The Covariate Software Failure and Reliability Assessment Tool (C-SFRAT)

[1]Jacob Aubertine, [2]Vidhyashree Nagaraju, and [1]Lance Fiondella
[1]Department of Electrical and Computer Engineering
University of Massachusetts, Dartmouth, MA, 02747 USA
[2]Tandy School of Computer Science
University of Tulsa, Tulsa, OK, 74107 USA
Email: {jaubertine, lfiondella}@umassd.edu, vidhyashree-nagaraju@utulsa.edu

***Abstract*—Covariate software reliability models characterize defect discovery as a function of test activities and related software metrics. These models also enable more detailed test activity allocation problems suitable for process improvement. However, the mathematical and algorithmic knowledge required to apply these models deters widespread adoption by software practitioners. This paper presents the C-SFRAT (Covariate Software Failure and Reliability Assessment Tool), a free and open source application to promote the adoption of covariate software reliability models. The tool is extensible, allowing for the contributions of new hazard functions, goodness-of-fit measures, and optimization problems. The steps to add a new hazard function are described. Application of the C-SFRAT to two data sets from the literature indicates that, in some cases, newly incorporated hazard functions perform best.**

***Keywords*—Software reliability, software reliability growth model, covariate software reliability growth model, test activity allocation problem, C-SFRAT**

I. Introduction

Non-homogeneous Poisson process (NHPP) software reliability growth models (SRGM) [1] were proposed over 40 years ago to quantify the improvement in software achieved during testing with defect discovery and resolution. More recently, covariate models [2] have emerged as an attractive alternative to NHPP SRGM because they explicitly link test activities to model parameters, enabling the assessment of alternative activities and process improvement. Inferences based on covariate models [3] have also been developed to enhance the utility of these models. NHPP models overemphasized progressively more complex mathematical forms. Without a structured environment to balance focus on models and inferences, covariate models risk similar distrust from the practitioner community.

Previous tools implementing NHPP SRGM, include (CASRE) [4] and the Software Failure and Reliability Assessment Tool (SFRAT) [5]. Existing covariate tools [6] are not explicitly open source or were implemented on spreadsheets [7] without a standalone graphical user interface. Moreover, these past covariate tools do not implement optimization problems to guide ongoing software testing, limiting their utility.

This paper presents the Covariate Software Failure and Reliability Assessment Tool (C-SFRAT), which implements the model presented in Nagaraju et al. [3] as well as the model selection and test activity allocation problems introduced there. The C-SFRAT is open source, enabling the addition of new hazard functions, goodness-of-fit measures, and optimization problems. To simplify the inclusion of new hazard functions, both numerical and symbolic algebra are employed within model fitting procedures. The C-SFRAT, therefore, encourages model extensions to improve predictive capabilities as well as inferences of interest to software practitioners.

This paper is organized as follows: Section II summarizes the current functionality of the tool and provides a brief outline of the software architecture. Section III reviews the Discrete Cox Proportional Hazard NHPP SRGM. Section IV presents the hazard functions implemented in the tool. Section V explains how to contribute a hazard function to the C-SFRAT. Section VI describes how maximum likelihood estimation is performed in the C-SFRAT. Goodness-of-fit measures are described in Section VII. Section VIII provides illustrative examples on two data sets from the literature. Section IX concludes and identifies future research directions.

II. Software Architecture

The C-SFRAT has been implemented in Python 3, with a graphical user interface in PyQt5. The source code is available from https://github.com/LanceFiondella/C-SFRAT. Example data sets [8] have been prepared in the C-SFRAT input format and can be obtained from https://lfiondella.sites.umassd.edu/research/software-reliability.

The primary functions of the C-SFRAT include:

- Displaying model fit and failure intensity plots of selected hazard function and covariate combinations
- Prediction of future failures and failure intensity based on a specified testing activity profile
- Comparison of fitted models based on information theoretic and predictive goodness-of-fit measures with user-defined weighting
- Recommendations for test activity allocation to maximize defect discovery within a specified budget or minimize the total testing resources required to discover a specified number of defects.

Figure 1 shows the C-SFRAT architecture.
To start the application, a user executes the file *main.py*, located in the root directory of the project. The *core* directory contains mathematical functions used for model fitting,

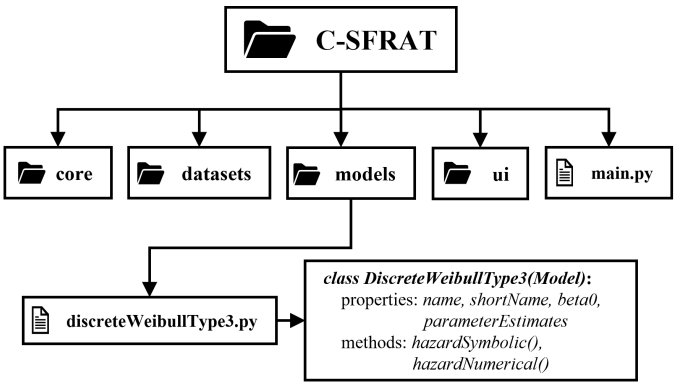

Fig. 1: C-SFRAT architecture

goodness-of-fit calculations, effort allocation, and prediction. The *datasets* directory contains example data sets. The files in the *ui* directory define the layout of the graphical user interface. Hazard functions are defined in the *models* directory.

III. Discrete Cox Proportional Hazard NHPP SRGM

This section describes the discrete Cox proportional hazard NHPP SRGM [3] implemented in the C-SFRAT.

The mean value function describes the mean number of defects detected through the n^{th} interval, denoted by

$$H_{n;\omega,\boldsymbol{\theta},\boldsymbol{\beta}} = \omega \sum_{i=1}^{n} p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}} \tag{1}$$

where $\omega > 0$ is the number of defects that can be discovered with indefinite testing.

The discrete Cox proportional hazards model characterizes the probability that a defect is discovered in interval i, given that it was not discovered in the first $(i-1)$ intervals

$$p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}} = \left(1-(1-h^0_{i;\boldsymbol{\theta}})^{g(\mathbf{x_i};\boldsymbol{\beta})}\right) \prod_{k=1}^{i-1}(1-h^0_{k;\boldsymbol{\theta}})^{g(\mathbf{x_k};\boldsymbol{\beta})} \tag{2}$$

where $h^0_{i;\boldsymbol{\theta}}$ is the baseline hazard function possessing parameters $\boldsymbol{\theta}$,

$$g(\mathbf{x_i};\boldsymbol{\beta}) = \exp(\beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}), \tag{3}$$

the vector $\mathbf{x_i} = (x_{i1}, x_{i2}, \ldots, x_{ij})$, $i = 1, 2, \ldots, n$ denotes the amount of effort dedicated to each of the j software test activities in the i^{th} testing interval, and $\boldsymbol{\beta}$ are the parameters corresponding to each of the test activities.

IV. Hazard Functions

This section presents the eight alternative baseline hazard functions that are currently implemented in the C-SFRAT. The first three were originally employed in the covariate software reliability model of Shibata et al. [8]. The remaining five are taken from the survey by Bracquemond and Gaudoin [9].

1) Geometric (GM) [8]:

$$h^0_{i;\boldsymbol{\theta}} = b \tag{4}$$

where $b \in (0,1)$ is the probability of detecting a fault.

2) Negative binomial of order two (NB2) [8]:

$$h^0_{i;\boldsymbol{\theta}} = \frac{ib^2}{1+b(i-1)} \tag{5}$$

where $b \in (0,1)$ and 2 indicates the order.

3) Discrete Weibull of order two (DW2) [8]:

$$h^0_{i;\boldsymbol{\theta}} = 1 - b^{i^2-(i-1)^2} \tag{6}$$

where $b \in (0,1)$ and 2 indicates the order.

4) Type III discrete Weibull (DW3) [10]:

$$h^0_{i;\boldsymbol{\theta}} = 1 - e^{-ci^b} \tag{7}$$

where $c > 0$ is the scale parameter and b is the shape parameter. For $b > 0$ ($b < 0$), the failure rate is increasing (decreasing) and setting $b = 0$ reduces to geometric.

5) "S" distribution (S) [11]:

$$h^0_{i;\boldsymbol{\theta}} = p(1-\pi^i) \tag{8}$$

where $p \in (0,1)$ is the probability of defect removal and $\pi \in (0,1)$ is the probability of a defect eluding detection in the first testing interval.

6) Truncated logistic (TL) [12], [13]:

$$h^0_{i;\boldsymbol{\theta}} = \frac{1-e^{-1/d}}{1+e^{-\frac{i-c}{d}}} \tag{9}$$

which truncates the logistic distribution to $i > c$ and $d > 0$.

7) IFR Salvia and Bollinger (IFR SB) [14]:

$$h^0_{i;\boldsymbol{\theta}} = 1 - \frac{c}{i} \tag{10}$$

where $c \in [0,1]$ and the distribution exhibits an increasing failure rate.

8) IFR generalized Salvia and Bollinger (IFRGSB) [10]:

$$h^0_{i;\boldsymbol{\theta}} = 1 - \frac{c}{(i-1)\alpha+1} \tag{11}$$

which introduces parameter $\alpha > 0$ to Equation (10). Setting $\alpha = 1$ reduces to Equation (10), while setting $\alpha = 0$ simplifies to the geometric.

V. Contributing a Hazard Function

This section describes how to contribute a hazard function to the C-SFRAT, using the Type III discrete Weibull as an example. Model fitting is performed using the symbolic methods contained in the *SymEngine* package, while the numerical methods are used to evaluate the fitted model after the model parameters are identified.

After downloading the source code of the project from GitHub, the contributor works in the *models* folder shown in Figure 1. Each file in the *models* folder defines a unique hazard function. To add a hazard function, create a new Python file (.py) within the *models* folder. For example, the complete contents of the type III discrete Weibull file *discreteWeibullType3.py* implementing Equation (7) is shown below.

```
import math
import symengine
from core.model import Model

class DiscreteWeibullType3(Model):
    name = "Type III Discrete Weibull"
    shortName = "DW3"
    beta0 = 0.01
    parameterEstimates = (0.1, 0.5)

    def hazardSymbolic(self, i, args):
        f = 1 - symengine.exp(-args[0] *
            i**args[1])
        return f

    def hazardNumerical(self, i, args):
        f = 1 - math.exp(-args[0] * i**args[1])
        return f
```

The contributor must import the *Model* class from *core/model.py*, which contains general definitions for the log-likelihood and mean value functions that can accept any hazard function with any number of covariates. Therefore, the only mathematical function that must be specified by the contributor is the hazard function itself.

To contribute a hazard function, a class must be defined that inherits from the base *Model* class. In the example implementing the type III discrete Weibull hazard function above, we name the class *DiscreteWeibullType3*. The *name* and *shortName* properties are required. They are used to display the hazard function's name in the graphical user interface. The *name* property is displayed on Tab 1 where the user selects the hazard functions to perform model fitting with and should be the full name of the model as a string. The *shortName* property, shown in tables and plot labels, is a string defining an abbreviated name for the hazard function. In the example, the *name* property is "Type III Discrete Weibull", the full name of the hazard function and the *shortName* is "DW3".

The *beta0* and *parameterEstimates* properties must be defined to provide initial estimates for model fitting. The *beta0* property corresponds to the test activity parameters $(\beta_1, \beta_2, \ldots, \beta_p)$ and should be defined as float. All elements of the vector $\boldsymbol{\beta}$ use *beta0* as an initial estimate. Based on our experiments, *beta0* is set to 0.01 for the example. The *parameterEstimates* property is a tuple containing estimates for the hazard function parameter values as floats. If a hazard function possesses multiple parameters the contributor may decide the order of the parameters within the tuple, as long as this order is used consistently throughout the hazard function methods. For the example, parameters c and b are the first and second element of the tuple with values 0.1 and 0.5 respectively. To achieve the best outcome, an individual contributing a new hazard function should experiment with alternative initial estimates to determine values that exhibit the best combination of performance and consistency of convergence to the maximum likelihood estimate.

As noted above, the hazard function must be defined symbolically and numerically. The symbolic definition enables generalization to any number of covariates and symbolic differentiation prior to model fitting. If the symbolic definition of the hazard function requires a mathematical function beyond arithmetic, such as an exponential or logarithm, a method from the *SymEngine* package must be used. In the DW3 example, exponentiation is performed with *symengine.exp()* rather than *math.exp()*. If a hazard function does not contain any advanced mathematical functions, such as GM (Equation 4), then the *hazardSymbolic* and *hazardNumerical* methods will be the same. The *hazardNumerical* method is used to compute numerical values for goodness of fit measures and graphical plots of fitted models.

When starting, the *models/__init__.py* file iterates over all .py files in the *models* folder. As long as a hazard function class inherits from the *Model* class, the hazard function will be displayed in the tool. The tool will not start if any of the properties or methods are not defined.

VI. Maximum Likelihood Estimation

The C-SFRAT estimates the hazard function and other model parameters according to the maximum likelihood estimation. The likelihood function is the joint distribution of the covariate data. The general form of the log-likelihood function implemented in the C-SFRAT is

$$\begin{aligned} LL(\omega, \boldsymbol{\theta}, \boldsymbol{\beta}) &= -\omega \sum_{i=1}^{n} p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}} + \sum_{i=1}^{n} y_i \ln(\omega) \\ &\quad + \sum_{i=1}^{n} y_i \ln(p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}}) - \sum_{i=1}^{n} \ln(y_i!) \end{aligned} \tag{12}$$

where y_i denotes the number of software defects discovered in the i^{th} interval.

The log-likelihood function specified in Equation (12) is reduced from ν to $(\nu - 1)$ parameters by differentiating the log-likelihood function with respect to ω, equating the result to zero, and solving for ω to produce

$$\widehat{\omega} = \frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}}}, \tag{13}$$

Substituting Equation (13) into Equation (12) produces the reduced log-likelihood (RLL) function

$$RLL(\boldsymbol{\theta},\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i + \ln\left(\frac{\sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}}}\right)\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} y_i \ln(p_{i,\mathbf{x_i};\boldsymbol{\theta},\boldsymbol{\beta}}) - \sum_{i=1}^{n}\ln(y_i!) \quad (14)$$

The maximum likelihood estimates are obtained by solving the system of equations $\frac{\partial RLL}{\partial \boldsymbol{\theta}} = \mathbf{0}$ and $\frac{\partial LL}{\partial \boldsymbol{\beta}} = \mathbf{0}$.

C-SFRAT employs the *SymEngine* package to perform maximum likelihood estimation with the symbolic implementation of the reduced log-likelihood function, allowing symbolic differentiation with respect to each parameter. The partial derivatives are converted to lambda functions by the *SymEngine lambdify* method for evaluation by a numerical optimization method. The resulting system of equations is solved using the *root* method from the *SciPy optimize* package, with initial parameter estimates *beta0* and *parameterEstimates*.

VII. Goodness of fit measures

This section summarizes goodness of fit measures to assess how well a model characterizes a data set, enabling comparison of alternative models implemented in the C-SFRAT.

1) Akaike Information Criterion: The AIC quantifies the tradeoff between model precision and complexity. The AIC of model i is a function of the maximized log-likelihood and the number of model parameters (ν).

$$AIC_i = 2\nu - 2LL(\mathbf{x_i};\widehat{\omega},\widehat{\boldsymbol{\theta}},\widehat{\boldsymbol{\beta}}) \quad (15)$$

Without loss of generality, model j preserves information better than model i and is preferred with statistical significance if $AIC_{i,j} = AIC_i - AIC_j > 2.0$.

2) Bayesian Information Criterion: The BIC of model i is a function of the maximized log-likelihood, number of model parameters ν, and sample size n.

$$BIC_i = -2LL(\mathbf{x_i};\widehat{\omega},\widehat{\boldsymbol{\theta}},\widehat{\boldsymbol{\beta}}) + \nu\log(n) \quad (16)$$

3) Sum of Squares Error (SSE): The sum of squares error, also known as the residual sum of squares, is

$$SSE = \sum_{i=1}^{n}(\widehat{H}_{i;\omega,\boldsymbol{\theta},\boldsymbol{\beta}} - Y_i)^2 \quad (17)$$

where $Y_i = \sum_{j=1}^{i} y_j$ is the cumulative number of defects discovered in the first i time intervals.

4) Predictive Sum of Squares Error (PSSE): PSSE compares the predictions of a model with data not used to perform model fitting.

$$PSSE = \sum_{i=n-k+1}^{n}(\widehat{H}_{i;\omega,\boldsymbol{\theta},\boldsymbol{\beta}} - Y_i)^2 \quad (18)$$

where the maximum likelihood estimates of the model parameters are determined from the first $n-k$ intervals.

A. Critic Method

In addition to providing multiple goodness of fit measures, the C-SFRAT provides a simple approach based on the critic method [3] to select models based on a weighted combination of one or more measures. Given r models and m measures, let $f_{i,j}$ be the jth measure of the ith model. Each measure is assigned a normalized score in the interval $(0,1)$ according to

$$s_{i,j} = \frac{f_{i,j} - f_j^-}{f_j^+ - f_j^-} \quad (19)$$

where f_j^+ (f_j^-) denotes the best (worst) value of a measure j across all models. Thus, $s_{i,j}$ indicates how close the jth measure of model i is to the ideal and a score of 1 (0) is the best (worst). C-SFRAT implements two ways to select a model with the critic method. The first computes the mean of each model's normalized scores and recommends the model possessing the highest mean. The second method recommends a model base on the median of the normalized scores.

VIII. Illustrations

This section applies the Discrete Cox Proportional Hazard NHPP SRGM to two defect discovery data sets [8] containing three covariates. Each of the 8 hazard functions described in Section IV was applied to all possible combinations of the covariates. For each data set, the 8 best-fitting hazard function and covariate combinations are reported. Plots also show the best-fitting combinations of hazard function and covariates along with the underlying data.

Table I shows the 8 combinations of hazard function and covariates that performed best on DS1 with the critic method according to the median of the normalized scores.

TABLE I: Top 8 best-fitting combinations of hazard function and covariates for DS1

Hazard	Cov.	LLF	AIC	BIC	SSE	Critic
NB2	E, F, C	-27.29	64.57	68.74	11.89	**1.000**
NB2	F	-28.80	**63.60**	**66.10**	25.94	0.995
NB2	E, F	-28.05	64.10	67.44	18.33	0.988
NB2	F, C	-28.37	64.75	68.08	20.76	0.965
DW2	F	-29.47	64.94	67.44	41.52	0.957
S	F	-28.72	65.44	68.77	23.00	0.940
S	E, F, C	**-27.14**	66.27	71.27	**9.18**	0.937
DW3	E, F	-28.23	66.46	70.62	22.50	0.931

Values in bold indicate the combination of hazard function and covariates that performed best with respect to the goodness-of-fit measures included in the critic method. In this case,

the negative binomial of order two (NB2) hazard function originally proposed by Shibata et al. [8] performed best. The IFR SB hazard function was excluded because it performed poorly, skewing the critic method of other models toward 1.0.

Figure 2 shows the number of defects discovered in each interval as well as the three combinations of hazard function and covariates that performed best according to Table I.

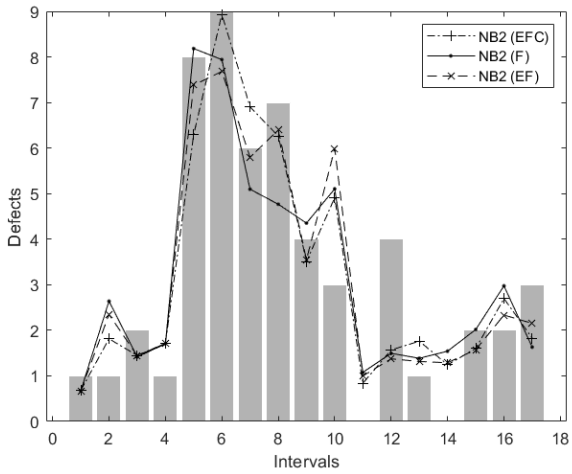

Fig. 2: Defects discovered in each interval of DS1

Table II shows the 8 combinations of hazard function and covariates that performed best on DS2 with the critic method according to the median of the normalized scores.

TABLE II: Top 8 best-fitting combinations of hazard function and covariates for DS2

Hazard	Cov.	LLF	AIC	BIC	SSE	Critic
IFRGSB	F, C	**-22.18**	54.36	57.56	**25.17**	**1.000**
GM	F, C	-23.01	54.03	56.58	48.63	0.999
GM	F	-23.29	**52.58**	**54.50**	49.61	0.996
S	F	-23.15	54.31	56.86	50.69	0.992
TL	F	-23.29	54.58	57.14	49.61	0.991
GM	C	-24.33	54.65	56.57	44.25	0.988
GM	E, F	-23.27	54.54	57.10	54.24	0.981
DW3	F	-23.43	54.85	57.41	46.32	0.980

The increasing failure rate generalized Salvia and Bollinger (IFRGSB) hazard function presented here performed best.

Figure 3 shows the number of defects discovered in each interval as well as the three combinations of hazard function and covariates that performed best according to Table II.

IX. Conclusion and Future Research

The C-SFRAT is an open source software reliability tool that implements covariate models and supports the addition of new hazard functions, goodness-of-fit measures, and optimization problems. Application of the C-SFRAT to two data sets from the literature indicated that, in some cases, newly incorporated hazard functions performed best.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Number (1749635).

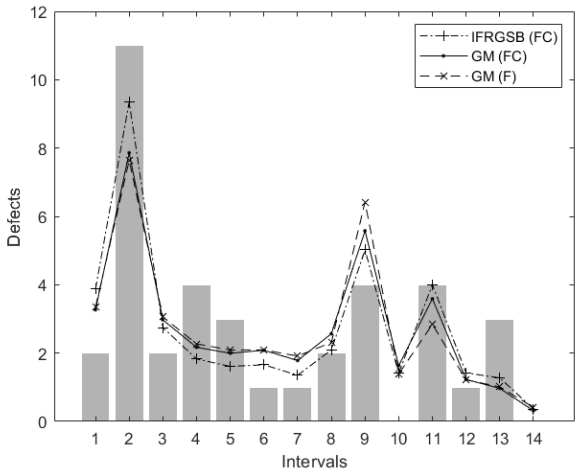

Fig. 3: Defects discovered in each interval of DS2

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] W. Farr and O. Smith, "Statistical modeling and estimation of reliability functions for software (SMERFS) users guide," Naval Surface Warfare Center, Dahlgren, VA, Tech. Rep. NAVSWC TR-84-373, Rev. 2, 1984.

[2] K. Rinsaka, K. Shibata, and T. Dohi, "Proportional intensity-based software reliability modeling with time-dependent metrics," in *IEEE International Conference on Computer Software and Applications*, vol. 1, 2006, pp. 369–376.

[3] V. Nagaraju, C. Jayasinghe, and L. Fiondella, "Optimal test activity allocation for covariate software reliability and security models," *Journal of Systems and Software*, p. 110643, 2020.

[4] M. Lyu and A. Nikora, "CASRE: A computer-aided software reliability estimation tool," in *IEEE International Workshop on Computer-Aided Software Engineering*, 1992, pp. 264–275.

[5] V. Nagaraju, V. Shekar, J. Steakelum, M. Luperon, Y. Shi, and L. Fiondella, "Practical software reliability engineering with the software failure and reliability assessment tool (SFRAT)," *SoftwareX*, vol. 10, p. 100357, 2019.

[6] K. Shibata, K. Rinsaka, and T. Dohi, "M-SRAT: Metrics-based software reliability assessment tool," *International Journal of Performability Engineering*, vol. 11, no. 4, 2015.

[7] H. Okamura and T. Dohi, "SRATS: Software reliability assessment tool on spreadsheet (experience report)," in *International Symposium on Software Reliability Engineering*, Nov 2013, pp. 100–107.

[8] K. Shibata, K. Rinsaka, and T. Dohi, "Metrics-based software reliability models using non-homogeneous Poisson processes," in *IEEE International Symposium on Software Reliability Engineering*, 2006, pp. 52–61.

[9] C. Bracquemond and O. Gaudoin, "A survey on discrete lifetime distributions," *International Journal of Reliability, Quality and Safety Engineering*, vol. 10, no. 1, pp. 69–98, 2003.

[10] W. Padgett and J. Spurrier, "On discrete failure models," *IEEE Transactions on Reliability*, vol. R-34, no. 3, pp. 253–256, Aug 1985.

[11] J. Soler, "Croissance de fiabilite des versions d'un logiciel," *Revue de statistique appliquée*, vol. 44, no. 1, pp. 5–20, 1996.

[12] G. Adams and R. Watson, "A discrete time parametric model for the analysis of failure time data," *Australian Journal of Statistics*, vol. 31, no. 3, pp. 365–384, 1989.

[13] B. Klar, "Theory & methods: Model selection for a family of discrete failure time distributions," *Australian & New Zealand Journal of Statistics*, vol. 41, no. 3, pp. 337–352, 1999.

[14] A. Salvia and R. Bollinger, "On discrete hazard functions," *IEEE Transactions on Reliability*, vol. R-31, no. 5, pp. 458–459, 1982.