Architecture-based Software Reliability incorporating Fault Tolerant Machine Learning

Maskura Nafreen, BS, University of Massachusetts Dartmouth, USA
Saikath Bhattacharya, MS, University of Massachusetts Dartmouth, USA
Lance Fiondella, PhD, University of Massachusetts Dartmouth, USA

Key Words: Architecture-based software, software reliability, machine learning, fault-tolerance, correlation

SUMMARY & CONCLUSIONS

With the increased interest to incorporate machine learning into software and systems, methods to characterize the impact of the reliability of machine learning are needed to ensure the reliability of the software and systems in which these algorithms reside. Towards this end, we build upon the architecture-based approach to software reliability modeling, which represents application reliability in terms of the component reliabilities and the probabilistic transitions between the components. Traditional architecture-based software reliability models consider all components to be deterministic software. We therefore extend this modeling approach to the case, where some components represent learning enabled components. Here, the reliability of a machine learning component is interpreted as the accuracy of its decisions, which is a common measure of classification algorithms. Moreover, we allow these machine learning components to be fault-tolerant in the sense that multiple diverse classifier algorithms are trained to guide decisions and the majority decision taken. We demonstrate the utility of the approach to assess the impact of machine learning on software reliability as well as illustrate the concept of reliability growth in machine learning. Finally, we validate past analytical results for a fault tolerant system composed of correlated components with real machine learning algorithms and data, demonstrating the analytical expression's ability to accurately estimate the reliability of the fault tolerant machine learning component and subsequently the architecture-based software within which it resides.

1 INTRODUCTION

Recently, machine learning has enjoyed significant attention as a potential enabler of autonomous systems and society is eager for such capabilities. However, high profile failures of these systems such as the death of a pedestrian by an automated taxi creates a justifiable level of distrust in the maturity of the underlying technology and subsequent concern for autonomous system safety. The reliability engineering community needs methods to quantitatively assess the impact of learning-enabled components on the software and system within which these components reside. In the absence of objective methods to model the reliability of systems incorporating machine learning and assess reliability growth, decision-makers will struggle to deliver dependable and trustworthy autonomous system.

To assess autonomous systems, one must recognize that autonomous systems are software enabled and that learning-enabled components reside within a software architecture. The earliest architecture-based software reliability research was performed by Cheung [1] who modeled the reliability of a software application in terms of the component reliabilities and the probabilistic transitions between the components with a discrete time Markov chain. Gokhale and Trivedi [2] presented a unification framework for architecture-based software reliability and performance modeling methods, identifying gaps where further contributions could be made. Additional modeling studies have demonstrated the applicability of the architecture-based approach to Service-Oriented Architectures [3], extended the architecture-based approach to concurrent applications [4], explicitly considered error propagation and recovery [5] and proposed an efficient branch and bound algorithm to characterize the impact of correlated component failures on the reliability of architecture-based software [6].

In addition to modeling the reliability of architecture-based software reliability, some researchers have proposed methods to conduct sensitivity analysis and optimization. For example, Fiondella and Gokhale [7] generalized the architecture-based model solution method to the analytical case to obtain an algebraic expression in terms of the component reliabilities and transition parameters and presented a frequentist method to quantify the uncertainty in these parameters. Doran et al. [8] subsequently presented a Bayesian method to estimate the component reliability and transition parameters of an architecture-based software reliability model. Multi-objective test resource allocation models for architecture-based software considering reliability and cost [9, 10] have also been proposed as well as multi-objective evolutionary algorithms to identify a

Pareto optimal set of solutions [11] for tradeoffs between reliability and cost constraints.

This paper presents an approach to assess the reliability of software based on its architecture, including fault tolerant machine learning components that perform classification and are also known as ensemble classifiers [12]. We apply an analytical method to quantify the reliability of a fault tolerant component to consider the case where multiple machine learning algorithms are used to enable a single decision. This method quantifies the reliability of fault tolerant machine learning in terms of the accuracy of the individual machine learning algorithms and the correlations between each pair of algorithms. The reliability estimate of the fault tolerant machine learning component is then combined with the architecture-based approach to software reliability assessment in order to estimate application reliability. The approach is demonstrated through examples, including the impact of reliability growth in fault tolerant machine learning on application reliability. The predictive accuracy of the analytical approach is also assessed.

Section II summarizes the architecture-based approach to software reliability modeling. Section III describes an analytical method to estimate the reliability of a majority fault tolerant system from the component reliabilities and correlations between the components and Section IV discusses fault-tolerant machine learning concept. Section V illustrates the approach, while Section VI offers conclusions and directions for future study.

2 ARCHITECTURE-BASED SOFTWARE RELIABILITY MODELING AND ANALYSIS

This section summarizes the composite approach to architecture-based reliability modeling and analysis [1]. Without loss of generality, consider an application composed of n components, where the first and nth components respectively denote the initial and final, where computation begin and end. Such a terminating application operates on demand, and distinguishes consecutive runs according to the inputs and corresponding branching behavior exhibited. A probabilistic control flow graph represents the application architecture, which is characterized by a discrete time Markov chain (DTMC) in order to conduct architecture-based analysis. There is a one-to-one correspondence between the components of the application and the states of the DTMC and the one-step transition probability matrix $\mathbf{P}$ of the DTMC encodes the application's architecture. Entry $p_{n,n}$ of $\mathbf{P}$ is set to 1.0 to denote that the application terminates after component n completes execution. Moreover, the vector $\mathbf{R}_{1\times n}$ denotes the reliabilities of the components, where $r_i \in (0,1)$ is the reliability of component i. The expected system reliability $E[\mathbf{R}]$ may be obtained in the following steps:

(1) Set $p_{n,n} = 0$
(2) Define $\mathbf{D}_{n\times n}$ to be the diagonal matrix with $d_{i,i} = r_i$
(3) Let $\mathbf{Q} = \mathbf{D} \cdot \mathbf{P}$
(4) Compute $\mathbf{S} = (\mathbf{I} - \mathbf{Q})^{-1}$
(5) $E[\mathbf{R}] = s_{1,n} \times r_n$

$s_{1,n}$ in step (5) represents the probability that the final, absorbing state will be reached from the state in which computation commenced. This value times the reliability of the component n, estimates the average application reliability. The above model assumes that both component transitions and component failures are independent. It also assumes that component failure results in application failure. We note that these assumptions are made in state of the art models [2] on architecture-based analysis with the exception of [6], which explicitly models correlation between the failures of the components.

3 COMPONENT-LEVEL MAJORITY FAULT TOLERANCE CONSIDERING CORRELATION

This section describes a method to model component-level majority fault tolerance with a trivariate Bernoulli distribution [13], which explicitly considers the negative impact of correlated failures on reliability. This method can be applied to traditional components within the architecture-based approach as well as components representing fault tolerant machine learning.

Let the vector $\mathbf{r}_k = (r_{k,1}, r_{k,2}, r_{k,3})$ represent the reliability of three diverse but functionally redundant implementations of component k and $\mathbf{\Sigma}$ their correlation matrix, with non-negative entries $\rho_{i,j}$ denoting the correlation between components $r_{k,i}$ and $r_{k,j}$, $i, j \in \{1,2,3\}$ and $i \neq j$. Define $\beta_{i,j} = \exp(\alpha_{i,j})$, where

$$\alpha_{i,j} = \log\left(1 + \rho_{i,j}\sqrt{\frac{(1-r_{k,i})(1-r_{k,j})}{r_{k,i}r_{k,j}}}\right) \quad (1)$$

Given component reliabilities $\mathbf{r}_k$ and correlations $\mathbf{\Sigma}$, the reliability expression for the two-out-of-three good system is [13]

$$R_{2,3} = R_s + \sum_{i<j} p_i p_j \beta_{i,j} - \left(\frac{\prod_{i=1}^{3} p_i \prod_{i<j} \beta_{i,j}}{\beta_{i,j}^{(1)}}\right) \quad (2)$$

where R_s is the reliability expression for the three component series system

$$R_s = \frac{\prod_{i=1}^{3} p_i \prod_{i<j} \beta_{i,j}}{\beta_{i,j}^{(1)}} \quad (3)$$

The term $\beta_{i,j}^{(1)}$ is determined by computing $\alpha_{i,j}$ and identifying the corresponding case in Table 1.

Table 1: Feasible orderings of $\alpha_{i,j}$

Case	Order	k	$\beta_{i,j}^{(1)}$
I	$\alpha_{1,2} < \alpha_{1,3} < \alpha_{2,3} < \alpha_{1,1} < \alpha_{2,2} < \alpha_{3,3}$	3	$\beta_{1,2}$
II	$\alpha_{1,2} < \alpha_{2,3} < \alpha_{1,3} < \alpha_{1,1} < \alpha_{2,2} < \alpha_{3,3}$	3	$\beta_{1,2}$
III	$\alpha_{1,3} < \alpha_{1,2} < \alpha_{2,3} < \alpha_{1,1} < \alpha_{2,2} < \alpha_{3,3}$	2	$\beta_{1,3}$
IV	$\alpha_{1,3} < \alpha_{2,3} < \alpha_{1,2} < \alpha_{1,1} < \alpha_{2,2} < \alpha_{3,3}$	2	$\beta_{1,3}$
V	$\alpha_{2,3} < \alpha_{1,2} < \alpha_{1,3} < \alpha_{1,1} < \alpha_{2,2} < \alpha_{3,3}$	1	$\beta_{2,3}$
VI	$\alpha_{2,3} < \alpha_{1,3} < \alpha_{1,2} < \alpha_{1,1} < \alpha_{2,2} < \alpha_{3,3}$	1	$\beta_{2,3}$
VII	$\alpha_{1,2} < \alpha_{1,3} < \alpha_{1,1} < \alpha_{2,3} < \alpha_{2,2} < \alpha_{3,3}$	3	$\beta_{1,2}$
VIII	$\alpha_{1,3} < \alpha_{1,2} < \alpha_{1,1} < \alpha_{2,3} < \alpha_{2,2} < \alpha_{3,3}$	2	$\beta_{1,3}$

Moreover, the reliability estimate produced by Equation (2) is valid only if the following condition is satisfied.

$$p_k < \frac{\left(\beta_{i,j}^{(1)}\right)^2}{\prod_{i<j}\beta_{i,j}} \tag{4}$$

which may be violated when the correlation is close to the maximum value feasible [14]

$$\rho_{i,j}^{+} = \min\left(\sqrt{\frac{r_{k,i}\overline{r}_{k,j}}{\overline{r}_{k,i}r_{k,j}}}, \sqrt{\frac{\overline{r}_{k,i}r_{k,j}}{r_{k,i}\overline{r}_{k,j}}}\right) \tag{5}$$

4 FAULT-TOLERANT MACHINE LEARNING CLASSIFIER

In machine learning, classification algorithms predict the category to which a new observation belongs based on training data. Here, we consider three diverse classification techniques, including the support vector machine [15], artificial neural network [16], and the naïve Bayes algorithm [17], which are combined to form a majority fault tolerance component within the application architecture. In cases where an autonomous system performs an action based on a classification, incorrect classifications may be interpreted as component unreliability. Thus, a classifier is reliable if it correctly classifies an observation and unreliable otherwise. A fault tolerant classifier therefore is capable of masking a single incorrect classification.

Sources of error in classification include noise, bias, and variance. Bootstrap sampling [18] is an iterative method to improve classifier performance. Given a training dataset, the Bootstrap selects a subset uniformly at random with replacement and a similar procedure is followed to select a subset of the dataset for testing. Each round of testing enables the accuracy of each classifier to be assessed to avoid overfitting.

5 ILLUSTRATIONS

This section first derives an analytical expression for the reliability of the architecture of an autonomous system's software. It then describes the data used to train and test the fault tolerant machine learning component. We then examine the impact of reliability growth in the fault tolerant machine learning component on application reliability. Finally, we assess the correlation between pairs of machine learning methods and compare the empirical reliability and analytical reliability prediction.

5.1 Architecture-based software reliability expression

Figure 1 shows the architecture of a component-based autonomous application and the inter-component transitions.

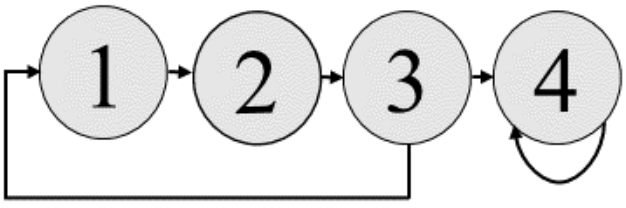

Figure 1 Autonomous system application architecture

Component one, two, and three respectively represent perception, decision-making, and execution, while the fourth component is the mission completion state. Thus, control cycles through components one through three in the sense-act loop. For the sake of illustration, it is assumed that the perception underlying component one is machine learning enabled.

The application architecture shown in Figure 1 maps to the following algebraic transition probability matrix [7].

$$\mathbf{P} = \begin{bmatrix} 0 & p_{1,2} & 0 & 0 \\ 0 & 0 & p_{2,3} & 0 \\ p_{3,1} & 0 & 0 & p_{3,4} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

where entry $p_{4,4}$ has been set to zero according to step one of the architecture-based analysis approach given in Section 3. Step two defines the four by four diagonal matrix with $d_{i,i} = r_i$ such that $\mathbf{D} \cdot \mathbf{P}$ of step three produces

$$Q = \begin{bmatrix} 0 & r_1 p_{1,2} & 0 & 0 \\ 0 & 0 & r_2 p_{2,3} & 0 \\ r_3 p_{3,1} & 0 & 0 & r_3 p_{3,4} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

Step four computes the fundamental matrix $\mathbf{S} = (\mathbf{I} - \mathbf{Q})^{-1}$, while step five multiplies entry $s_{1,4}$ by r_4 to provide the algebraic expression of application reliability in terms of its architecture and component reliabilities

$$E[\boldsymbol{R}] = \frac{p_{12}p_{23}r_1r_2r_3r_4}{1 - p_{12}p_{23}p_{31}r_1r_2r_3} \tag{8}$$

5.2 Data Description

Machine learning algorithms were applied to the NASA Turbofan Engine Degradation Simulation Data Set [19], available from their Prognostics Data Repository. Engines were run to failure. Operation is characterized by sequence of discrete cycles, consisting of 21 sensor readings as well as three operational settings. Four sets of data are included. Each data set contains separate training and testing data, but both possess the same format. Thus, perception in Figure 1 corresponds to the state of health of a vehicle's critical subsystem, namely the engine. However, the architecture can be applied to other applications of machine learning to enable autonomy in the perception, decision-making, and execution loop.

To illustrate reliability growth of the systems within which the fault tolerant machine learning resides, between ten and ninety percent of the training data was used to train a support vector machine, an artificial neural network, and the naïve Bayes algorithm. Fifty percent of the test data was then used to quantify each algorithms accuracy [20]. Both the training and test data were selected using the Bootstrap sampling method. In all cases, the algorithms were trained to recommend maintenance when it was predicted that there were less than 70 cycles of remaining useful life (RUL). Therefore, if the RUL remains above this value until engine failure, the system may experience system failure.

5.3 Impact of fault-tolerant machine learning on reliability of architecture-based software

For the sake of illustration, we assume component one

performs the perception function according to the fault-tolerant machine learning classifier described above. The reliability of components two through four are held constant at $r_2 = 0.93$, $r_3 = 0.95$, and $r_4 = 1.0$ for the sake of exposition.

Figure 2 shows the reliability growth of the system given in Figure 1 and Equation (8) as the training sample size increases the reliability of r_1. To ensure comparability, the individual classifiers were trained and tested with the same subset of the data. Four cases are shown, namely the three cases where the reliability of component one is characterized by a single classifier, including the support vector machine, artificial neural network, or naïve Bayes classifier as well as the case where all three are combined in a majority voting scheme.

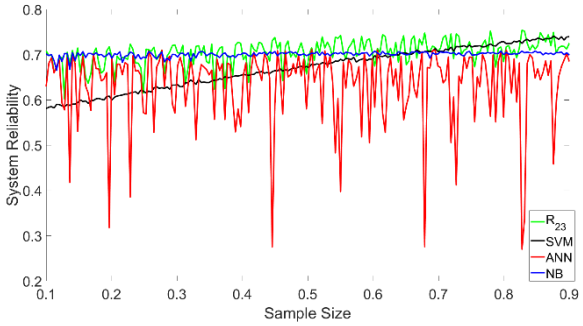

Figure 2 Impact of classifier accuracy on architecture-based software reliability

The system reliability attained by each approach is $E[\boldsymbol{R}]|_{r_1=SVM} = 73.95\%$, $E[\boldsymbol{R}]|_{r_1=ANN} = 66.39\%$, $E[\boldsymbol{R}]|_{r_1=NB} = 69.90\%$, and $E[\boldsymbol{R}]|_{r_1=R_{2,3}} = 68.48\%$. Thus, the support vector machine attained the highest reliability, followed by majority voting because the ANN performs erratically and exhibits correlation failure (inaccurate classification) with SVM and NB, which we examine in greater detail in the next example. While a fault-tolerant classifier that is more reliable than the individual classifiers is desirable, the focus is assess the accuracy of the analytical method described in Section 3.

5.4 Assessment of correlation between classifiers

Figure 3 shows the Pearson correlation coefficient between the classifications of the SVM and NB classifiers throughout the training process as well as the upper bound ($\rho^{+}_{SVM,NB}$) determined from Equation (5).

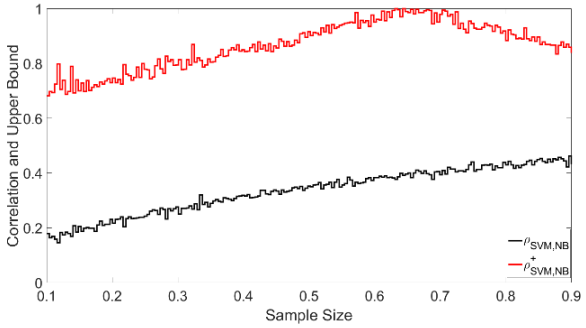

Figure 3 Correlation between pair of classifiers and upper bound

Figure 3 shows that the correlation is positive, which is expected because the classifiers are not totally random and make correct predictions more often than not, so correct predictions lead to positive correlation. Thus, as training progresses and accuracy increases, so does correlation. Moreover, the correlation is less than the upper bound, indicating that application of the analytical expressions in Section 3 are more likely to succeed. A similar analysis of the correlations between the other two pairs of classifiers identified similar trends and determined that the corresponding upper bounds were also satisfied.

5.5 Comparison of analytical and empirical methods

To assess the accuracy of the analytical method presented in Section 3, the empirical reliability (accuracy) and correlation between each pair of machine learning methods as the amount of sample data increased. Equation (2) was then applied to these intermediate estimates of the component reliabilities and correlations. Figure 4 shows the reliability estimate produced by the analytical method and the empirical values produced by the testing procedure described in Section 5.2.

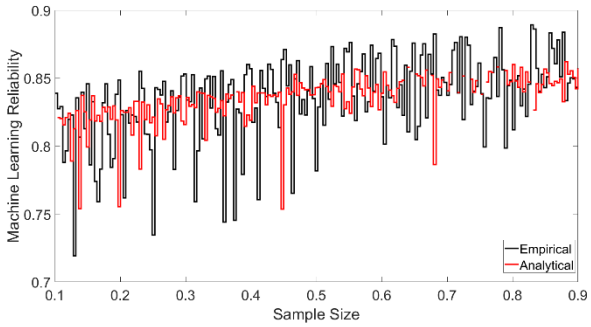

Figure 4 Analytical and empirical reliability assessments of fault tolerant machine learning component

The range of error between these methods was $(-0.09, 0.11)$, which is not trivial. However, this error decreases as sample size increases. The large variations in the reliability of the neural network may also have contributed to this error. The method described in Section 3 failed for 18 of 200 points, primarily in the latter half of the training process between 0.5 and 0.9. Thus, in Figure 4 above and Figures 5 below, the trends are drawn by simply connecting the point values before and after these unsuccessful applications of the analytical method.

To further assess the error between the analytical and empirical assessments in Figure 4, Figure 5 shows the minimum and maximum error of estimates at each point during training. For example, Figure 5 shows that after twenty percent of the training had been conducted, the worst overestimate for any value of sample size between 0.2 and 0.9 was slightly greater than 0.1, whereas the worst overestimate was slightly less than -0.09. However as sampling continued, to 0.5 the worst case over and underestimates by the analytical method decreases to below 0.1 and about -0.05 respectively. While the bounds on the trend were not monotonic, both over and underestimates decreased substantially as the sample size approached ninety percent. These trends are promising, considering that only 200 samples were taken. Thus, after 180 samples, the estimate was

slightly below the horizontal line denoting zero error, indicating that error remained, but was substantially reduced.

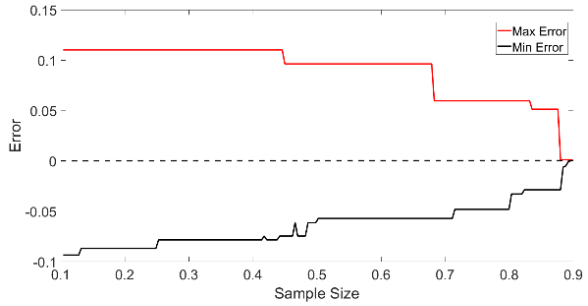

Figure 5 Maximum and minimum error bounds

To further study the difference between the analytical and empirical assessments, Figure 6 shows a histogram of the error between the two methods as well as multiple fits of a normal distribution to data to assess for convergence.

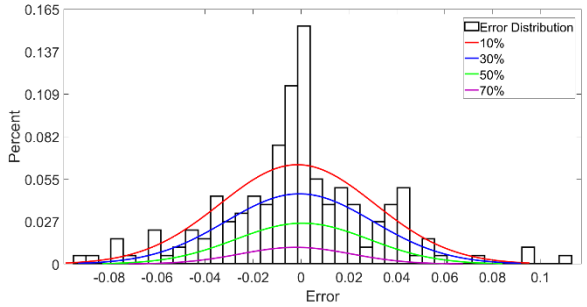

Figure 6 Distribution of error between analytical and empirical reliability assessments

The upper normal distribution was fit to all of the data, including estimates made when between ten and ninety percent of the data was used, whereas reduced sets omit a prefix. Thus, subsets of errors observed from thirty, fifty, and seventy to ninety were also use to fit normal distributions, which exhibit decreasing variance. Figure 6 suggests that the errors were approximately normally distributed, but that the errors are more heavily concentrated about zero, which is desirable.

To complement graphical analysis, formal hypothesis tests were applied. In each case, the null hypothesis H_0 is that the errors between the analytical and empirical reliability estimates are normally distributed, while the alternative H_a is that the errors are not normally distributed. Table 2 states the names, test corresponding test statistics, and p-values.

Table 2: Hypothesis tests for normality

Test Name	Test Statistic	p-value
D'Agostino and Pearson Test	6.87	0.0322
Jarque-Bera Test	11.20	0.0037
Anderson-Darling Test	1.78	0.0004
Kolmogorov-Smirnov Test	1.16	0.1354
K-S Marsaglia Method	1.16	0.1277
Shapiro-Wilk Test	0.97	0.0017

Table 2 indicates that the D'Agostino and Person, Jarque-Bera, Anderson-Darling, and Shapiro-Wilk tests reject the null hypothesis at the 95% level of significance, whereas the Kolmogorov-Smirnov and K-S Marsaglia tests do not reject the null hypothesis at the 95% level of significance. While several tests reject the null hypothesis, this is likely due to the heavy concentration of errors around zero observed in Figure 6.

6 CONCLUSIONS AND FUTURE WORK

This paper presented an architecture-based software reliability model where one or more components can be characterized by fault tolerant machine learning. Moreover, an analytical method to estimate the reliability of a majority system with explicit correlation parameters was applied to the accuracy and correlation data of three machine learning algorithms and the predictions compared to empirical results. The observed prediction errors of the analytical method ranged between -0.09 and 0.11, but these errors decreased as the amount of training data employed increased. We also examined the distribution of the errors and found it was well characterized by a normal distribution, but that the errors were heavily centered around zero, suggesting that the analytical method may be approximately asymptotically unbiased. Thus, the analytical method can be used to complement empirical reliability assessments during architecture design to identify reliability requirements and corresponding training data requirements.

Future research will seek to further develop the modeling methods for application to autonomous system architectures as well as analyze the statistical properties of the analytical method more formally.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant Number (#1749635). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

1. R. Cheung, "A User-Oriented Software Reliability Model," *IEEE Transactions on Software Engineering,* vol. 6, no. 2, pp. 118-125, 1980.
2. S. Gokhale and K. Trivedi, "Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework," *IEEE Transactions on Reliability,* vol. 55, no. 4, pp. 578-590, 2006.
3. V. Cortellessa and V. Grassi, "Reliability Modeling and Analysis of Service-Oriented Architectures," *Test and Analysis of Web Services,* Springer: Berlin, pp. 339-362, 2007.
4. R. Kharboutly, S. Gokhale, and R. Ammar, "Architecture-based Software Reliability Analysis incorporating Concurrency," *International Journal of Reliability, Quality and Safety Engineering* , vol. 14, no. 5, pp. 479-499, 2007.
5. L. Fiondella and S. Gokhale, "Architecture-based software reliability with error propagation and recovery," In Proc. *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, Toronto,

Canada, pp. 38-45, 2013.
6. L. Fiondella, S. Rajasekaran, and S. Gokhale, "Efficient Software Reliability Analysis with Correlated Component Failures," *IEEE Transactions on Reliability,* vol. 62, no. 1, pp. 244-255, 2013.
7. L. Fiondella and S. Gokhale, "Importance Measures for Modular Software with Uncertain Parameters," *Software testing, Verification and Reliability,* vol. 20, no. 1, pp. 63–85, 2010.
8. D. Doran, M. Tran, L. Fiondella, and S. Gokhale, "Architecture-based Reliability Analysis With Uncertain Parameters," In Proc. *International Conference on Software Engineering and Knowledge Engineering*, Miami, FL, pp. 629-634, 2011.
9. L. Fiondella and S. Gokhale, "Optimal Allocation of Testing Effort Considering Software Architecture," *IEEE Transactions on Reliability,* vol. 61, no. 2, pp. 580-589, 2012.
10. H. Okamura and T. Dohi, "Optimizing Testing-Resource Allocation Using Architecture-Based Software Reliability Model," *Journal of Optimization,* 2018.
11. B. Yang, Y. Hu and C. Huang, "An Architecture-Based Multi-Objective Optimization Approach to Testing Resource Allocation," *IEEE Transactions on Reliability,* vol. 64, no. 1, pp. 497-515, 2015.
12. L.I. Kuncheva, C.J. Whitaker, C.A. Shipp, and R.P.W. Duin, "Limits on the majority vote accuracy in classifier fusion," *Pattern Analysis & Applications,* vol. 6, no. 1, p. 22–31, 2003.
13. L. Fiondella and P. Zeephongsekul, "Trivariate Bernoulli Distribution with Application to Software Fault Tolerance," *Annals of Operations Research,* vol. 244, no. 1, pp. 241–255, 2016.
14. R. Prentice, "Binary Regression using an Extended Beta-binomial Distribution, with Discussion of Correlation induced by Covariate Measurement Errors.," *Journal of the American Statistical Association,* vol. 81, no. 394, p. 321–327, 1986.
15. J. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers," *Neural Processing Letters,* vol. 9, no. 3, pp. 293-300, 1999.
16. G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 30, no. 4, pp. 451-462, 2000.
17. I. Rish, "An empirical study of the naive Bayes classifier," In Proc. *International Joint Conferences on Artificial Intelligence: Workshop on Empirical Methods in Artificial Intelligence*, Seattle, WA, vol. 3, no. 22, pp. 41-46, 2001.
18. T. Dietterich, "Ensemble Methods in Machine Learning." In: Multiple Classifier Systems. Lecture Notes in Computer Science," *Springer,* vol. 1857, pp. 1-15, 2000.
19. A. Saxena and K. Goebel, "Phm08 challenge data set," 2008.
20. T. Wang, Jianbo Yu, D. Siegel and J. Lee, "A similarity-based prognostics approach for Remaining Useful Life estimation of engineered systems," In Proc. *International Conference on Prognostics and Health Management*, Denver, CO, pp. 1-6, 2008.

BIOGRAPHIES

Maskura Nafreen
Department of Electrical and Computer Engineering
University of Massachusetts - Dartmouth
285 Old Westport Road
North Dartmouth, MA 02747, USA

e-mail: mnafreen@umassd.edu

Maskura Nafreen is a PhD student in the Department of Electrical and Computer Engineering at the University of Massachusetts Dartmouth (UMassD). She received her BS.c. (2018) in Electrical and Electronics Engineering from Ahsanullah University of Science and Technology in Bangladesh.

Saikath Bhattacharya
Department of Electrical and Computer Engineering
University of Massachusetts - Dartmouth
285 Old Westport Road
North Dartmouth, MA 02747, USA

e-mail: sbhattacharya@umassd.edu

Saikath Bhattacharya is a PhD student in Computer Engineering at UMassD. His research interests include prognostics and health management and tradespace exploration incorporating reliability, availability, maintainability, affordability. He has published over ten peer-reviewed papers on these topics.

Lance Fiondella, PhD
Department of Electrical and Computer Engineering
University of Massachusetts - Dartmouth
285 Old Westport Road
North Dartmouth, MA 02747, USA

e-mail: lfiondella@umassd.edu

Lance Fiondella is an assistant professor in the Department of Electrical & Computer Engineering at UMassD. He received his PhD (2012) in Computer Science & Engineering from the University of Connecticut. Dr. Fiondella's papers have been the recipient of ten conference paper awards, including the 2015 R.A. Evans/P.K. McElroy Award from the Reliability and Maintainability Symposium (RAMS). His research has been funded by the Department of Homeland Security, National Aeronautics and Space Administration, Army Research Laboratory, Naval Air Warfare Center, and National Science Foundation, including a CAREER Award.