Army Vehicle Software Complexity Prediction Metric – Five Factors

Macam S Dattathreya and Harpreet Singh, *Senior Member, IEEE*

Abstract— Army vehicle software provides mission critical complex functions. It interacts with complex electronics from multiple vendors and has unique software interfaces. The software structure complexity is influenced by many factors prior to software development. Understanding, predicting and resolving complexity of vehicle software prior to its development is a necessity for army mission success. Current complexity metrics are historical data distribution dependent. It focuses on software and its technical structure with no consideration of its influencing factors. Non-technical metrics related to software complexity are required to address diverse skill set including the management. Using non-technical metrics prior to software development enables management to spend resources early to resolve issues faster. In this paper, the authors propose five non-technical factor metrics based on the current software development process to predict future Army vehicle software complexity. Factor analysis and fuzzy logic techniques are used for developing, modeling and analyzing the software complexity prediction metric. The proposed metric is independent of software, programming language, and domain. This metric is data distribution independent.

Index Terms—Maintainability, Metrics/Measurements, Process metrics, Reliability

1 Introduction

1.1 Army Vehicles and its Software

ARMY is one of the military divisions specialized in land warfare. In a given war mission, the war zone is very hostile, the terrain is unknown and the Army has to perform several critical complex functions to defeat enemy forces. To carry out a mission, the Army uses many different types of vehicles including armored fighting, medical, reconnaissance, fire support, and mortar carrying vehicles. These vehicles have multiple electronic devices, networks, and computing resources. The vehicles are built with tight security rules, multiple unique devices integration, multiple unique legacy software integration, strict performance requirements, involvement with human lives, and precision requirements.

The vehicle on-board devices are from multiple vendors and have unique software interfaces. These devices interact with each others using a complex network. Besides interacting within a given vehicle, these devices have to communicate with other devices on other vehicles and commanding centers. The Army battle success depends on effective interoperable communications between these devices and its software.

The Army vehicles' devices are real-time and are controlled by embedded operating systems. The Army vehicle software on these devices is developed using Ada, Java, C, C++, and etc programming languages. The Army vehicles have software related to radios, communications, navigation, fighting, training, diagnostics, and etc. For the Army, the vehicle software is crucial and any complexity in it hinders the mission success. The Army software is developed under intense tight requirements using multiple vendors. The software has millions of lines of code. They are developed using many different programming languages and require efficient integration to minimize possible software complexity. The Army has to focus more on its vehicle software complexity than any other commercial software, because, the Army environment is dynamic and its requirements are changing frequently to meet mission needs. The complex software in an Army vehicle introduces many defects and makes it difficult to understand the software and correct the defects in a relatively faster pace. The Army cannot afford to have defects in any critical functions. If the software cannot accommodate frequent changes in a faster way, the vehicles cannot perform its intended function and it is not acceptable to the Army.

The software complexity is a major concern for any Army vehicle software to maintain its peak performance at all time. There is a bigger need for identifying metrics to predict the Army vehicle software complexity in very early stages of its development cycle. Many researchers have postmortem historical software and identified the reasons why a given software structure is complex, but, to our knowledge, no body has made an attempt to identify the Army vehicle software complexity contributing factors by inspecting the current software development process. Many researchers have defined several software complexity metrics but they all are historical technical data dependent and may not work in all software development.

The complexity is influenced by many factors prior to software development. Understanding, predicting and

- *Macam S Dattathreya is with the Tank Automotive Research, Development and Engineering Center, Warren, MI 48397.*
- *H. Singh is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202 USA.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

xxxx-xxxx/0x/$xx.00 © 200x IEEE

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
14 APR 2010	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Army Vehicle Software Complexity Prediction Metric Five Factors(PREPRINT)	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Macam S Dattathreya; Harpreet Singh	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
US Army RDECOM-TARDEC 6501 E 11 Mile Rd Warren, MI 48397-5000, USA Wayne State University, Detroit, MI	**20723**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
US Army RDECOM-TARDEC 6501 E 11 Mile Rd Warren, MI 48397-5000, USA	**TACOM/TARDEC/RDECOM**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
	20723

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES

Submitted for publication in IEEE Transication on Software Eengineering, The original document contains color images.

14. ABSTRACT

Army vehicle software provides mission critical complex functions. It interacts with complex electronics from multiple vendors and has unique software interfaces. The software structure complexity is influenced by many factors prior to software development. Understanding, predicting and resolving complexity of vehicle software prior to its development is a necessity for army mission success. Current complexity metrics are historical data distribution dependent. It focuses on software and its technical structure with no consideration of its influencing factors. Non-technical metrics related to software complexity are required to address diverse skill set including the management. Using non-technical metrics prior to software development enables management to spend resources early to resolve issues faster. In this paper, the authors propose five non-technical factor metrics based on the current software development process to predict future Army vehicle software complexity. Factor analysis and fuzzy logic techniques are used for developing, modeling and analyzing the software complexity prediction metric. The proposed metric is independent of software, programming language, and domain. This metric is data distribution independent.

15. SUBJECT TERMS

Maintainability, Metrics/Measurements, Process metrics, Reliability

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**SAR**	**10**	

resolving complexity of vehicle software prior to its development is a necessity for Army mission success. In this paper we describe the proposed software complexity prediction metric for Army vehicle software.

1.2 Software Complexity

Honglei et al [1] summarizes the software complexity definition as difficulty of understanding the program, difficulty of correcting the defects and maintaining the software, difficulty of explaining the software to other people, difficulty of updating the program according to some assigned rules, work load of writing programs according to the design, and availability of necessary resources when programs are executing. Kearney et al [2] defines software complexity as applying to the interaction between a program and a programmer working on some programming task.

We define true software complexity as a combination of three main elements i.e. reliability, availability, and maintainability (RAM). There are many features that may be qualified as software complexity elements, but, from our experience the main complexity is truly a result of RAM. The definitions of RAM described in next few lines confirm that the software complexity is a combination of RAM. Reliability (R) is the probability of performing a required function under stated conditions for a specified period of time [3]. Availability (A) is a measure of the degree to which software is in an operable state and can be committed at the start of a mission when called for at an unknown (random) point in time. Availability as measured by the user is a function of how often failures occur and corrective maintenance is required, how often preventative maintenance is performed, how quickly indicated failures can be isolated and repaired, and how quickly preventive maintenance tasks can be performed [3]. Maintainability (M) is the ability of software to be retained in, or restored to, a specified condition when maintenance is performed by personnel having specified skill levels, using prescribed procedures and resources, at each prescribed level of maintenance and repair [3]. The software complexity influences a lot of software defects and introduces unknowns when it comes to fixing them. If the software is complex, its reliability is hard to achieve, due to this, the software may suffer availability issues. If the software is hard to understand, hard to fix defects, then the maintainability suffers. The bottom-line is, the RAM controls the entire software dependability, and without it the Army cannot perform its intended functions. These arguments confirm that the software complexity is really consisting of RAM elements.

Many existing software complexity metrics are historical data distribution dependent and they focus only on software & its technical structure with no consideration of its influencing factors. These metrics provide unique software attributes to define complexity and they are programming language or industry specific. These complexity metrics require skilled resources to understand, implement, and resolve complexity. Identifying complexity based on one source of software historical data and applying it to another may not work for all software. There are multiple factors which contribute to its complex structure. Management may not understand the technical complexity data obtained from the historical data. In many cases, the developers may not be skilled enough to understand this to implement a good solution. Convincing management to spend resources on resolving problems in the development phase just based on the technical complexity data is very hard. Many skilled developers and designers must be assigned to fix issues. This could be expensive and the problems may not be solved due to tight schedules and wrong historical data. Basing the decisions on these metrics and attempting to reduce complexity may work in some but not in all cases because the factors which contributed to complexity are still not examined. Historical data change when the contributing factors and the target software are changed. These metrics are complex mathematical formulations and need tools to find the elements proposed by these researchers to collect data from the historical complex software. These metrics may not be easily understood by all applicable parties. There is a need for non-technical metrics which are common to any software development and for any industry.

The McCabe's [4] software complexity metrics are based on a software program's control flow. It introduces the concept of Cyclomatic Complexity, where the number of flow graph edges, nodes and predicate nodes are combined to represent the complexity. The Cyclomatic Complexity of a source code is the linearly independent paths count through the source code. It is mathematically defined as $C = E - N + 2P$, where E = number of edges of the graph, N = number of nodes in the graph, and P = number of predicate nodes.

The Halstead [5] software complexity metric provides the measure of software algorithm complexity. It measures the complexity by counting number of operators and operands in software. It measures the software's ability to understand and estimates the effort required to develop a software algorithm. It also indicates the amount of time to implement an algorithm. Halstead metrics are difficult to calculate and it is very hard to count the distinct and total operators and operands in a software program.

The Henry and Kafura [6] software complexity metric provides the measure of couplings between modules in terms of number of parameters, global variables and function calls. It measures given software's procedure, module and interfaces. Henry and Kafura believe that measurement of software quality for large scale systems using information flow to represent the system interconnectivity is an important visible technique.

The Entropy based software complexity measure [7] is based on the average information content of each operator in a software program's source code.

The Cognitive weights software complexity metrics [8] from Jingqiu Shao and Yingxu Wang models the software complexity based on the cognitive functional size of the software. The cognitive functionalize provides a foundation for cross-platform analysis of complexities, sizes and comprehension effort of software specifications in various design phases.

The Relative complexity metrics [9] represents a single,

unified measure on the structure of a software program. It serves to classify a set of software programs in order of their increasing complexity in relation to each other.

The [4], [5], [6], [7], [8], and [9] metrics discussed earlier are too technical and focus only on technical structure of a software program. These data are hard to compute and requires many skilled resources to understand and implement solutions.

Above arguments lead to a conclusion that we need metrics captured from the current software development process documents rather than the software itself. These metrics must be easily understood by both the technical and non- technical resources. These metrics can be used to measure the current software development process elements and predict the future software complexity. If the software complexity is predicted early enough in the software planning lifecycle, the complexity can be managed through many preventive measures such as revisiting the requirements, restructuring the development team, scheduling for reviews, modifying the testing strategy, etc. With early prediction, less effort can be put into the software development and restructuring the executing plan. If the complexity is predicted using the factors which are available from the beginning of a software development program, it makes it easy for the management to visualize and act fast.

We describe the proposed software complexity prediction metric in the following sections.

1) The prediction metric development - using factor analysis
2) The prediction algorithm - using fuzzy logic
3) The prediction model development - using fuzzy logic

2 THE PREDICTION METRIC DEVELOPMENT

2.1 Metric Development

The following ten non-technical variables are extracted from the Army vehicle software development project plan, development strategy, test strategy, technology strategy, and requirements analysis documents.

1) Technical readiness level (TRL)
2) Number of planned skilled resources (PSR)
3) Number of planned code reviews (PCR)
4) Number of planned design reviews (PDR)
5) Number of planned architecture reviews (PAR)
6) Number of planned integration reviews (PIR)
7) Number of planned design documents (PDC)
8) Number of planned test case documents (PTD)
9) Number of planned configuration management tasks (CM)
10) Number of open requirements (OR)

In an Army vehicle software development process, characteristics of these variables influence schedules, number of defects, cost, number of modules, use of best practices, etc. Issues from these variables compounds and results in a complex software structure which is responsible for a less reliable and available, and hard to maintain software. The above list of multiple dimensions with many mutually correlated variables is reduced to a smaller set of uncorrelated variables with conceptual indices to measure similar factors. Dr. Deok H Nam et al [10] data reduction technique reduces the data in both the rows and columns. This technique reduces dataset based on the historical data distribution and is applicable when the measurement variables have no impact on the reduced data set. For the proposed metrics development, too much data reduction produces bad results.

The factor analysis technique is applied to reduce a multitude of measurable variables to smaller manageable factors. The factor analysis extracts factors from a sample of data collection (observations) where variables are distributed in a consistent manner, e.g. code review is related to the number of defects in a software test, because, code with higher reviews produces fewer defects.

TABLE 1
ROTATED FACTOR LOADINGS

Variables	F1	F2	F3	F4	F5
TRL	-0.062	-0.034	-0.940	-0.002	0.048
PSR	0.604	0.405	-0.463	0.281	0.132
PCR	0.393	0.103	0.492	-0.716	0.007
PDR	0.026	0.896	0.268	0.154	0.124
PAR	0.275	0.390	0.180	0.812	0.133
PIR	0.707	0.455	-0.130	-0.203	-0.127
PDC	- 0.108	0.022	-0.053	0.072	0.978
PTD	0.232	0.731	-0.286	0.082	-0.096
CM	0.875	-0.077	0.123	0.030	-0.157
OR	0.674	0.510	0.315	0.121	0.108

TABLE 2
EIGENVALUES (VARIANCE)

Factors	Eigenvalue
F1	2.383
F2	2.138
F3	1.661
F4	1.343
F5	1.071
Communality	8.595

A factor analysis function was applied to 24 Army vehicle software development projects data (see Fig. 10) to produce a covariance output using the principle component analysis, Varimax rotation and the Kaiser criterion. The covariance was observed to capture common conceptual indexes from the factor analysis output. Five factors from the ten mutually correlated variables were extracted. The Kaiser criterion retains only factors with eigenvalues greater than 1.0. The eigenvalues are the variance of the extracted factors. The Eigen-value for a given factor measures the variance in all the variables which is accounted for by that factor. TABLE 1 lists the factor analysis output from the analysis tool. Ten variables with a variance of one for each transform to a total extracted variability of 10 (10*1). From TABLE 2, it is clear that all factor's Eigen-values are over one and five extracted factors show 85% of the variances from ten variables. This reduction emphasizes that fewer variables in the selected pool

could be conceptually connected to achieve the intended function.

Statistics is a mathematical formulation based on a past data distribution snapshot and may not be a future predictor. The environment is dynamically changing with time and might pose a different situation. In the our's and other researcher's experience, historical data factor analysis cannot solve the reduction problem in all cases. This has to be integrated with human skills in interpreting and identifying the correct common indexes.

Per TABLE 1 (gray highlighted text), PCR, PDR, PAR, PDC, and CM show higher covariance among other variables. If one were to choose purely on this analysis, it would be badly judged, as PCR, PDR, and PAR can be linked to a single factor. In addition to analysis data and our experience dealing with software, the proposed software complexity prediction factors are listed in TABLE 3. F1 can consist of PCR, PDR, PAR, PIR and PSR. F2 can consist of PDC & PTD. F3 can consist of TRL which plays a big role in any software development. F4 can consist of OR alone because ORs create a greater impact on any software development. F5 can consist of CM alone because CM is very important in making sure the developed software is properly controlled and configured.

Based on the earlier discussions, we propose the following five factors metric to predict software complexity.

1) Technical readiness level (TRL)
2) Number of open requirements (OR)
3) Number of planned technical reviews (TR)
4) Number of planned documentation tasks (DOC)
5) Number of planned configuration management tasks (CM)

As previously stated, the true software complexity is a combination of R, A, and M. The DOD's RAM guide [3] defines RAM essentialness to military systems and software and describes various affecting factors. Subsequent paragraphs describe the proposed metric elements and its association with R, A, and M.

For predicting the Army vehicle software complexity, all the five factors must be considered because the combination of factors predicts R, A, and M component of the software complexity as shown below.

1) TRL, TR, and OR factors predict reliability.
2) TRL and TR factors predict availability.
3) DOC and CM factors to predict maintainability.

2.2 Metric Elements

Technology readiness level (TRL)

The TRL measures evolving technologies maturity prior to its implementation. The readiness is indicated by 1 to 9 levels.

TRL1) Basic principles observed and reported
TRL2) Technology concept and/or application formulated
TRL3) Analytical and experimental critical function and/or characteristic proof of concept
TRL4) Breadboard validation in laboratory environment
TRL5) Breadboard validation in relevant environment
TRL6) Model or prototype demonstration in a relevant environment
TRL7) Prototype demonstration in an operational environment
TRL8) Actual system completed and 'flight qualified' through test and demonstration
TRL9) Actual system 'flight proven'

A technology with a lower TRL contributes to frequent failures when the software is developed. This creates lower mean time between failures (MTBF), increased downtime, lower meantime between repairs (MTBR), etc. When the TRL level is more than six it is considered mature enough to provide good reliable software. Higher MTBF indicates more reliable software and decreased downtime. Lower MTBR reduces the availability of software to perform intended functions. Careful analysis must be performed before a given technology is chosen to satisfy a given Army vehicle software requirement. More resources and efforts are needed to make lower TRL technology work and it cause a lot of problems. The TRL is a very good indicator of future R & A of given software.

TABLE 3
PROPOSED PREDICTION METRIC

Factors	Named Factor
F1	Number of planned technical reviews (TR)
F2	Number of planned documentation tasks (DOC)
F3	Technical readiness review (TRL)
F4	Number of open requirements (OR)
F5	Number of planned configuration management tasks (CM)

Number of open requirements (OR)

Open requirements have issues & unanswered questions. Unknown clarity on the requirements contributes to misunderstood requirements, increased redesigns, missed schedules, skipped technical documentation, cutting design corners, un-maintainable complex modules susceptible to higher failures and defects, etc. These characteristics jeopardize the reliability of future Army vehicle software.

Number of planned technical reviews (TR)

Technical reviews are performed during software development, design, and test phases to find problems soon. If the Army vehicle software development has planned for relatively fewer required technical reviews, it will be hard to find the problems in the code, design, test cases, and architecture. Fewer planned technical reviews increases rework, redesign, bad coding practices, failures, defects, etc. The technical reviews consist of code, design, architecture, and integration reviews. These are designed to help software recover from possible future errors or defects or downtime. Lack of these planned reviews is a very good indicator of future R & A of Army vehicle software.

Number of planned documentation (DOC)

Tasks for creating technical documents are a must requirement for software development. The higher the number of documentation tasks scheduled for complex functionality the lower the data integrity, information assurance, interoperability, operational, maintenance,

testing, and rework issues. Documentation includes code, design, architecture, test cases, and requirements in order to reduce future unknowns and maintenance issues. This is a very good indicator of future M of Army vehicle software.

Configuration management (CM)

Configuration management for software is vital to the success of Army vehicle software. CM allows all parts and versions of software to be properly integrated and documented. A greater number of configuration management tasks scheduled for complex functionality reduce information assurance, interoperability, operational, maintenance, and testing issues, as well as MTBF, MTBR, MTTR, system downtime. CM tasks such as source and documentation control, release schedules, etc contributes to a reduced logistics and maintenance footprint. This is a very good indicator of future M of Army vehicle software.

3 THE SOFTWARE COMPLEXITY PREDICTION ALGORITHM

Software complexity prediction using a number of inputs is tricky and is not always precise. Many mathematicians tend to apply complex math to derive expressions to obtain near accurate results. The software development planning resources consist of both technical and non-technical personnel, and the software discipline needs simpler methods and tools to evaluate complex phenomena such as software complexity during the planning phase. For this situation, fuzzy logic solution offers great advantages to solve complex problems using a number of inputs. Fuzzy logic [11] proposed by L. A. Zadeh has been used where uncertainty and no mathematical relations exists. Software complexity research area is vast and very hard to predict or analyze with the limited amount of measurable data. Empirical studies consume time and produce lower fidelity results. Fuzzy logic provides rule based approaches to solve a given problem using simple steps. We propose the following Army vehicle software complexity prediction algorithm using fuzzy logic (see Fig.1 for algorithm flow chart).

Step1: Read String array inputs S = {Software 1…. Software N} for predicting software complexity;
Step2: for i=1 to N (for each software)
// refer section 2 for the details of TRL, TR, OR, DOC, and CM
N (1) = collect TRL number from the technology strategy document for software (i);
N (2) = calculate number of planned technical reviews (TR) from the project plan for software (i);
N (3) = calculate number of open requirements (OR) from the requirements analysis document for software (i);
N (4) = calculate number of planned documentation tasks (DOC) from the project plan for soft ware (i);
N (5) = calculate number of planned configuration management tasks (CM) from the project plan for software (i);
M (i) = N; (Store N array for software (i) in M array).
end for
Step3: Read integer inputs array from M array;
// refer section 4 for fuzzy rules details.
Step4: Store Fuzzy rules in array X = {rule1.... rule15};
Step5: for i = 1 to N // Loop for computing software complexity for each software
W = M(i) //get the ith element from W array
for j=1 to 5
Y(i) = fuzzify (W(i));
end for
Step6: for i = 1 to 15
if i <=11 then
Z(i) = apply fuzzyrule(X(i)) on Y(1) & Y(2) & Y(3);
else
Z(i) = apply fuzzyrule(X(i)) on Y(4) & Y(5);
end if
end for
Step7: Compute Reliability, R = centroid De-fuzzufication of Z(1) to Z(11);
Compute Availability, A = centroid De-fuzzufication of Z(1) to Z(10);
Compute Maintainability, M = centroid De-fuzzufication of Z(12) to Z(15);
Step8: Predict Software complexity from R, A, and M values
end for // ends for loop for each software in a software array M.

Fuzzification [11] is a process of transforming crisp input values into grades of membership for linguistic terms of fuzzy sets. De-fuzzification is a process of producing a quantifiable result in fuzzy logic, given fuzzy sets and corresponding membership degrees. A fuzzy set is a class of objects with various grades of membership under almost any condition which can assign a value in between zero and one, in other words, a fuzzy set is a pair (*A*,*m*) where *A* is a set and m: A →[0,1]. Degree of membership is the grade displacement for the zero to one range in a fuzzy set. A fuzzy number is a convex, normalized fuzzy set whose membership function is at least continuous and has the functional value $\mu A(x) = 1$ at one element. Fuzzy sets and membership functions are defined mathematically using the following definition. If X is a collection of objects denoted by z, then a fuzzy set A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\} \quad (1)$$

$\mu_A(x)$ is called the membership function (MF) of x in A. The MF maps each element of X to a continuous membership value (or membership grade) between zero and one.

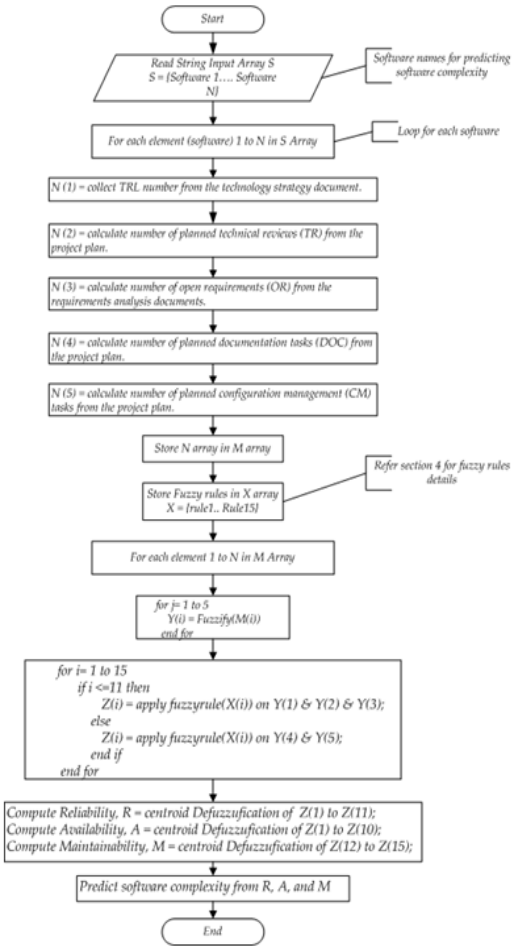

Fig. 1. Software complexity prediction algorithm flowchart

4 THE SOFTWARE COMPLEXITY PREDICTION MODEL

We propose a fuzzy logic based Army vehicle software complexity prediction model using the proposed five factors metrics. The software fuzzy logic toolbox was used to develop the model. The TABLE 4 lists the fuzzy variables and its mapping with the proposed metric elements.

The prediction model has three components i.e. fuzzification, rule-based fuzzy inference engine, and defuzzification. The model consists of five inputs, three outputs and 15 rules. According to the predefined rules, the model predicts the appropriate software complexity in terms of RAM. Both the fuzzy inputs and outputs are modeled using the trapezoidal membership functions. The membership grades for TRL are described by LOW, MEDIUM, and HIGH membership functions (see Fig. 5) and all other the fuzzy inputs are described by NOTHING, SOME and FULL membership functions (See Fig. 4). The fuzzy output's membership grades are described by RED, YELLOW, and GREEN membership functions (See Fig. 6).

Fig.2 describes the typical characteristics of a trapezoidal membership function. The core of a normal fuzzy set A is the crisp set that contains all the elements of X that have the membership grades of one in, A i.e.

$$\text{Core}(A) = \{x \in X \mid \mu_A(x) = 1\} \quad (2)$$

The boundary is the crisp set that contains all the elements of X that have the membership grades of $x < \mu_A(x) < 1$ in A. Fig. 3 shows the trapezoidal function details.

For all the fuzzy inputs and outputs, the core (A) is defined as follows:

Set A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Support (LOW) = Support (RED) = {0, 1, 2, 3}
Core (LOW) = Core (RED) = {1, 2}
Support (MEDIUM) = Support (YELLOW) = {2, 3, 4, 5, 6}
Core (MEDIUM) = Core (YELLOW) = {3, 4, 5}
Support (HIGH) = Support (GREEN) = {5, 6, 7, 8, 9}
Core (HIGH) = Core (GREEN) = {6, 7, 8, 9}

The fuzzy logic toolbox provides a rule-based model as a software prototype to analyze all the inputs and compute the output. The de-fuzzification rules are based on the proposed five factors metrics. The list below shows the 15 fuzzy rules to predict the software complexity using five inputs and three outputs.

1) If TRL= LOW & TR = NOTHING & OR =NOTHING then R = RED & A = RED
2) If TRL= LOW & TR = SOME & OR = NOTHING then R = RED & A = RED
3) If TRL = LOW & TR=FULL & OR=NOTHING then R=RED & A=YELLOW
4) If TRL=MEDIUM & TR=NOTHING & OR=NOTHING then R=RED & A=YELLOW
5) If TRL=MEDIUM & TR=SOME & OR=NOTHING then R=RED & A=YELLOW
6) If TRL=MEDIUM & TR=FULL & OR=NOTHING then R=YELLOW & A=YELLOW
7) If TRL=HIGH & TR=NOTHING & OR=NOTHING then R=RED & A=YELLOW
8) If TRL=HIGH & TR=SOME & OR=NOTHING then R=YELLOW & A=YELLOW
9) If TRL=HIGH & TR=FULL & OR=NOTHING then R=GREEN & A=GREEN
10) If OR=SOME then R=YELLOW
11) If OR=FULL then R=RED
12) If DOC=NOTHING || CM=NOTHING then M=RED
13) If DOC=SOME & CM=SOME then M=YELLOW
14) If DOC=SOME & CM=FULL then M=YELLOW
15) If DOC=FULL & CM=FULL then M=GREEN

R in the RED membership grades indicates that the reliability component of the Army vehicle software complexity is in trouble and needs significant improvements in "TRL" or "TR" or "OR". A in the RED membership grades indicates that the availability component of the Army vehicle software complexity is in trouble and needs significant improvements in "TRL" or "TR" factors. Similarly, M in the RED membership grades indicates that the maintainability component of the Army vehicle software complexity is in trouble and needs improvements in "CM" or "DOC" factors. The output values of YELLOW indicate that some improvements are needed for the associated factors. The output values of GREEN indicate no

improvements needed for the associated factors. The Fig. 8 describes the model elements.

The fuzzification model element fuzzifies the model inputs via a max function evaluation based on the membership functions defined to determine its appropriate membership grades. Fig. 7 shows the fuzzified input value of 2.8 as an example of fuzzification process.

In this model, when the inputs and outputs are fuzzified, the max function is applied between the membership functions. Per Fig. 7, the value of 2.8 falls into two membership functions i.e. LOW (0.4) and MEDIUM (0.8), but when max function is applied between them, the membership grade falls into MEDIUM membership function. The fuzzified value is 0.8 for an input of 2.8. In this model, when the results are de-fuzzied, they use the centroid method of de-fuzzification. This process returns the center area of the curve (see Fig. 9).

TABLE 4
FUZZY VARIABLES

Fuzzy Variables	Fuzzy Input/output	Associated prediction metric
TRL	Input	TRL
TR	Input	Technical Reviews (TR)
DOC	Input	Documentation (DOC)
CM	Input	Configuration management (CM)
OR	Input	Open requirements (OR)
R	Output	Reliability (R)
A	Output	Availability (A)
M	Output	Maintainability (M)

5 VALIDATION AND EXAMPLE

Let's explain the Army vehicle software complexity prediction model using the simple example. The following are the example data: TRL = 5, TR=4, OR=1, DOC=3 and CM=4.

Per the software complexity prediction algorithm (see Section III), all the above five inputs are fuzzified using the max function. The fuzzified input values are TRL = 1 (MEDIUM), TR = 1 (SOME), OR = 1 (NOTHING), DOC = 1 (SOME), CM = 1 (SOME).

Now the above fuzzified inputs are tested by 15 fuzzy rules (see Section IV). Per rule#5, *If TRL=MEDIUM & TR=SOME & OR=NOTHING then R=RED & A=YELLOW.* Per rule#13, *If DOC=SOME & CM=SOME then M=YELLOW* From the two fuzzy rules #5 & #13, software complexity can be predicted. The results indicate that the reliability part of the software complexity as RED (crisp values between 0 & 3). Availability part of the software complexity is YELLOW (crisp values between 2 & 6). The maintainability part of the software complexity is YELLOW (crisp values between 2 & 6). The software fuzzy logic toolbox can be used to simulate five factors metric input to determine fuzzy outputs and the appropriate defuzzified values. Depending on the output membership grades appropriate fuzzy value for the output can be determined from the fuzzy logic toolbox output.

6 CONCLUSION

Many existing software complexity metrics are historical data distribution dependent and focus only on software & its technical structure with no consideration of influencing factors for complex software structure. These metrics are too technical and requires skilled resources to understand and implement solutions to fix the software complexity. Identifying complexity based on one source of software historical data and applying it to another may not work for all software. Management may not understand the too technical complexity data obtained from the historical data. These types of metrics historical software data based and focus on fixing the symptoms rather than the problem.

The proposed software complexity metric fills this gap by providing non-technical variables (factors) to predict the Army vehicle software complexity. The proposed metric elements are known to all parties involved in a software development project and the metric data collection is simple. The data can be captured from the software development project plan, development strategy, test strategy, technology strategy, and requirements analysis documents. In an Army vehicle software development process, characteristics of these factors influence schedules, number of defects, cost, number of modules, use of best practices, etc. Issues from these variables compounds and results in a complex software structure which is responsible for a less reliable and available, and hard to maintain software. The proposed metric is independent of software, programming language, and domain. This metric is independent of data distribution and is suitable for any software development. This provides complexity information very early in the development cycle and allows applicable personnel to take actions immediately and resolve the issue before it happens. This metric does not require any technical solution fix the future software complexity and does not require skilled technical people. The resources involved in the project planning phase can easily understand software prediction output and restructure the proposed metric elements in the appropriate documents to resolve future software complexity problems. The true software complexity has three main elements i.e. reliability, availability, and maintainability.

The factor analysis data reduction technique, along with human logical analysis can be used to extract smaller uncorrelated factors from mutually correlated variables. Fuzzy techniques produce satisfactory results with very minimal effort compared to mathematical formulations. The input data distribution and the rule driven output values are analyzed using the fuzzy logic tool box which serves as a software prototype. Custom software can be developed to interact with the fuzzy logic toolbox to provide input and extract the fuzzy inference engine output for visualization.

When complexity is expressed in terms of RAM, the user can visualize which part of the complexity is having problems e.g. if R = YELLOW, A = GREEN, and M = RED then we can say the reliability and maintainability part of the software complexity is in trouble. To fix this problem, we need to adjust the factors responsible for them. By looking at the current data for TRL and TR, we can determine the gap and take the necessary actions for fixing the reliability issue as these three factors contribute the most for the reliability parts of the software complexity. The DOC & CM data can be used to determine the gap for maintainability and to take action.

The proposed software complexity prediction mechanism using five non-technical factors and three outputs is a novel technique. Historical RAM data are not available to compare the results. Based on our experience, these factors truly contribute to RAM.

This proposal opens up software complexity research gates to produce more simplified non-technical factors to predict software complexity using multiple inputs and outputs.

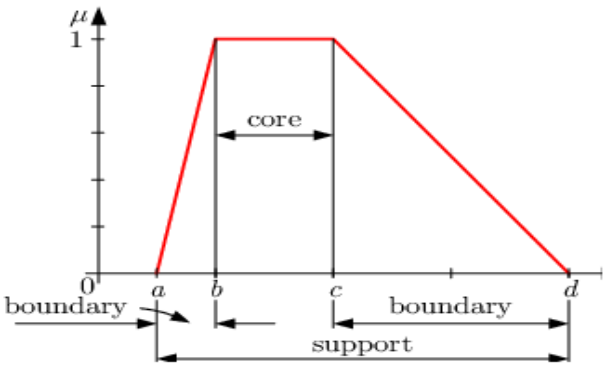

Fig. 2. Characteristics of trapezoidal membership function. This figure depicts the boundary, core, and support elements of a trapezoidal membership function.

$$\mu(x,a,b,c,d) = \begin{cases} 0, & x < a,\ x > d \\ \frac{x-a}{b-a}, & a \le x \le b \\ 1, & b < x < c \\ \frac{d-x}{d-c}, & c \le x \le d \end{cases},$$

Fig. 3. Trapezoidal function details. This figure describes the trapezoidal function behavior for the crisp inputs.

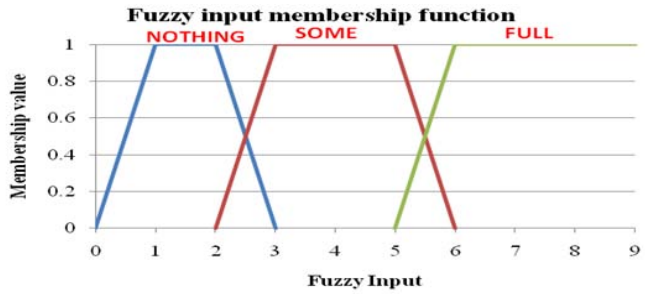

Fig. 4. Fuzzy input membership functions for TR, OR, CM, and DOC. This figure describes the three membership functions NOTHING, SOME, and FULL for TR, OR, CM, and DOC crisp input.

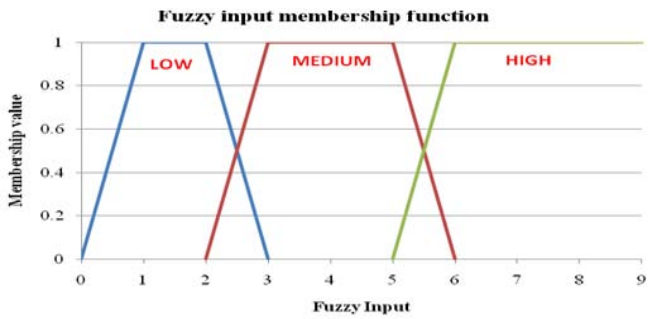

Fig. 5. Fuzzy input membership functions for TRL. This figure describes the three membership functions LOW, MEDIUM, and HIGH for TRL crisp input.

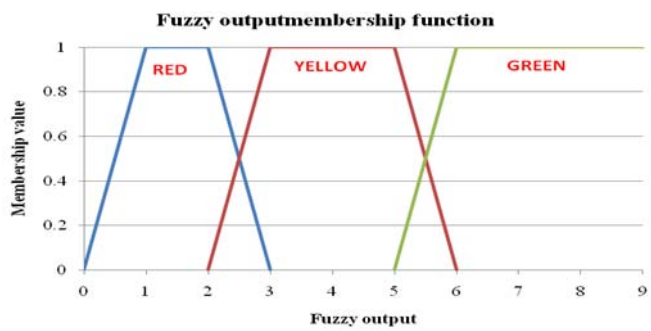

Fig. 6. Fuzzy output membership functions for R, A, and M. This figure describes the three membership functions RED, YELLOW, and GREEN for R, A, and M outputs.

Fig. 7. Fuzzified input for an example of 2.8 crisp input.

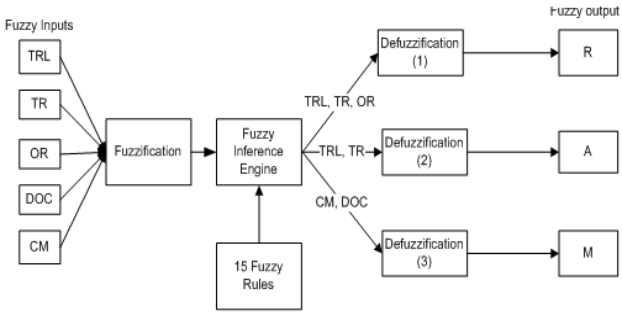

Fig.8. Army vehicle software complexity prediction model. This figure describes the Army vehicle software complexity prediction model.

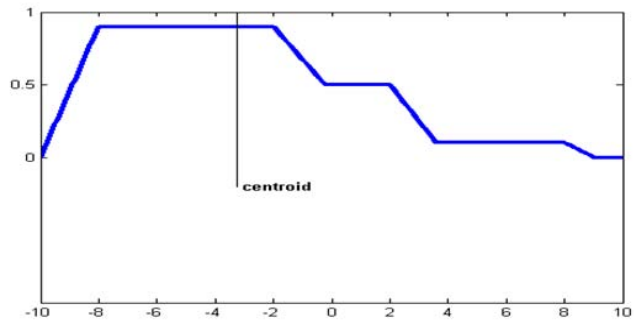

Fig.9. Centroid de-fuzzification. Courtesy: http://www.mathworks.com

Soft#	TRL	PSR	PCR	PDR	PAR	PIR	PDC	PTD	CM	OR
1	7	2	1	2	2	1	2	3	2	1
2	5	3	1	4	5	2	2	3	2	4
3	3	1	3	2	0	1	1	3	2	2
4	6	3	2	4	3	1	5	4	1	2
5	4	2	3	2	1	1	2	2	2	2
6	5	2	3	2	1	2	2	3	2	2
7	7	3	3	2	1	2	2	3	2	2
8	7	2	3	2	1	2	2	3	2	2
9	7	3	2	2	1	2	2	3	2	2
10	7	3	2	2	2	2	2	3	2	3
11	7	3	1	2	1	2	2	3	2	1
12	7	3	1	1	2	1	2	3	2	1
13	7	3	1	1	2	1	2	4	2	1
14	7	2	1	1	2	1	2	2	2	1
15	6	2	2	1	1	1	6	2	2	1
16	7	2	2	1	1	1	3	2	2	1
17	5	1	2	2	2	1	2	2	2	1
18	6	1	2	1	1	1	1	2	1	0
19	6	2	1	2	2	1	3	3	1	1
20	6	1	1	2	1	1	4	2	1	1
21	6	2	1	1	2	1	3	2	1	1
22	4	2	2	2	2	1	2	3	2	0
23	5	2	2	1	1	1	2	1	1	1
24	3	3	3	2	4	2	3	3	3	4

Fig. 10. Army vehicle software metric data from various documents

ACKNOWLEDGMENT

The authors wish to thank wishes to thank Pame Watts Dean of TARDEC University for her support.

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] Tu Honglei, Sun Wei, Zhang Yanan, "The Research on Software Metrics and Software Complexity Metrics", 2009 International Forum on Computer Science-Technology and Applications.

[2] K. Kearney et al, "Software Complexity Measurement", *ACM comm.*, Vol.29, Nov. 1986.

[3] Department of Defense, "A Guide for Achieving Reliability, Availability, and Maintainability", Dimensions," *DoD Guide,* http://www.acq.osd.mil/sse/docs/RAM_Guide_080305.pdf. 2005.

[4] T.J. McCabe, "Complexity Measure", *IEEE Trans. Soft Eng.*, vol. SE-2, no. 4, pp. 308-320, Dec. 1976.

[5] Halstead, and H. Maurice, "Elements of Software Science", Elsevier North-Holland, New York, 1977.

[6] Henry and Kafura,"Software Structure Metrics Based on Information Flow", *IEEE Trans. Soft Eng.*, vol. SE-7, no. 5, pp. 510-518, Sep. 1981.

[7] Warren Harrison,"An Entropy Based Software Complexity Measure", *IEEE Trans. Soft Eng.*, vol. SE-18, no. 11, pp. 1025-1029, Nov. 1992.

[8] Jingqiu Shao and Yingxu Wang, "A New Measure of Software Complexity based on Cognitive Weights", *Canadian Journal of Electrical and Computer Eng.*, vol.28, no.2, pp. 1333-1338, Apr. 2003.

[9] Khoshgoftaar and Munson, "Applications of a Relative Complexity Metric for Software Project Management ", *Journal of Systems Software*, vol. 12, no. 3, pp. 283-293, Jul. 1990.

[10] Dr. Deok H Nam and Dr. Harpreet Singh," Pattern Recognition using Multivariate-based Fuzzy Inference Rule Reduction on Neuro Fuzzy", *Fuzzy Information Processing Society,* pp. 573-578, Jun 2005.

[11] L. Zadeh "Fuzzy Sets," *Inform. Contr.*, vol. 8, no. 3, pp. 338-353, 1965.

Macam S. Dattathreya received B.E degree in Industrial and Production Engineering from the University of Mysore, India in 1994, and the M.S degree in Computer Engineering from Wayne State University, Detroit, MI, USA in 1999. He has been working in the software engineering area as a software developer, designer, and a lead IT architect. He has worked in IBM global services as a lead IT architect from 1999 to 2009, where seven patent applications were filed on his behalf, and three technical journal articles were published to IBM journals.

Harpreet Singh (M'73–SM'85-SLM'09) received his B.Sc.Engg from Punjabi University, India, in 1963, and his M.S. and Ph.D. from the University of Roorkee, India, in 1966 and 1971 respectively. He taught at the University of Roorke from 1963 to 1981. Since 1981 he has been with Wayne State University, Detroit, MI, where he is now a professor in the department of Electrical and Computer Engineering. He has worked in diversified areas of systems, networks, controls, computers, image processing, fuzzy logic, software engineering, communication network reliability, and VLSI design. Dr. Singh has more than 250 publications in international journals and conferences and has received a number of awards. He has organized a number of national and international conferences, and has supervised 15 Ph.D theses in different areas in electrical and computer engineering.