deraman A. 2005. Integrating Software Traceability for Change Impact Analysis. International Arab Journal of Information Technology. 2(4): 301-308.

[5] Pressman R. S. 2005. Software engineering: a practitioner's approach. Vol. 5, McGraw-Hill International Edition. pp. 466-472.

[6] Rajasekaran S. and Pai G.V. 2003. Neural Networks, Fuzzy Logic and Genetic Algorithm: Synthesis and Applications, PHI Learning Pvt. Ltd.

[7] Ossher H. and Tarr P. 2001. Using Multidimensional Separation of Concerns to Reshape Evolving Software. Communications of the ACM. 44(10): 43-50.

[8] Dempsey J. and Cahill V. 1997. Aspects of System Support for Distributed Computing. Proc. *ECOOP* Workshop on Aspect-Oriented Programming, Finland.

[9] Kiczales G. 1996. Aspect-Oriented Programming. ACM Computing Surveys. 28(4): 154.

[10] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J. M. and Irwin J. 1997. Aspect-Oriented Programming. Springer Berlin Heidelberg. pp. 220-242.

[11] Elrad T., Aksits M., Kiczales G., Lieberherr K.J. and Ossher H. 2001. Discussing Aspects of AOP. Communications of the ACM. 44(10): 33-38.

[12] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J. and Griswold G. 2001. Getting Started with Aspect J. Communications of the ACM. 44(10): 59-65.

[13] Dhir R. 2012. Bayesian and Fuzzy Approach to Assess and Predict the Maintainability of Software: A Comparative Study. ISRN Software Engineering.

[14] Dubey S.K., Rana A. and Sharma A. 2012. Usability Evaluation of Object Oriented Software System using Fuzzy Logic Approach. International Journal of Computer Applications. 43(19): 1-6.

[15] Kumar M., Sharma A. and Kumar R. 2011. Towards Multi-Faceted Test Cases Optimization. Journal of Software Engineering and Applications. 4(9): 550-557.

[16] Singh Y., Bhatia P.K. and Sangwan O.P. 2009. Predicting software maintenance using fuzzy model. ACM SIGSOFT Software Engineering Notes. 34(4): 1-6.

[17] Tyagi K. and Sharma A. 2012. A rule-based approach for estimating the reliability of component-based systems. Advances in Engineering Software. 54: 24-29.

[18] Coleman D., Ash D., Lowther B. and Oman P. 1994. Using metrics to evaluate software system Maintainability. Computing Practices. 8(27): 44-49.

[19] Driver C. 2022. Evaluation of Aspect-Oriented Software Development for Distributed Systems. Masters Thesis, University of Doublin.

[20] Garcia A., Silva V., Chavez C. and Lucena C. 2002. Engineering Multi-Agent Systems with Aspects and Patterns. Journal of the Brazilian Computer Society. 1(8): 57-72.

[21] Hannemann J. and Kiczales G. 2002. Design Pattern Implementation in Java and Aspect J. ACM SIGPLAN Notices. 37(11): 161-173.

[22] Kersten M. and Murphy G. 1999. Atlas: A Case Study in Building a Web-based learning environment using aspect-oriented programming. ACM SIGPLAN Notices. 34(10): 340-352.

[23] Sant'Anna C., Garcia A., Chavez C., Lucena C. and VonStaa A. 2003. On the reuse and maintenance of Aspect Oriented software: an assessment framework. Proc. Brazilian Symposium on Software Engineering. pp. 19-34.

[24] Li J., Kvale A.A. and Conradi R. 2006. A Case Study on Improving Changeability of COTS-Based System Using Aspect-Oriented Programming. Journal of Information Science and Engineering. 22(2): 375-390.

[25] Burrows R., Ferrari F.C., Garcia A. and Taiani F. 2010. An empirical evaluation of coupling metrics on aspect-oriented programs. Proc. ICSE Workshop on Emerging Trends in Software Metrics, ACM, Cape Town, South Africa. pp. 53-58.

[26] Shen H., Zhang S. and Zhao J. 2008. An empirical study of maintainability in aspect-oriented system evolution using coupling metrics. Proc. 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering. pp. 233-236.

[27] Eaddy M., Zimmermann T., Sherwood K. D., Garg, V., Murphy G. C., Nagappan N. and Aho A. V. 2008. Do crosscutting concerns cause defects? IEEE Transactions on Software Engineering. 34(4): 497-515.

[28] Walker R. J., Rawal S. and Sillito J. 2012. Do crosscutting concerns cause modularity problems? Proc. 20th International Symposium on the Foundations of Software Engineering, ACM SIGSOFT. p. 49.

[29] Gélinas J.F., Badri M. and Badri L. 2006. A Cohesion Measure for Aspects. Journal of Object Technology. 7(5): 75-95.

[30] Greenwood P., Bartolomei T., Figueiredo E., Dosea M., Garcia A., Cacho N., Sant'Anna C. and Rashid A.

2007. On the impact of aspectual decompositions on design stability: An empirical study. Proc. ECOOP 2007-Object-Oriented Programming, Springer Berlin Heidelberg. pp. 176-200.

[31] Kulesza U., Sant'Anna C., Garcia A., Coelho R., VonStaa A. and Lucena C. 2006. Quantifying the effects of aspect-oriented programming: A maintenance study. Proc. 22nd IEEE International Conference on Software Maintenance. pp. 223-232.

[32] Kumar A., Kumar R. and Grover P.S. 2007. An evaluation of maintainability of aspect-oriented systems: a practical approach. International Journal of Computer Science and Security. 1(2): 1-9.

[33] Singh P.K. and Sangwan O.P. 2013. Aspect Oriented Software Metrics Based Maintainability Assessment: Framework and Model. Published in proceedings of Confluence-2013, The Next Generation Information Technology Submit, 26th - 27th September, Amity University, Noida, India, Available on IET Digital Library and IEEE Xplore.

[34] Basili V.R., Briand L.C. and Melo W.L. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering. 22(10): 751-761.

[35] Chidamber S. R. and Kemerer C.F. 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 20(6): 476-493.

[36] Zakaria A. A. and Hosny H. 2003. Metrics for aspect-oriented software design. Proc. 3rd International Workshop on Aspect-Oriented Modeling, Boston, USA. Vol. 3.

[37] Ceccato M. and Tonella P. 2004. Measuring the effects of software aspectization. Proc. 1st Workshop on Aspect Reverse Engineering, Delft, Netherlands. Vol. 12.

[38] Tarr P., Ossher H., Harrison W. and Sutton S. M. 1999. N Degrees of Separation: Multi-Dimensional Separation of Concerns. Proc. of the 21st International Conference on Software Engineering. pp. 107-119.

[39] 2014. Fuzzy Logic, Stanford Encyclopedia of Philosophy, Stanford University. http://plato.stanford.edu/entries/logic-fuzzy/Last Accessed on 30-04. 2014.

[40] Zadeh L.A. 1965. Fuzzy sets. Information and Control. 8(3): 338-353.

[41] 2002. Fuzzy Logic Toolbox, User's Guide version 2. The Math Works Inc.42

[42] Singh P.K., Sangwan O.P., Pratap A. and Singh A.P. 2014. An Analysis on Software Testability and Security in Context of Object and Aspect Oriented Software Development. International Journal of Information Security and Cybercrime. 3(1): 17-28.