Online Optimal Release Time for Non-homogeneous Poisson Process Software Reliability Growth Model

Vidhyashree Nagaraju, MS, University of Massachusetts Dartmouth, USA

Lance Fiondella, PhD, University of Massachusetts Dartmouth, USA

Key Words: Software reliability, online optimal release policy, non-homogeneous Poisson process, maximum likelihood estimation

SUMMARY & CONCLUSIONS

A large number of software reliability growth models have been proposed in the literature. Many of these models have also been the subject of optimization problems, including the optimal release problem in which a decision-maker seeks to minimize cost by balancing the cost of testing with field failures. However, the majority of these optimal release formulations are either unused or untested. In many cases, researchers derive expressions and apply them to the complete set of failure data in order to identify the time at which cost was minimized, but this is clearly unusable, since it is not possible to go back in time to make a release decision. The only other implicit strategy implied by these optimal release formulations is to refit a model every time a failure occurs and to assess if the optimal release time has past or if additional testing should be performed.

To assess these limitations, which prevent the application of software reliability growth models in practice, this paper presents an online approach to software optimal release planning. In this approach, the model is periodically refit and optimal release time updated. We objectively compare the quality of decisions by comparing the ratio of the cost achieved by a release decision and the unknown true optimum, which only becomes known after the complete data is observed. We also consider a more conservative approach, where the release recommendation must be satisfied for more than one observation period. Our results indicate that the optimal release strategy implied by past studies, which would release software after a single observation can be suboptimal with respect to cost attained and that a more conservative approach that defers release until release is recommended in multiple successive observation periods may be more appropriate. The approach can thus complement decision-maker judgement.

1 INTRODUCTION

Software reliability growth modeling (SRGM) [1] is a well-established methodology that enables multiple quantitative inferences about software during the testing process, including optimal release time [2], often to minimize cost subject to a reliability constraint. While past studies have derived "policies" to determine if the optimal release time has passed or how much additional time is required, no previous studies have described practical methods on how to apply these policies during the testing process.

Goel and Okumoto [2] were the first to formulate and solve the optimal release problem in the context of software reliability, considering reliability and cost criteria. Koch and Kubat [3] incorporated a penalty cost for the delay of software release. Ross [4] discussed estimation of error rate and developed a stopping rule to identify software release time considering exponential failure rate. Yamada and Osaki [5-6] modeled cost and reliability as multiple objectives. Leung [7] presented a non-homogeneous Poisson process (NHPP) based optimal release problem to minimize cost of testing and debugging and maximize reliability. Kapur and Garg [8] studied optimal release under a model with imperfect debugging, while Pham [9] considered imperfect debugging with random life cycle duration and penalty cost. The models by Pham and Zhang [10-11] modeled fault removal times, warranty cost, and risk of software failures. Huang and Lyu [12-13] characterized testing efficiency with a generalized logistic testing effort function. Lin et al. [14] derived optimal release policies considering cost and reliability criteria for multiple changepoint models, while Inoue et al. [15] derived optimal release time expressions for single changepoint models incorporating testing effort. Yang et al. [16] proposed a method to manage the risk associated with uncertainty in the expected cost. Xie [17] presented an optimal software release policy under parametric uncertainty, enabling decisions according to a user's risk tolerance.

This paper proposes an online method to guide release decisions, which explicitly distinguishes the true cost and optimal release time from predictions made by software reliability growth models. Unlike past studies, the proposed approach acknowledges that identifying a globally optimal release time is not possible without knowledge of the full failure data set. Instead, the approach aspires to come as close to the true optimum as possible, which can only be assessed after testing is complete. We apply the method with a model and dataset from the literature. Our results indicate that robust optimal release policies are needed to offer practical guidance during testing as failure data becomes available.

The remainder of the paper is organized as follows: Section

2 briefly reviews software reliability growth models. Section 3 discusses parameter estimation methods, including update rules of an Expectation Conditional Maximization algorithm as well as expressions to obtain initial parameter estimates. Section 4 derives cost optimal release policies for NHPP SRGM and proposes an iterative approach to guide optimal release decisions as testing data becomes available. Section 5 illustrates the approach through a series of examples, while Section 6 offers conclusions and directions for future research.

2 *NON-HOMOGENEOUS POISSON PROCESS SOFTWARE RELIABILITY GROWTH MODELS*

This section provides a brief overview of NHPP software reliability growth models and presents the SRGM to which the proposed approach is applied.

2.1 NHPP SRGM

The non-homogeneous Poisson process is a stochastic process [18] that counts the number of events observed as a function of time. In the context of software reliability, the NHPP counts the number of faults detected by time t. This counting process is characterized by a mean value function (MVF) $m(t)$, which can assume a variety of forms. Many MVF can be written

$$m(t) = a \times F(t) \tag{1}$$

where a is interpreted as the expected number of faults that would be discovered if debugging was performed indefinitely and $F(t)$ is the CDF of a continuous probability distribution characterizing the software fault detection process.

The MVF of the Goel-Okumoto model (GO) [19] is

$$m(t) = a(1 - e^{-bt}) \tag{2}$$

where a is interpreted as the number of faults to be detected with infinite testing and b is the fault detection rate.

2.2 Parameter estimation

This section describes methods to estimate the parameters of a software reliability model, including maximum likelihood estimation and the expectation conditional maximization algorithm as well as a method to estimate initial parameter values.

2.2.1 Maximum Likelihood Estimation

Maximum likelihood estimation maximizes the likelihood function, also known as the joint distribution of the failure data. Commonly, the logarithm of the likelihood function is maximized because taking the logarithm of the likelihood function enables simplification and the monotonicity of the logarithm ensures that the maximum of the log-likelihood function is equivalent to maximization of the likelihood function.

Failure time data consists of a vector of individual failure times $\mathbf{T} = \langle t_1, t_2, \dots, t_n \rangle$ with density function $f(t_i; \Theta)$. The log-likelihood function of a failure times data set is

$$LL(t_i; \Theta) = -m(t_n) + \sum_{i=1}^{n} \log\big(\lambda(t_i)\big) \tag{3}$$

where $\lambda(t) = \frac{dm(t)}{dt}$ is the instantaneous failure rate.

2.2.2 Expectation Conditional Maximization (ECM) Algorithm

We employ the Expectation Conditional Maximization [20] algorithm to identify the maximum likelihood estimates of the model considered in this study. The ECM algorithm [21] simplifies computation by dividing a single M-step of the EM algorithm [22] into p conditional-maximization (CM) steps, where p denotes the number of model parameters. Unlike the EM algorithm which solves a system of simultaneous equations as a single p-dimensional M-step, the CM steps of the ECM algorithm update only one parameter at a time holding all others constant and thus reduces the maximum likelihood estimation process to a sequence of p one-dimensional problems.

The CM-steps of the GO model parameters are obtained by first substituting the MVF of the GO model in Equation (2) into the log-likelihood function given in Equation (3) and taking the partial derivative with respect to a, the maximum likelihood estimate of parameter a is

$$\hat{a} = \frac{n}{1 - e^{-bt_n}} \tag{4}$$

Substituting Equation (4) into Equation (3) produces the reduced log-likelihood (RLL) function, with only one unknown parameter b, which can be estimated with a single application of a numerical root finding algorithm. Thus, in cases such as this, where the RLL contains only one parameter, the ECM algorithm reduces to a simple root finding problem, requiring only a single iteration. Thus, in this case, the CM-step for parameter b'' obtained by differentiating the RLL with respect to b is identical to the traditional maximum likelihood estimate

$$b'' = \frac{n}{\sum_{i=1}^{n} t_i - \dfrac{n\, t_n}{1 - e^{b'' t_n}}} \tag{5}$$

Parameter b'' in Equation (5) must be solved numerically since it does not possess closed form solution.

2.2.3 Initial Parameter Estimation

For a mean value function of the form $m(t) = a \times F(t)$, an initial estimate of the number of faults (a) is simply the observed number of faults (n), while the remaining initial parameter estimates can be determined by maximizing the log-likelihood function of the probability density function $f(\cdot; \Theta) = 0$ and solving to obtain closed-form expressions for these additional parameters.

By the first-order optimality condition, initial estimates of parameter a and the additional parameters of the probability distribution function $f(t; \Theta)$ are given by [23]

$$a^{(0)} = N \tag{6}$$

and

$$\Theta^{(0)} := \sum_{i=1}^{N} \frac{\partial}{\partial \Theta} \log\big(f(t_i; \Theta)\big) = \mathbf{0} \tag{7}$$

In practice, N is replaced with n in Equation (6) when deriving initial parameter estimates.

The initial estimate of the scale parameter of the GO

SRGM obtained from Equations (2) and (7) is

$$b^{(0)} = \frac{n}{\sum_{i=1}^{n} t_i} \tag{8}$$

3 OPTIMAL RELEASE POLICY BASED ON COST CRITERIA

During software testing, one challenge is to identify the optimal time to release the software. However, there is significant cost associated with development, testing, and debugging to ensure the functionality is implemented and is reliable. Optimal release planning provides a quantitative strategy to assess the trade-offs between reliability, testing time, and life cycle cost. This section presents a cost model and derives the optimal release time. Several researchers have proposed cost models in previous research [15, 24-25]. Let T be the length of the software lifecycle and t the time of software release, then the estimated cost to release the software under model x [26-27] is

$$c_x(t) = c_1 m_x(t) + c_2\big(m_x(T) - m_x(t)\big) + c_3 t \tag{9}$$

where $m_x(t)$ is the MVF of model x, c_1 is the cost of removing a fault during testing, c_2 is the cost of removing a fault after release ($c_2 > c_1$), and c_3 is the cost of testing per unit time. The ultimate goal is to identify $t < T$ that minimizes the *true cost*

$$c(t) = c_1 N(t) + c_2\big(N(T) - N(t)\big) + c_3 t \tag{10}$$

where $N(t)$ is the actual number of faults discovered by time t.

However, the true optimal release time must be based on the estimate under model x, denoted t_x^*, which is computed by solving

$$\frac{\partial c_x(t)}{\partial t} = 0 \tag{11}$$

The optimal release time of the GO NHPP SRGM is [2]

$$t^* = \frac{1}{b}\log\left(\frac{ab(c_2 - c_1)}{c_3}\right) \tag{12}$$

3.1 Online optimal release procedure

This section presents an online optimal approach to guide the release decision process. Past studies [17] typically perform the following three steps to estimate the optimal release time:

- (S.1) Fit a model to the complete failure data to obtain parameter estimates.
- (S.2) Compute the optimal release time, t^*, by substituting the MLEs obtained in step (S.1) into Equation (9).
- (S.3) Plot the cost in Equation (9) as a function of t according to the MLEs obtained in step (S.1) to illustrate the trend and identify the minimum.

The traditional approach has a major disadvantage because many times the software release is recommended before the end of testing, which is computed only after the complete data is collected. In practical situations, it is not possible to go back in time to release the software, which limits the usability of this approach in practice. The only other alternative implied by the literature Is to use Equation (12) and release the software after the present time t exceeds t^*.

The online optimal release procedure developed through the illustrations addresses the limitation associated with the traditional approaches by periodically estimating optimal release as data becomes available. The enhanced approach exhibits some error because it is impossible to precisely know the future fault detection process.

4 ILLUSTRATIONS

This section demonstrates the need for a robust online optimal release policy. The first example illustrates the traditional method to estimate optimal release time from complete data. The second example illustrates the online optimal release procedure implied by past studies, while the third example illustrates the potential for substantial improvement over existing and implied methods. The examples are provided in the context of the GO model applied to the CSR1 dataset [28].

4.1 Optimal release time considering complete data

This section briefly reviews the most basic method employed to identify the cost optimal release time when all failure data is available. It then introduces the notion of true cost as a method to assess the effectiveness of any retroactive or online approach.

Figure 1 shows the estimated cost of release time determined by the GO model when applied with the entire CSR1 data set [28], with cost parameters $c_1 = \$100$, $c_2 = \$15{,}000$, $c_3 = \$20$ and the software lifecycle, $T = 200{,}000$ for the sake of illustration.

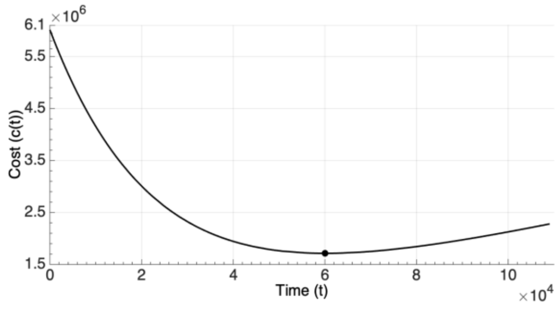

Figure 1: Impact of release time on cost considering GO for CSR1

When the models are fit to the entire data set, the optimal release time is $t_{GO}^* = 60{,}069$ with a corresponding estimated optimal cost denoted by the dot at the minimum in Figure 1 at $c_{GO}(t_{GO}^*) = 1.71$ million. However, there is no clear indication how accurate this release time and cost actually are.

To connect theory with practice, Figure 2 plots the true cost for the CSR1 data set [28], where Equation (10) has been employed to apply an expense of c_1 for each fault discovered before release and a penalty of c_2 for each fault not discovered prior to the release time. Figure 2 exhibits a sharp drop at each fault discovery time t_i because this corresponds to one less post release failure, which would incur a higher cost. Similarly, periods of testing time with no faults exhibit a slow increase corresponding to the cost of testing per unit time c_3. The true optimal release time denoted by the dot in Figure 2 and corresponding cost are $t^* = 79{,}397$ and $c = 1.92$ million, whereas the cost incurred at the time recommended by the GO model shown in Figure 1 incur cost of 1.986 million, which is 1.039 times higher than the true minimal cost. This observation indicates that even if it were possible to utilize the entire failure history and then go back in time to make a decision, the optimal

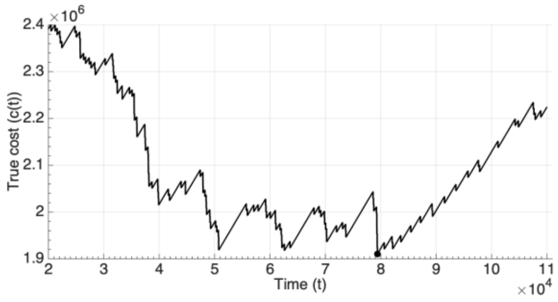

Figure 2: True cost assuming n faults

release time recommended by this model would exhibit approximately four percent inefficiency relative to the true optimum.

For the sake of concreteness, we have assumed that the number of faults to be observed is the number of faults observed by time t_n. The authors acknowledge that it is unlikely that only the $n = 397$ faults observed by the end of testing at time t_n were present in the software. Complete data sets that contain all failures throughout the operational lifetime T can be used in place of data sets from the research literature such as this one, which lack a complete history. Thus, while this assumption may not hold for the data considered here, it serves the purpose of establishing a model independent approach to objectively evaluate the effectiveness of alternative model predictions.

4.2 Online estimation of release time

In order to develop a practical online approach, Figure 3 shows the release time determined from the GO model according to the failure data observed up until time t as well as the line of unit slope. Thus, Figure 3 compares the present time (linearly increasing curve) to the release time estimated from progressively larger prefixes of the CSR1 dataset fit to the GO model, which is first applied when approximately 8% of the total testing time has passed or 8,500 time units. The model is applied updated each additional 500 time units and the optimal release recomputed.
Therefore, release times above the line of unit slope indicate that the release time recommended by the model is greater than the present time and should therefore be subject to additional testing, whereas times below this line suggest that the optimal release time has already passed. Moreover, the farther a model's recommended release time falls below the present time suggests that the software could be released with greater confidence. As time progresses, it can be seen that the release times of the GO model in Equation (12) first cross below the line at $t^*_{GO} = 30{,}500$ with corresponding true costs $c(t^*_{GO}) = \$2.32$ million computed using Equation (10). The ratio between the true costs achieved at the recommended time and the true minimum cost

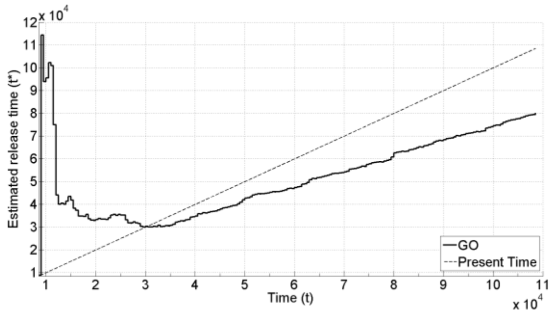

Figure 3: Estimated release time for GO model on CSR1 data

is 1.208.

4.3 Sensitivity of cost ratio to model recommendation

A logical extension to the optimal release decision implied by Equation (12) is to require that the model recommend release for more than one successive time interval.

Figure 4 examines how this approach fares by plotting the ratio of the actual and optimal cost (1.91 million), starting from the first time ($t^*_{GO} = 30{,}500$) at which release was recommended.

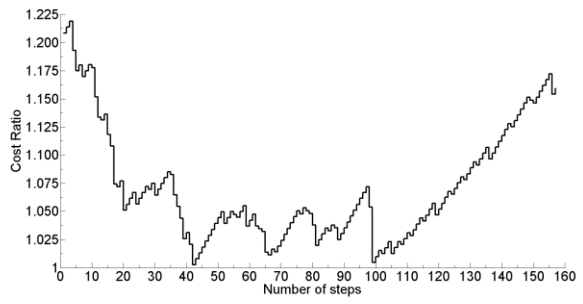

Figure 4: Sensitivity of cost ratio to model release recommendation

Figure 4 indicates that the cost ratio continues to decrease below 1.208, which was recommended by Equation (12) at $t^*_{GO} = 30{,}500$. Thus, if the software is not released the very first time Equation (12) is satisfied, then the cost ratio is nearly minimized at the 42^{nd} time step or $t = 51{,}000$ time units, the cost ratio decreases to 1.00269. Similarly, deferring release for just 20 time steps reduces the cost ratio to approximately 1.05. These results indicate that optimal release policies implied by dozens of past studies are potentially far from optimal and that more robust online release methods are needed to make this application of software reliability growth models feasible in practice.

5 CONCLUSIONS AND FUTURE WORK

This paper presents a procedure for online estimation of optimal release time. The examples illustrate the potential for substantial improvement over approaches implied by past research. The data set considered demonstrated that a method utilizing the complete data is suboptimal as are methods that release software the first time the optimal release expression is satisfied. Examination of the ratio attained between actual and optimal costs indicated that a robust online approach that defers release may further reduce cost.

Future research requires developing a model selection strategy based on goodness-of-fit measures, release recommendations considering cost and reliability, and accelerated algorithms to improve performance of the model fitting while subsetting the data. However, more critical reflection on the practical role of NHPP SRGM is needed to determine if failure time event statistics is sufficient to accurately predict future fault detection, software reliability growth, and optimal release. Software engineering is a complex process and the underlying activities are non-trivial. Greater dialog among software engineers, testers, and modelers is likely needed to address the optimal release problem in a truly satisfactory manner.

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. M. Lyu, Ed., *Handbook of Software Reliability Engineering*. NY: McGraw-Hill, 1996.
2. K. Okumoto and A. Goel, "Optimum release time for software systems based on reliability and cost criteria," *Journal of Systems and Software*, vol. 1, no. 4, pp. 315–318, 1980.
3. H. Koch and P. Kubat, "Optimal release time of computer software," *IEEE Transactions on Software Engineering*, no. 3, pp. 323–327, 1983.
4. S. Ross, "Software reliability: the stopping rule problem," *IEEE Transactions on Software Engineering*, no. 12, pp. 1472–1476, 1985.
5. S. Yamada and S. Osaki, "Optimal software release policies with simultaneous cost and reliability requirements," *European Journal of Operational Research*, vol. 31, no. 1, pp. 46–51, 1987.
6. S. Yamada and S. Osaki, "Cost-reliability optimal release policies for software systems," *IEEE Transactions on Reliability*, vol. 34, no. 5, pp. 422–424, 1985.
7. Y. Leung, "Optimum software release time with a given cost budget," *Journal of Systems and Software*, vol. 17, no. 3, pp. 233–242, 1992.
8. P. Kapur and R. Garg, "Optimal software release policies for software reliability growth models under imperfect debugging," *RAIRO-Operations Research*, vol. 24, no. 3, pp. 295–305, 1990.
9. H. Pham, "A software cost model with imperfect debugging, random life cycle and penalty cost," *International Journal of Systems Science*, vol. 27, no. 5, pp. 455–463, 1996.
10. H. Pham and X. Zhang, "A software cost model with warranty and risk costs," *IEEE Transactions on Computers*, vol. 48, no. 1, pp. 71–75, 1999.
11. H. Pham, "Software reliability and cost models: Perspectives, comparison, and practice," *European Journal of Operational Research*, vol. 149, no. 3, pp. 475–489, 2003.
12. C. Huang, "Cost-reliability-optimal release policy for software reliability models incorporating improvements in testing efficiency," *Journal of Systems and Software*, vol. 77, no. 2, pp. 139–155, 2005.
13. C. Huang and, M. Lyu, "Optimal release time for software systems considering cost, testing-effort, and test efficiency," *IEEE Transactions on Reliability*, vol. 54, no. 4, pp. 583–591, 2005.
14. C. Lin, C. Huang, and J. Chang, "Integrating generalized Weibull-type testing-effort function and multiple change-points into software reliability growth models," in *IEEE Asia-Pacific Software Engineering Conference*, 2005.
15. S. Inoue, Y. Nakagawa, and S. Yamada, "Optimal software shipping time estimation based on a change-point hazard rate model," *International Journal of Reliability, Quality and Safety Engineering*, vol. 21, no. 2, 2014.
16. B. Yang, H. Hu, and L. Jia, "A study of uncertainty in software cost and its impact on optimal software release time," *IEEE Transactions on Software Engineering*, vol. 34, no. 6, pp. 813–825, 2008.
17. M. Xie, X. Li, and S. Ng, "Risk-based software release policy under parameter uncertainty," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 225, no. 1, pp. 42–49, 2011.
18. S. Ross, *Introduction to Probability Models*, 8th ed. New York, NY: Academic Press, 2003.
19. A. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.
20. V. Nagaraju, L. Fiondella, P. Zeephongsekul, C. Jayasinghe, and T. Wandji, "Performance optimized expectation conditional maximization algorithms for nonhomogeneous Poisson process software reliability models," *IEEE Transactions on Reliability*, vol. 66, no. 3, pp. 722–734, 2017.
21. D. Rubin and R. Little, "Statistical analysis with missing data," *Hoboken, NJ: John Wiley & Sons*, 2002.
22. A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B*, vol. 39, no. 1, pp. 1–38, 1977.
23. H. Okamura, Y. Watanabe, and T. Dohi, "An iterative scheme for maximum likelihood estimation in software reliability modeling," in *International Symposium on Software Reliability Engineering*, nov 2003, pp. 246–256.
24. S. Inoue and S. Yamada, "Optimal software release policy with change- point," *in IEEE International Conference on Industrial Engineering and Engineering Management*, 2008, pp. 531–535.
25. S. Chatterjee, S. Nigam, J. Singh, and L. Upadhyaya, "Effect of change point and imperfect debugging in

software reliability and its optimal release policy," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 18, no. 5, pp. 539–551, 2012.

26. M. Zhao and M. Xie, "Robustness of optimum software release policies," in *IEEE International Symposium on Software Reliability Engineering*, 1993, pp. 218–225.
27. X. Li, M. Xie, and S. Ng, "Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points," *Applied Mathematical Modelling*, vol. 34, no. 11, pp. 3560–3570, 2010.
28. M. Lyu, "Handbook of Software Reliability Engineering: Data directory," *http://www.cse.cuhk.edu.hk/lyu/book/reliability/*, 2005, [Online; accessed 23-May-2005].

BIOGRAPHIES

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

e-mail: vnagaraju@umassd.edu

Vidhyashree Nagaraju is a PhD student in the Department of Electrical and Computer Engineering at the University of Massachusetts Dartmouth (UMassD), where she received her MS (2015) in Computer Engineering. She received her BE (2011) in Electronics and Communication Engineering from Visvesvaraya Technological University in India.

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.