2011 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
VEHICLE ELECTRONICS AND ARCHITECTURE (VEA) MINI-SYMPOSIUM
AUGUST 9-11 DEARBORN, MICHIGAN

SOFTWARE RELIABILITY PREDICTION FOR ARMY VEHICLE

Macam S. Dattathreya
TARDEC
Warren, MI

Harpreet Singh
Wayne State University
Detroit, MI

ABSTRACT

Army vehicles are complex due to various on-board mission critical communication devices. The Army cannot afford unreliable software to interact between the devices. The Army vehicle software's reliability is influenced by multiple factors during or prior to its development. Using complex statistical and mathematical models, software's reliability can be predicted, but it is dependent on the accuracy and context of the historical software failure data. The cost of developing such complex models does not yield a good return on investment. The data collection process to use these models is very difficult and time consuming. In this paper, we propose reliability metrics based on the current software development and design process factors. We also propose a fuzzy logic based software reliability prediction algorithm using the proposed reliability metrics.

INTRODUCTION

Software reliability is one of the growing challenges for the on-board devices in an Army's combat vehicle (CV). During combat missions, soldiers perform complex operations using the devices. Each device has its own software and we refer to it as *Army vehicle software (AVS)*. Combat command and control, and vehicle management are some examples of AVS. An AVS performs its intended functions when needed. This allows defending against the enemy threats and protecting the soldiers. Therefore, AVS reliability is a key factor for successful missions.

An AVS is considered reliable when it exhibits all its designed features with no defects at all times when it is operated in multiple predefined environments and conditions. We use the term *AVS reliability* to refer to a probability of defect-free condition for a given AVS. As the probability of defects increases, the AVS reliability decreases.

Normally, testing reveals software defects and the developers correct them before it is used in practice. This process assures that the software is reliable when it is used in a combat mission. However, fixing the defects after they are found during testing is costly; it is always better to apply efficient techniques during early development phases and avoid future defects in the software. For early development phases, researchers have proposed several complex mathematical and statistical techniques to predict the probability of future defects in the software. These techniques require skilled resources to use them. The Army tends to react to a dynamic situation and they need easy to understand software reliability prediction techniques.

Multiple factors during a software development process influences AVS reliability. The Army pays considerable attention to predict AVS reliability using easy to understand techniques. The AVS reliability measures software quality. Ensuring AVS reliability is absolutely essential [1].

Software architects develop IT architecture documents (ITAD) during early design phases of an AVS development project. ITAD reflect how the software will be implemented and tested. Therefore, ITAD captures necessary details for a successful software development project. We can investigate ITAD and formulate reliability metrics and then use them to develop AVS reliability prediction techniques.

In this paper, we propose the following:

1. Deriving AVS reliability metrics from ITAD.
2. AVS reliability prediction algorithm using fuzzy logic and the derived AVS reliability metrics.

Our research is in its initial stages. In this paper, we introduce our approach and an initial solution to predict AVS reliability. We describe the proposed solution in this paper using an example dataset that we developed to introduce the concept. We are in the process of validating our work for a practical AVS development project.

RELATED WORK

Most of the architecture related predictions surveyed in [2] are dependent on the software structure or the type of modeling languages used.

Mohanta et al. provide an approach in [3] to derive design metrics and use them in a Bayesian Belief Network to predict software reliability. In this approach, the prediction accuracy depends on the accuracy of how well the design metrics are derived. In many cases, it is overkill and restrictive.

Wang et al. recommend a state model framework in [4] using software architecture as a basis for reliability prediction. In this approach, designers create a Unified Modeling Language (UML) state model for individual components of a software program and then solve a Markov Chain process problem to predict reliability. Although this is a good approach, dissecting the UML architecture to identify its structure to evaluate future problems is cumbersome and error prone. It also has a dependency on the type of architecture style used to document software architecture.

Goseva-Popstojanova et al. propose a prediction method [5] using component interaction within software architectures. This method is based on analytical approaches using control graphs. Possible execution paths within software programs are also used to predict reliability. Identifying and analyzing the architecture construct is time consuming and complex. It also depends on the style used to develop software architectures.

Gokhale et al. provide a framework for predicting software reliability [6] using composite models built from individual architecture components and their failure behavior. A prediction using this method is accurate only when one understands all the component interfaces and their failure behaviors. The model used in this approach is too analytical and it is very hard to collect data and process it to predict reliability.

Nagappan et al. provide a list of metrics [7] for early software tests and reliability early warning. But most of the metrics are derived after the software and test cases are written. So, the early prediction here only serves to predict reliability before the software is fielded and used in practice.

Smidts et al. recommend a fault tree based Bayesian quantification framework [8] to evaluate software reliability using software functional architectures which represent both functional and non-functional requirements. This framework concentrates on studying or experimenting with software which is already developed. This cannot be used for early predictions.

Kong et al. provide a cause and effect Graphing Analysis [9] approach for early software reliability prediction by identifying errors in the software requirements specification (SRS) document. Every SRS document is written differently, and this method does not provide any techniques to identify defects in terms of incompleteness, inconsistencies, ambiguities, and redundancies in each software requirement.

Yacoub et al. propose a scenario based reliability analysis [10] technique for component-based software using the path-based approach introduced in [5]. This approach models component interactions using a component-dependency graph. This technique is too analytical and the complexity grows as the number of components within the software increases. Furthermore the data collection process is cumbersome.

AVS RELIABILITY METRICS

ITAD captures the information required to transform user requirements into tangible implementation. Some ITAD are pure text documents and some are built using modeling languages such as UML. ITAD describes requirements realization without providing low level implementation details. The developers use ITAD for developing specific solution designs.

At any point of time during a software development project, any person with ordinary computer skill can take a snapshot of information from ITAD and derive *fault handling (F)* mechanisms, *data handling (D)*, *interoperability (I)*, and *configurability (C)* details documented in the ITAD.

AVS project managers can use the derived 'D', 'I', 'C', and 'F' values as metrics at any phase of an AVS development project and then use them to predict AVS reliability. Project managers can rectify issues and predict reliability again to determine if the reliability has improved.

We describe how to derive AVS reliability metrics using ITAD information in next few sections.

Data handling (D)

We can quantify *data handling* to represent the understanding of the required data and its characteristics, and the test cases planned for them. Using this, we can determine the impact of 'D' on AVS reliability. As the 'D' value calculated from (1) increases, the AVS reliability decreases.

$$D = 3 - \left(\frac{D_2 + D_3}{D_1} + \frac{T_1}{D_1} \right) \quad (1)$$

Where,

D_1 = required number of distinct data elements.

D_2 = number of distinct data elements captured with necessary details.

D_3 = number of distinct data elements that have captured required data characteristics.

T_1 = total test cases for all the data elements. It is calculated using (2).

$$T_1 = \sum_{I=1}^{D_1} \frac{\sum_{i=1}^{N_c} T_{1Ii}}{N_c} \quad (2)$$

Where, T_{1I} = number of test cases that are planned for testing all data characteristics per data element. I.e., every data element must have at least one test case planned for every data characteristic. N_c = total number of data characteristics. I = I^{th} data element in D_1, and i= i^{th} data characteristic in N_c. For example, assume N_c = 3 and D_1 = 3, Then the T_{1Ii} can be represented as: T111, T112, T113, T121, T122, T123, T131, T132, and T133. For the 1^{st} data element, if the 1^{st} data characteristic has a test case planned, then T_{111} takes the value of 1 else it takes the value of 0. Similarly all other T_{1xx} can be calculated.

The most important data characteristics are: processing requirements, data format, the security constraints, data size limitations, and the data storage requirements. Understanding the data elements during early phases of an AVS development improves the quality of AVS design, and it reduces future defects. If fewer data elements are known and fewer test cases are planned than the actual required data elements, then 'D' poses a major negative impact on achieving AVS reliability. Each required data element with its details has to be known and should have at least one test case planned to test each of the data characteristics. This avoids any future surprises.

Interoperability (I)

Based on the IEEE definition, interoperability is the capability of an AVS to exchange and use data in an operating environment within predefined access restrictions. An interoperable AVS is expected to understand all the available interfaces when exchanging data with other AVSs. This capability enables an AVS to produce reliable operations.

We can quantify *interoperability* to represent the understanding of the required inputs and outputs, and the test cases planned for them. Using this, we can determine the impact of 'I' on AVS reliability. As the 'I' value calculated from (3) increases, the AVS reliability decreases.

$$I = 8 - \left(\frac{I_2}{I_1} + \frac{T_2 + T_4 + T_8}{I_1} + \frac{O_2}{O_1} + \frac{T_3 + T_6 + T_{10}}{O_1} \right) \quad (3)$$

Where,

I_1 = required number of distinct inputs.

I_2 = number of distinct inputs captured with necessary details.

O_1 = required number of distinct outputs.

O_2 = number of distinct outputs captured with necessary details.

T_2 = total test cases planned for testing all the details for all the inputs. It is calculated using (4).

$$T_2 = \sum_{I=1}^{I_1} \frac{\sum_{i=1}^{N_i} T_{2Ii}}{N_i} \quad (4)$$

T_3 = total test cases planned for testing all the details for all the outputs. It is calculated using (5).

$$T_3 = \sum_{I=1}^{O_1} \frac{\sum_{i=1}^{N_o} T_{3Ii}}{N_o} \quad (5)$$

Where, T_{2I} and T_{3I} are the number of test cases planned for testing all the details per input and output, respectively. I.e., every input and output must have at least one test case planned for every detail. N_i and N_o are the total number of input and output details, respectively. I = I^{th} input in I_1 and I^{th} output in O_1, respectively. i= i^{th} detail in N_i and i^{th} detail in N_o, respectively. For the I^{th} input, if the 1^{st} detail has a test case planned, then T_{211} takes the value of 1 else it takes the value of 0. The same rule applies toT_{311} also. Similarly all the other T_{2xx} and T_{3xx} can be calculated.

T_4 = number of distinct inputs planned for testing its event logging.

T_6 = number of distinct outputs planned for testing its event logging.

T_8 = number of distinct inputs planned for testing its fault handling.

T_{10} = number of distinct outputs planned for testing its fault handling.

An AVS may have one or many input sources and one or many output destinations. The complete knowledge of the following reduces defect rates and increases AVS reliability:

1. From where are the inputs received and to where will the outputs be delivered?
2. The security constraints between the input sources → AVS and AVS → output destinations.
3. The transport mechanism for all inputs and output deliveries e.g., frequency, mode of initiation (automatic push or pull).

Understanding the inputs and outputs early in the design phase improves the quality of AVS design and development, and it reduces future defects. If fewer inputs or outputs details are known and fewer test cases are planned than the

actual required inputs or outputs, then the 'I' poses a major negative impact on achieving AVS reliability. Each required input and output with its details has to be known and should have at least one test case planned to test each of the details. This avoids any future surprises.

Configurability (C)

We can quantify c*onfigurability* to represent the configurability planned for each data elements, inputs, and outputs, and the test cases planned for them. Using this, we can determine the impact of 'C' on AVS reliability. As the 'C' value calculated from (6) increases, the AVS reliability decreases.

$$C = 8 - \left(\frac{C_1 + C_3}{I_1} + \frac{T_5 + T_9}{I_1} + \frac{C_2 + C_4}{O_1} + \frac{T_7 + T_{11}}{O_1} \right) \quad (6)$$

Where,

C_1 and C_2 are the number of distinct inputs and outputs, respectively planned for configurable event logging.

C_3 and C_4 are the number of distinct inputs and outputs, respectively planned for configurable fault handling.

T_5 and T_7 are the number of distinct inputs and outputs, respectively planned for testing its configurable event logging.

T_9 and T_{11} are the number of distinct inputs and outputs, respectively planned for testing its configurable fault handling.

Event logging mechanisms minimize confusion in dealing with operational faults. Configurable event logging enables the users or the maintainers to understand what really went wrong or why a specific scenario executed when something else was expected. When developers or testers exercise this feature, it enables them to understand and fix problems correctly. It enhances the defect removal process and increases AVS reliability.

If configurability is planned for each of the data elements, inputs, and outputs, it enables an AVS to operate in multiple operating environments and conditions. If no configurability is planned, there is no easy way to allow it to work in multiple operating environments without changing the code. Using configurability in an AVS improves the quality of AVS design and development, and it reduces the abrupt failures when it is operated in an unknown environment for which no code exists. If fewer configurability contingencies are planned and fewer test cases are planned than the actual required inputs or outputs or data elements, then 'C' poses a major negative impact on achieving AVS reliability. Each required input, output, and data element with its details has to be planned for configurability and should have at least one test case planned to test each configurable option. This avoids any future surprises and problems.

Fault handling (F)

We can quantify f*ault handling* to represent the fault handling mechanisms planned for each data elements, input, and output, and the test cases planned for them. Using this, we can determine the impact of 'F' on AVS reliability. As the 'F' value calculated from (7) increases, the AVS reliability decreases.

$$F = 6 - \left(\frac{E_1 + F_1}{I_1} + \frac{E_2 + F_2}{O_1} + \frac{T_8}{I_1} + \frac{T_{10}}{O_1} \right) \quad (7)$$

Where,

E_1 = number of distinct inputs planned for event logging.

F_1 = number of distinct inputs planned for fault handling.

E_2 = number of distinct outputs planned for event logging.

F_2 = number of distinct outputs planned for fault handling.

T_8 = number of distinct inputs planned for testing its fault handling.

T_{10} = number of distinct outputs planned for testing its fault handling.

Fault handling mechanisms allow users to get graceful notifications during a failure through readable messages instead of ending the program abruptly. Using proper fault handlers indicates problem areas and the reasons for it. This allows users to take corrective actions to increase the reliability of AVS operations e.g., when the user enters invalid data in an AVS, instead of ending the AVS operation abruptly, a fault handling mechanism in an AVS notifies the user to rectify the data and allow re-execution of the operation. This allows successful operation and increases AVS reliability.

If fault handing mechanisms are planned for each of the data elements, inputs, and outputs, it enables developers to understand the details of when and how a fault happens. If no fault handling is planned, there is no easy way to gracefully inform the user of any faults or errors. Using fault handling improves the quality of AVS design and development, and it reduces abrupt failures. If fewer fault handlings are planned and fewer test cases are planned than the actual required inputs or outputs or data elements then the 'F' poses a major negative impact on achieving AVS reliability. Each required input, output, and data element with its details has to be planned for fault handling and they should have at least one test case planned to test each fault handling mechanism.

All the parameters described above are simple to understand and a person with no special skills can gather them very easily from reading ITAD.

In the next section, we describe how the metrics defined in this section can be used to predict AVS reliability. For a given AVS development project, all four metric elements have to be measured or assessed. Each metric element has a default value as shown in Table 1.

Table .1 Metric elements default values

Metric Element	Default Value
Data Handling (D)	3
Interoperability (I)	8
Fault Handling (F)	6
Configurability (C)	8

AVS RELIABILITY PREDICTION ALGORITHM

In this section, we describe our proposed AVS reliability prediction algorithm. This algorithm is based on fuzzy logic and it uses AVS reliability metrics. Figure 1 depicts an outline of the prediction algorithm for a given AVS development project.

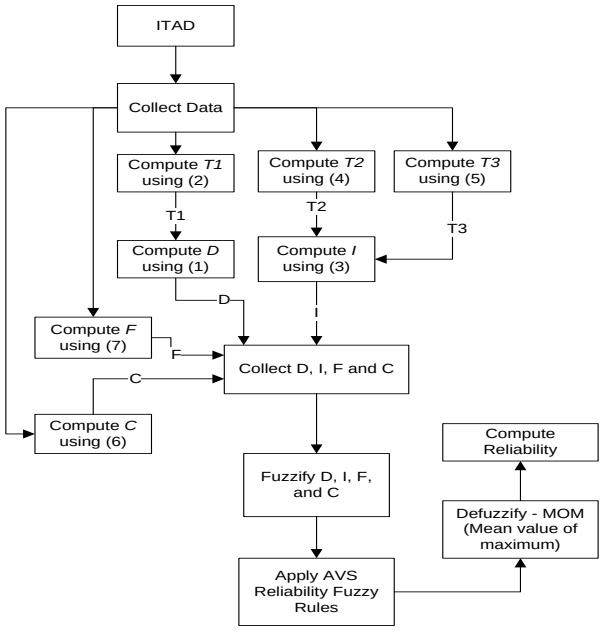

Figure.1 AVS Reliability prediction algorithm outline

The Algorithm is as follows:

1. *For every AVS project*
2. *Docs ← Collect ITAD()*
3. *Begin*
4. *Data ← Populate Data (Docs)*
5. *MetricData ← ReadInputs (Data)*
6. *D ← Compute DataHandling(Metricdata)*
7. *I ← Compute Interoperability (Metricdata)*
8. *F ← Compute FaultHandling(Metricdata)*
9. *C ← Compute Configurability (Metricdata)*
10. Fuzzy_inputs ← fuzzify (D,I,F,C)
11. Fuzzy_rules ← GetFuzzyRules ()
12. Y ← AppliedFuzzyRule
13. For each fuzzy_rule in the Fuzzy_rules
14. Begin
15. Y ← ApplyFuzzyRules (D,I,F,C, Fuzzy_rule)
16. Y ← AggregateUsingMoM(Y)
17. End for
18. R ← ComputeReliabilityFromDefuzzfiation (Y)
19. End for

In the algorithm, the first few steps are for deriving AVS reliability metrics using ITAD information. The details of the derivation are discussed in earlier sections. In this section, we describe the prediction portion of the algorithm.

We predict AVS reliability based on AVS reliability metrics ('D', 'I', 'C', and 'F'). The prediction is performed using an approximation process. Because of the vagueness in the metrics values, approximation has to be done using imprecise data. We use fuzzy logic [12] for this situation. Fuzzy logic is a soft computing heuristic technique based on the fuzzy sets theory [11]. Refer to [13] for additional information on fuzzy logic.

Fuzzy set of elements have different truth values (membership grades) ranging between 0 and 1. For example, a fuzzy set of very tall people can be represented as S= {7', 7'5", 6'5", 6', and 5'}, using expert knowledge, the grades of membership for this set can be defined as 0.9, 1, 0.8, 0.7, and 0.3. In this example, 'very tall people" is a linguistic term to represent a set of people's heights.

If X is a set denoted by Y, then a fuzzy set S in X is a set of ordered pairs i.e., $S = \{(x, \mu Y(x)) \mid x \varepsilon X\}$, e.g., S = {(7', 0.9), (7'5", 1), (6'5", 0.8), (6', 0.7), (5', 0.3)}.

After the metric values are calculated, the algorithm uses the following main steps to predict AVS reliability as a number.

1. Fuzzify inputs
2. Apply fuzzy rules
3. Defuzzify results

We explain our algorithm using an example. Assume the ITAD are investigated and the values of 'D', 'I', 'C', and 'F' are calculated to be 'D' = 0.225, 'I' = 0.45, 'C' = 0.4, and 'F' = 0.26.

The fuzzification or fuzzify step assigns a specific membership to each crisp input. The membership is a membership function and a membership grade. We suggest using a triangular membership function for each input (Figure 2 through Figure 5). A membership function is a plot of points which reflect how each input point from the input space is mapped to membership grades ranging between 0 and 1.

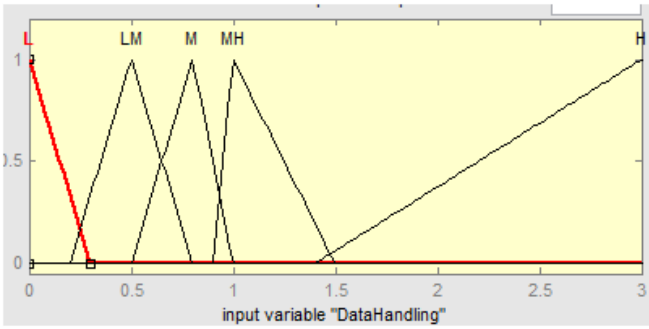

Figure.2 Data Handling (D) fuzzy membership functions

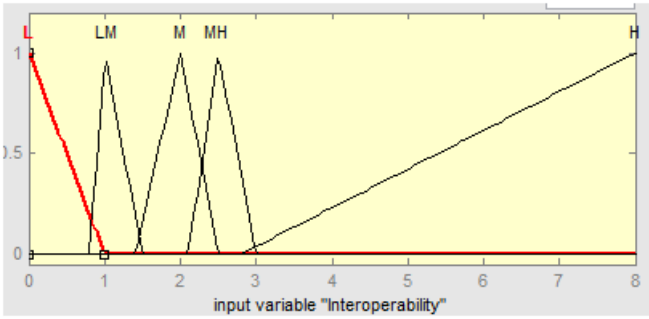

Figure.3 Interoperability (I) fuzzy membership functions

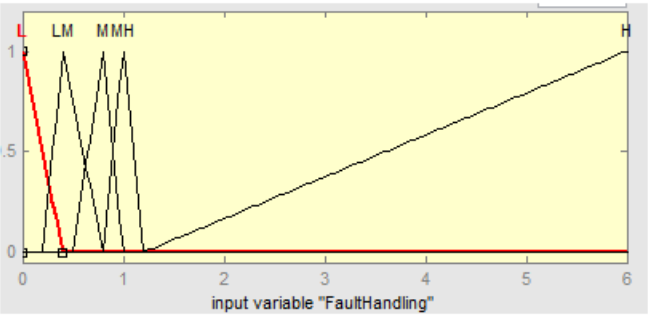

Figure.4 Fault Handling (F) fuzzy membership functions

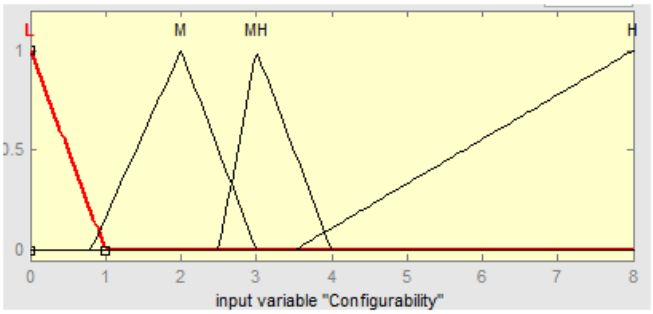

Figure.5 Configurability (C) fuzzy membership functions

Table .2 Linguistic triangular membership variables

Linguistic terms	D Range	I Range	C Range	F Range	Reliabiliy Range
L	0.0 - 0.3	0.0 - 1.0	0.0 - 1.0	0.0 - 0.4	0.0 - 0.3
LM	0.2 - 0.8	0.8 - 1.5	0.8 - 3.0	0.2 - 0.8	N/A
M	0.5 - 1.0	1.4 - 2.5	2.5 - 4.0	0.5 -1.0	0.2 - 0.6
MH	0.9 - 1.5	2.1 - 3.0	N/A	0.8 - 1.2	N/A
H	1.4 - 3.0	2.8 - 8.0	3.5 - 8.0	1.2 - 6.0	0.5 - 1.0

In Table 2, the 'Linguistic terms' column represents a membership function's labels. The 'D' value of 0.225 belongs to two functions i.e., 'L' and 'LM'. The fuzzification (Figure 6) process determines an appropriate membership assignment for it.

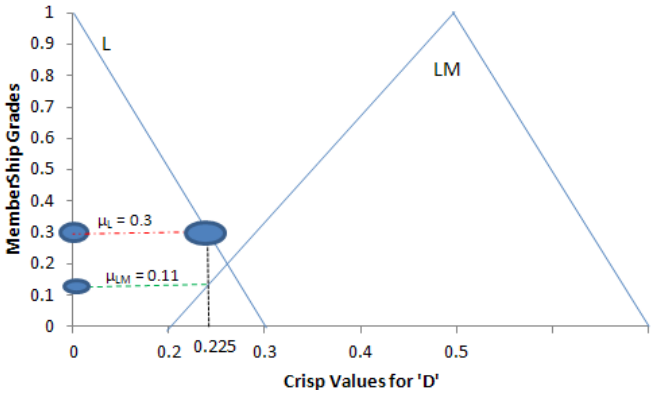

Figure.6 Fuzzification process

As shown in Figure 6, 0.225 has a membership grade μ_{LM} = 0.11 for the 'LM' membership function and a membership grade μ_L = 0.3 for the 'L' membership function. In this situation, the fuzzification process assigns a maximum of two membership grades i.e., max (μ_{LM}, μ_L). For 0.225, the 'L' membership function with a membership grade of 0.3 is the maximum. If 'L' had no overlapping with 'LM', then

there is no need for a maximum function check, and a μ_L value of 0.3 will be assigned. This fuzzified number is called a fuzzy number. Similarly 'I', 'F', and 'C' inputs can be fuzzified.

In the next step of the algorithm, all the fuzzy inputs will be verified against expert knowledge based fuzzy rules (Figure 7). This process is called a fuzzy reasoning process also known as a fuzzy inference. Every rule is applied to all the fuzzy inputs and the results from each rule are aggregated using a "maximum of mean value" function.

1. If (D is L) and (I is L) and (F is L) and (C is L) then (Reliability is H) (1)
2. If (D is L) and (I is LM) and (F is L) and (C is L) then (Reliability is H) (1)
3. If (D is L) and (I is L) and (F is LM) and (C is L) then (Reliability is H) (1)
4. If (D is L) and (I is L) and (F is L) and (C is M) then (Reliability is H) (1)
5. If (D is LM) and (I is L) and (F is L) and (C is L) then (Reliability is H) (1)
6. If (D is LM) and (I is LM) and (F is L) and (C is L) then (Reliability is M) (1)
7. If (D is LM) and (I is L) and (F is LM) and (C is L) then (Reliability is M) (1)
8. If (D is LM) and (I is LM) and (F is L) and (C is L) then (Reliability is M) (1)
9. If (D is LM) and (I is L) and (F is L) and (C is M) then (Reliability is M) (1)
10. If (D is LM) and (I is L) and (F is LM) and (C is M) then (Reliability is M) (1)
11. If (D is M) and (I is L) and (F is L) and (C is L) then (Reliability is M) (1)
12. If (D is MH) or (I is H) or (F is H) or (C is MH) then (Reliability is L) (1)
13. If (D is H) or (I is MH) or (F is MH) or (C is H) then (Reliability is L) (1)

Figure.7AVS reliability prediction fuzzy rules

Fuzzy rules are expressed in linguistic statements and they represent how the algorithm decides to assign a fuzzy input to a fuzzy output space. For example, in rule #1 (Figure 7), "if (D is L) and (I is L) and (C is L) then the reliability is H," L and H are linguistic labels used to represent membership functions. When the 'and' operator is used in a rule, the result of the rule application will get the minimum membership value out of all the fuzzy inputs used in the rule. For example, assume the following membership grades/values are assigned based on the fuzzify step for each input ('D', 'I', 'F', 'C'): $\mu_D = 0.3$, $\mu_I = 0.4$, $\mu_F = 0.2$, and $\mu_C = 0.1$. With these membership grades, when the rule#1 is applied, the result will be as follows:

$$D \wedge I \wedge F \wedge C = \min(\mu_D \mu_I \mu_F \mu_C) \quad (8)$$

When the 'or' operator is used in a rule, the result of the rule application will get the maximum membership value out of all the fuzzy inputs used in the rule. For example, assume the following membership grades/values are assigned based on the fuzzify step for each input ('D', 'I', 'F', 'C'): $\mu_D = 0.3$, $\mu_I = 0.4$, $\mu_F = 0.2$, and $\mu_C = 0.1$. With these membership grades, when the rule#12 is applied, the result will be as follows:

$$D \wedge I \wedge F \wedge C = \max(\mu_D \mu_I \mu_F \mu_C) \quad (9)$$

From (8), the result of the rule#1 is min (0.3 0.4 0.2 0.1) = 0.1. From (9), the result of the rule #12 is min (0.3 0.4 0.2 0.1) = 0.4. Similarly all the rules will be applied and a resulting output distribution is obtained (Figure 8).

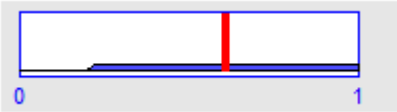

Figure.8 Output distribution

Defuzzification or defuzzify is the next step in the algorithm. In this step, the output distribution obtained from the previous step is aggregated and a mean of maximum (MOM) function is applied to it to get a crisp output number. A detailed explanation of MOM defuzzification process is found in [14]. Figure 9 shows AVS reliability output membership functions. Using the output membership functions, the MOM process maps a crisp output number. The number obtained from the defuzzification process is the AVS reliability prediction number. This number is the predicted probability that the AVS will work defect-free. For example, if AVS reliability prediction number is 1, then it means an AVS is 100% defects-free. If the number is 0, then the AVS is full of defects. If the number is 0.5, then the AVS is 50% defect-free. Figure 10 shows the entire fuzzy logic process to obtain AVS reliability prediction number. Based on the example dataset, the predicted AVS reliability number is 0.615 (Figure 10).

Based on the current AVS reliability number obtained during a given development phase, project managers can take corrective measures to improve the current development situation by planning to capture missing details. Higher the predicted AVS reliability number, lesser the probability of defects.

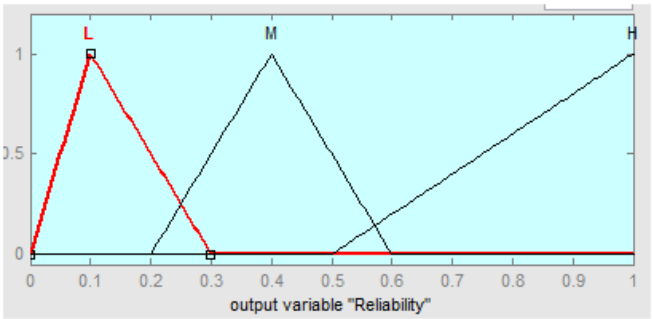

Figure.9 AVS reliability Fuzzy membership functions

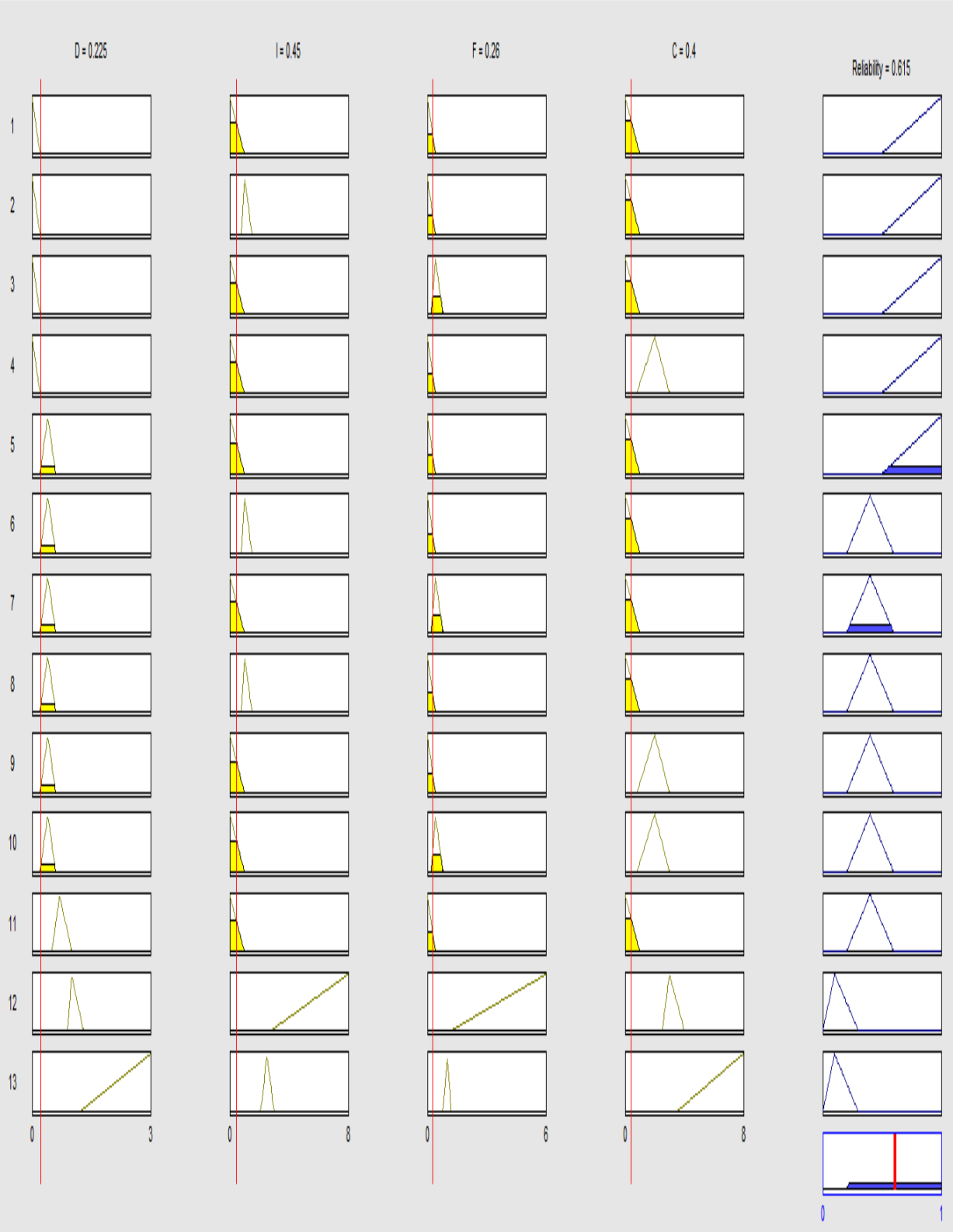

Figure.10 AVS reliability Fuzzy membership functions

CONCLUSION

We introduced the AVS reliability prediction metrics, which are simple to use so that a person with ordinary computer skills can investigate ITAD and collect data for calculating metrics values. The AVS reliability prediction algorithm is simple and a tool can be developed to perform predictions. Since our research is in its initial stages, we are in the process of validating and improving our work on a practical AVS development project

DISCLAIMER

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors- expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

REFERENCES

[1] Department of Defense, "A Guide for Achieving Reliability, Availability, and Maintainability", Dimensions," *DoD Guide,* 2005.

[2] A. Immonen, and E. Niemela, "Survey of Reliability and Availability Prediction Methods from the Viewpoint of Software Architecture," Software and Systems Modeling, 2007.

[3] S. Mohanta, G. Vinod, A. K. Gosh, and R. Mall, "An Approach for Early Prediction of Software Reliability," ACM SIGSOFT Software Eng. Notes, vol 35, num. 6, pages 1-9, 2010.

[4] W.Wang, Y.Wu, and M. Chen, "An Architecture-based Software Reliability Model," Proceedings of the Pacific Rim International Symposium on Dependable Computing, pages 143–150, 1999.

[5]K. Goseva-Popstojanova, K. S. Trivedi. "Architecture–Based Approach to Reliability Assessment of software systems," Performance Evaluation 45, pages 179-204, 2001.

[6] S. S. Gokhale and K. S. Trivedi, "Analytical Models for Architecture-based Software Reliability Prediction: A Unification Framework," IEEE Trans. on Reliability, 55(4), pages 578–590, 2006.

[7] N. Nagappan, Williams, L., Vouk, M.A., "Towards a Metric Suite for Early Software Reliability Assessment," International Symposium on Software Reliability Engineering, FastAbstract, Denver,CO, pages 238-239, 2003.

[8] C. Smidts and D. Sova, "An Architectural Model for Software Reliability Quantification: Sources of Data," Reliability Engineering & System Safety, 64(2), pages 279–290, 1999.

[9] W. Kong, Y. Shi, and C. S. Smidts, "Early Software Reliability Prediction Using Cause-effect Graphing Analysis," The 53rd Annual Reliability and Maintainability Symposium (RAMS 2007), pages 173 - 178, 2007.

[10] S. Yacoub, B. Cukic, and H. H. Ammar, "A Scenario-Based Reliability Analysis Approach for Component-Based Software," IEEE Transactions on Reliability, vol 53, num. 4, pages 465-480, 2004.

[11] L. A. Zadeh "Fuzzy Sets," Inform. Contr., vol 8, num. 3, pages 338-353, 1965.

[12] L.A. Zadeh, "Is There a Need for Fuzzy Logic," Information Sciences, vol 178, pages 2751-2779, 2008.

[13] L. Zadeh, "Fuzzy Logic," Computer, Vol 21, Num. 4, pages 83–93, 1988

[14]http://www.cs.princeton.edu/courses/archive/fall07/cos436/HIDDEN/Knapp/fuzzy004.htm