SoftwareX 17 (2022) 100909

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

A covariate software tool to guide test activity allocation

Jacob Aubertine [a], Kenan Chen [a], Vidhyashree Nagaraju [b], Lance Fiondella [a,*]

[a] Department of Electrical and Computer Engineering, University of Massachusetts, Dartmouth, MA, 02747, USA
[b] Tandy School of Computer Science, University of Tulsa, Tulsa, OK, 74107, USA

ARTICLE INFO

Article history:
Received 12 June 2021
Received in revised form 10 October 2021
Accepted 19 November 2021

Keywords:
Software reliability
Software reliability growth model
Covariates
Test activity allocation

ABSTRACT

This paper presents the Covariate Software Failure and Reliability Assessment Tool (C-SFRAT) to automatically apply methods from software reliability engineering to accurately characterize the defect discovery process in terms of the test activities performed. The tool enables calculations and visualizations, including plots of the defect discovery data, covariate models fit to this data, and inferences made possible by these models as well as assessment of model goodness-of-fit. A generalized optimization procedure, referred to as the test activity allocation problem, has been implemented to guide the distribution of limited resources among specific test activities in order to maximize defect discovery for corrective action and improved reliability. The application and source code are freely available from the GitHub repository dedicated to this project. The open source nature of this tool will enable collaboration among researchers and practitioners from industry and government within a single shared platform.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00110
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Python, GitHub
Compilation requirements, operating environments & dependencies	https://github.com/LanceFiondella/C-SFRAT/blob/master/requirements.txt
If available Link to developer documentation/manual	https://lfiondella.sites.umassd.edu/research/software-reliability/
Support email for questions	lfiondella@umassd.edu

Software metadata

Current software version	v1.0
Permanent link to executables of this version	https://github.com/LanceFiondella/C-SFRAT
Legal Software License	MIT License
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows, and Unix-like
Installation requirements & dependencies	Refer to user's guide
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://lfiondella.sites.umassd.edu/research/software-reliability/
Support email for questions	lfiondella@umassd.edu

1. Motivation and significance

For decades, the software and reliability engineering [1] community has utilized traditional software reliability growth models

* Corresponding author.
E-mail addresses: jaubertine@umassd.edu (Jacob Aubertine), kchen3@umassd.edu (Kenan Chen), vidhyashree-nagaraju@utulsa.edu (Vidhyashree Nagaraju), lfiondella@umassd.edu (Lance Fiondella).

https://doi.org/10.1016/j.softx.2021.100909

(SRGM) [2] to characterize the defect discovery process during testing as a non-homogeneous Poisson process (NHPP). While tools implementing NHPP models [3] can quantify and predict trends in metrics such as failure intensity and mean time to failure as a function of testing time or effort [4], the parametric nature of these models cannot be linked to the underlying testing activities in a concrete manner. More recently, metrics-based software reliability growth models [5,6] have been proposed to characterize defects discovered as a function of multiple software testing activities such as calendar time, number of test cases executed, and test execution time. Tools incorporating these models [7] have also been developed. However, they are restricted to the models and predictions proposed by the authors and implemented in an Excel spreadsheet or the R programming language. A general purpose software tool is needed to encourage the widespread use of metrics-based software reliability growth models as well as the consistent collection of high quality metrics data required to apply these models.

An early metrics-based tool was the Enhanced Measurement for Early Risk Assessment of Latent Defects (EMERALD) [8] system, which analyzed source code to identify fault prone modules. ROBUST [9] included static modeling to characterize metrics data, with time-based NHPP SRGM and coverage-based models for reliability estimation. The Software Reliability Estimation and Prediction Tool (SREPT) [10] characterized failure data along with product metrics to quantify reliability during different phases of the software life cycle. The Computer-Aided Software Reliability and Estimation (CASRE) [11] tool introduced combination models that reduced the predictive biases of the component models. More recently, The Software Failure and Reliability Assessment Tool (SFRAT) [3] implemented traditional software reliability growth models and provided an open source framework through which additional models could be added. The Proportional Intensity-based Software Reliability Assessment Tool (PISRAT) [12] characterized failure data along with testing metrics based on 11 proportional intensity based software reliability models. The Metrics-based Software Reliability Assessment Tool (M-SRAT) [7] builds upon PISRAT by adding additional modeling approaches, characterizing testing effort as covariates based on three hazard rate functions. However, these tools are not explicitly open source or were not designed to support extensibility beyond models originally included by the authors. To help realize the full potential of covariate models, Nagaraju et al. [13] developed optimization procedures, referred to as test activity allocation problems, to guide the distribution of limited resources among specific test activities in order to maximize defect discovery, facilitate their correction, and improve reliability.

This paper presents an open source Covariate Software Failure and Reliability Assessment Tool (C-SFRAT) that implements the models and effort allocation strategy developed in [13]. The open source nature of the tool and its flexible architecture will promote collaboration among members of the software reliability research community and users from industry and government organizations. The architecture enables incorporation of existing covariate models into a single tool, enabling more systematic comparison of models. Present functionality includes: (i) eight models based on discrete hazard rate functions, (ii) five measures of goodness-of-fit, (iii) a model selection strategy based on these measures of goodness-of-fit, according to the user's preference, (iv) inferences such as defect prediction and failure intensity prediction, and (v) optimal test activity allocation, which recommends how to distribute limited resources across two or more alternative test activities or metrics in order to maximize the number of defects discovered, so that they can be resolved prior to release.

The remainder of the paper is organized as follows: Section 2 provides an overview of the tool's user interface. The four primary screens within a typical workflow are described in Section 2.2 through 2.5. Section 3 describes the tool's potential for impact. Section 4 concludes the paper and identifies future work.

	A	B	C	D	E
1	T	FC	E	F	C
2	1	1	0.0531	4	1
3	2	1	0.0619	20	0
4	3	2	0.158	1	0.5
5	4	1	0.081	1	0.5

Fig. 1. C-SFRAT input data format.

2. Software description

This section provides a detailed description of the data requirements, graphical user interface, and workflow of the C-SFRAT. Section 2.1 describes covariate data, explains how spreadsheets containing covariate data should be formatted for use in the C-SFRAT, and provides an overview of the tool's workflow. Section 2.2 explains how to import covariate data into the tool and perform model fitting with desired combinations of hazard functions and covariates. Section 2.3 describes plots of the fitted models and predictions of future failures and failure intensity based on a specified set of testing activities. Section 2.4 explains how to assess alternative models based on the goodness-of-fit measures computed from the fitted models as well as a weighted model selection strategy. Section 2.5 illustrates the application of two types of test activity allocation: (i) maximization of the number of defects to be discovered despite a budget constraint and (ii) minimization of the budget required to discover a specified number of additional defects.

Readers wishing to gain hands-on experience with the tool may wish to first install the C-SFRAT before proceeding. The C-SFRAT has been implemented in the Python programming language and runs on Windows, macOS, and Linux. The source code is accessible through the GitHub repository at https://github.com/LanceFiondella/C-SFRAT as well as installation instructions. Example data sets for the purpose of evaluating the tool's functionality can be found at https://lfiondella.sites.umassd.edu/research/software-reliability, eliminating the need to collect and format testing data. The data set used for the following example is DS1.

2.1. Covariate data

Covariate data [6] records the number of defects detected as well as the effort expended on one or more activities, in discrete non-overlapping time intervals. The activities, also referred to as covariates, encompass a variety of reliability or security testing techniques and tools, including code reviews, walkthroughs, execution time, and fuzz testing as well as metrics correlated with software quality such as code coverage.

Fig. 1 shows the header and first few rows of a covariate data set possessing three covariates.

The tool accepts data formatted in Excel spreadsheet (.xlsx) or a comma-separated values (.csv) format. Each sheet within a spreadsheet can specify a unique data set, whereas CSV files are restricted to a single data set. The first two columns of Fig. 1, A and B, are labeled *T* (time interval) and *FC* (failure count), respectively denoting the discrete time interval and the number of defects detected during that time interval. Column *T* is optional. If it is omitted, the tool automatically generates integer indexed failure intervals. The remaining column headers E (execution time measured in hours), F (failure identification

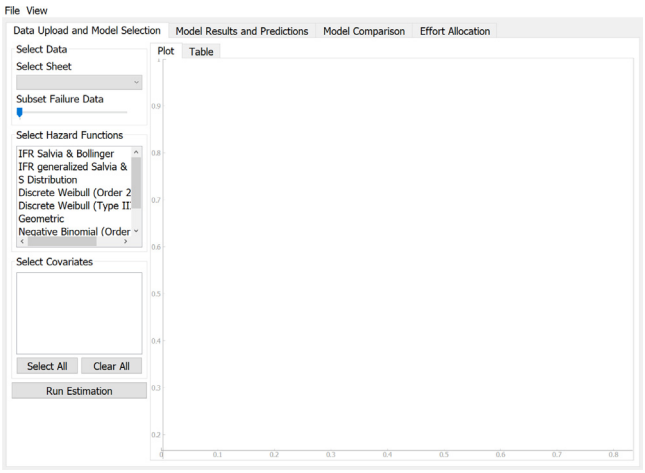

Fig. 2. Graphical User Interface of C-SFRAT.

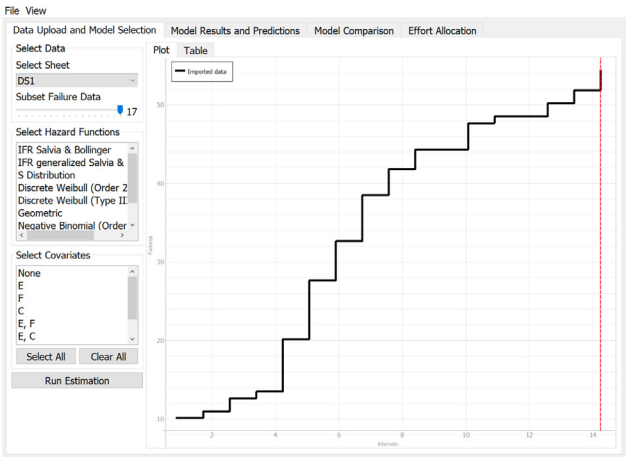

Fig. 3. Tab 1: Plot of cumulative failures.

work measured in person hours), and C (computer time failure identification measured in hours) in columns (C through E) are user-defined column names for covariate data. These covariate names specified in the first row are displayed within the user interface. However, if names are not explicitly specified in the header, the program will automatically labels of the form Cov*n*, where *n* is the number of columns after the failure counts. The values in each row denote the time dedicated to each activity in the *i*th interval. For example, in interval two, 0.0619 units of computer time failure identification, 20 units of failure identification work, and no execution time were applied, which resulted in the discovery of one defect.

Fig. 2 shows the view of the tool's graphical user interface on startup. The C-SFRAT workflow is organized across four tabs. The layout of each tab includes a menu of controls on the left and an area for plots and tables of results on the right. A summary of the primary function of each tab is as follows:

- **Tab 1: Data Import and Model Selection**: Allows the user to import failure data from a file in Excel or CSV format, as shown in Fig. 1. This tab also allows the user to select the data set to work with as well as the hazard functions and covariates for model fitting and assessment.
- **Tab 2: Model Results and Predictions**: Displays the model fit and failure intensity plots of the hazard function and covariate combinations selected in Tab 1. The user can also predict the future failures and failure intensity based on a specified testing activity profile.
- **Tab 3: Model Comparison**: Compares the fitted models based on information theoretic and predictive goodness-of-fit measures. The user may also assign weights to individual measures to support model selection.
- **Tab 4: Effort Allocation**: Recommends test activity allocation [13] in order to maximize defect discovery within a specified budget or minimize the total testing resources required to discover a specified number of defects.

The following subsections describe each tab in greater detail.

2.2. Tab 1: Data import and model selection

To import a data set, the user must select the *Open* option under the *File* menu and select an Excel or CSV file.

Fig. 3 shows Tab 1 after data has been successfully imported. The control menu on the left side of Fig. 3 shows the name of the data sheet, hazard functions available, and possible combinations of covariates. If the data source is an Excel spreadsheet, the user can select a specific data set from the drop-down list under *Select Sheet*. The slider under *Subset Failure Data* allows the user to select the number of intervals of data to be used for model fitting. Excluding a subset of the intervals is appropriate for testing the predictive accuracy of models, whereas fitting models with all intervals is suitable for applying the test activity allocation optimization problems because this utilizes all of the data observed so far.

The *Plot* and *Table* options above the right side of Fig. 3 displays the selected data set visually or numerically. The default plot shows the cumulative defects detected in successive intervals. Plot settings can be modified from the *View* drop down menu on the menu bar. Options include showing the data using points, lines, or both points and lines as well as switching between the cumulative failures and failure intensity plots. The toolbar below the plot enables an image to be saved in several standard file formats for inclusion in reports and publications. Adjusting the slider automatically updates the plot and table to only the selected subset.

To apply models, the user must select at least one hazard function from the list under *Select Hazard Functions* and at least one combination of covariates under *Select Covariates*. For the sake of illustration, the running example that follows uses three hazard functions, including the second order discrete Weibull (DW2), Geometric (GM), and second order negative binomial (NB2). All possible combinations for covariate data sets are listed under the *Select Covariates* option, including *None*, which corresponds to the special case of the discrete nonhomogeneous Poisson process model with likelihood function defined in Ref. [13]. In the cases where there are several covariates, *Select All* enables selection of all 2^n possible combinations of covariates. The purpose of applying models with different hazard functions and subsets of covariates is to identify the combinations that achieve the most accurate prediction and goodness-of-fit. Therefore, it is suggested that all combinations be applied and those exhibiting lower goodness-of-fit be eliminated from further consideration. It should also be noted that the relative performance of models can change as more data is collected.

After selecting at least one hazard function and subset of covariates, the combinations are applied to the present data set by clicking the *Run Estimation* button located under the *Select Covariates* box. During these calculations, a separate window lists the model currently being fitted and progress bar indicating the fraction of models fitted so far.

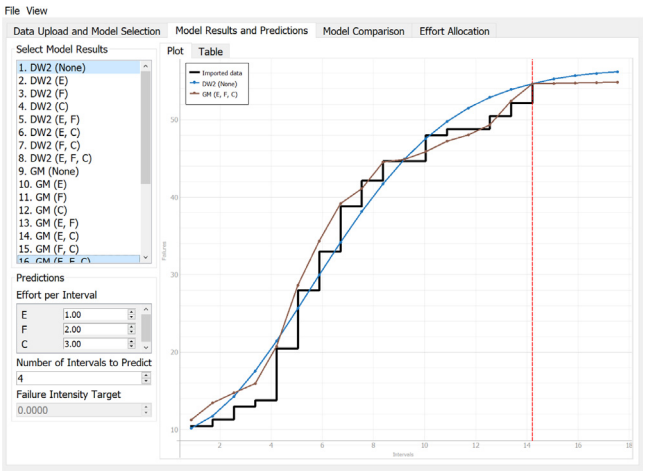

Fig. 4. Tab 2: Plot of cumulative failures with fitted models superimposed.

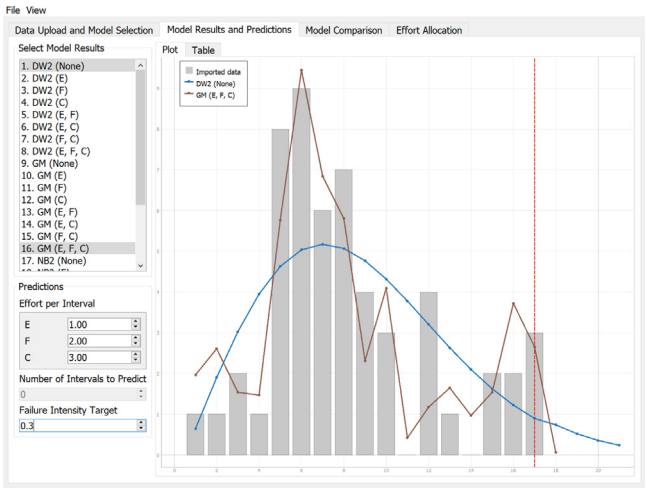

Fig. 5. Tab 2: Plot of failures detected in each interval and failure intensity of fitted models superimposed.

2.3. Tab 2: Model results and predictions

Tab 2 is used to display plots of fitted models and predict the number of future failures or failure intensity using these models. Fig. 4 shows Tab 2 with two models superimposed on the cumulative plot of the DS1 data set. All combinations of models and covariates that were selected in Tab 1 are listed in the box labeled *Select Model Results*. For the sake of illustration, the discrete Weibull with no covariates *DW2 (None)* and geometric model with all three covariates *GM (E, F, C)* were selected and are therefore displayed on the plot.

Below the list of fitted models are controls to specify *Prediction* parameters using the selected combinations. When the cumulative failures plot is displayed, as in Fig. 4, the user must specify the *Number of Intervals to Predict* and *Effort per Interval*. For example, Fig. 4 shows that model prediction for the number of additional defects that would be detected if 1.0, 2.0, and 3.0 units of effort to covariates E, F, and C respectively were applied in each of the four intervals beyond the vertical line at interval 17. The effort per interval may be interpreted as the testing schedule, but can also be used to predict field defect or vulnerability discovery in the case where covariates correspond to the intensity of various types of attack leveled against a system in its operational environment.

By switching plots from the *View* menu, Fig. 5 shows the number of defects detected in each interval as well as the failure intensity of the fitted models selected. Fig. 5 clearly illustrates how covariate models more accurately track the number of defects detected in each interval because the testing activities driving defect discovery are explicitly considered.

In addition to tracking the observed data, Fig. 5 can predict the number of additional intervals to achieve a *Failure Intensity Target*. For each combination selected, the prediction shows all points until the interval that attains a value below the desired failure intensity, as shown in Fig. 5. For example, to achieve of failure intensity of 0.3, the discrete Weibull model with no covariates predicts four additional intervals would be required, whereas the geometric model with E, F, and C covariates predicts one additional interval would be required. The number of intervals required is also reported in a table.

2.4. Tab 3: Model comparison

In statistics, no model characterizes all data sets best and no measure of goodness-of-fit is ideal for model selection. Therefore, Tab 3 allows the user to compare alternative models with respect to common statistical goodness-of-fit measures [14], including those based on the log-likelihood value such as the Akaike information criterion (AIC) and Bayesian Information Criteria (BIC) as well as sum of squares (SSE) and predictive sum of squares error (PSSE).

In addition to providing multiple goodness of fit measures, the C-SFRAT provides a simple approach based on the critic method [13] to select models based on a weighted combination of one or more measures. Given n models and m measures, let $f_{i,j}$ be the jth measure for the ith model. Each measure is assigned a normalized score in the interval (0, 1) according to

$$x_{i,j} = 1 - \frac{f_{i,j} - f_j^+}{f_j^- - f_j^+} \tag{1}$$

where f_j^+ and f_j^- respectively denote the best and worst values of a measure j across all models. Thus, $x_{i,j}$ indicates how close the jth measure of model i is to the ideal, where $x_{i,j} = 1.0$ if model i performs best on measure j. Two alternative methods to select a model are to compute the mean or median of each model's normalized scores and recommend the model possessing the highest mean or median normalized value.

Since it is rare that a model performs best on all measures and no single measure is a complete indicator of a model's suitability, a simple discrete weighted average technique enables the user to specify their preference without overcomplicating the weighting process. Each measure can be assigned a weight in the range 0–10, where zero indicates no weight and ten the highest weight. The left pane of Fig. 6 indicates that the default weighting is one (uniform) for each measure, which corresponds to equal weighting. By default, PSSE values are computed by performing model fitting on 90% of the data. The percentage of data can be changed in the spinbox labeled *Specify subset data for PSSE*. The user must then press the button labeled *Run* to perform model fitting on the new subset.

Setting a weight to zero eliminates a measure and setting all but one weight to zero is the special case considering only a single measure. The right pane reports each hazard function and combination of covariates successfully applied, the log-likelihood attained, and goodness-of-fit values. The right two columns of Fig. 6 model scores based on these weightings according to the mean and medium measure respectively. The user can sort the

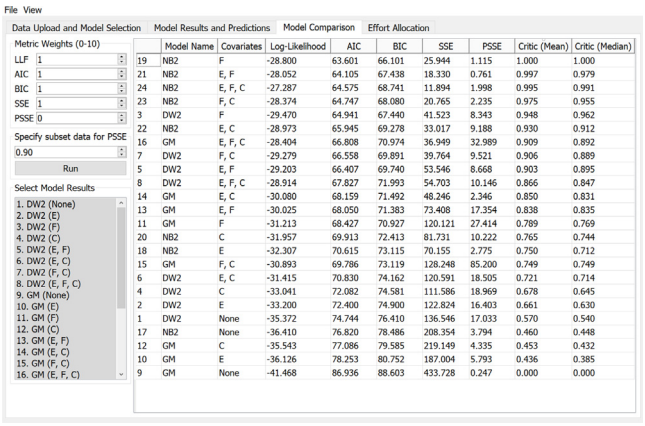

Fig. 6. Tab 3: Model selection.

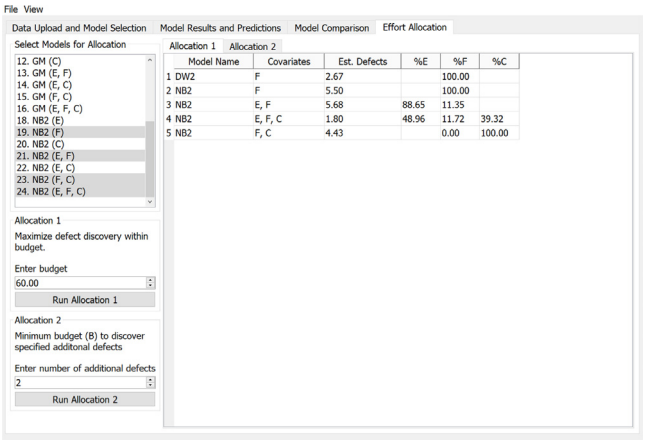

Fig. 7. Tab 4: Effort Allocation 1.

columns in descending order by clicking the header. This enables model selection for statistical inference on Tab 2 and effort allocation on Tab 4.

2.5. Tab 4: Effort allocation

Tab 4 computes the optimal allocation of testing activities based on the models applied in Tab 2. Two effort allocation problems [13] allow a user to (i) maximize defect discovery within a budget constraint or (ii) minimize the budget required to discover a specified number of defects. The two problems are referred to as *Allocation 1* and *Allocation 2* in the tool.

Prior to computing recommended test activity allocations, the user specifies a budget for *Allocation 1* or the number of defects to be exposed for *Allocation 2*. Here, budget can be interpreted as time, but also as cost in cases where alternative test activities incur different costs per unit time. For example, one activity may require outside consultants with highly specialized expertise, whereas others rely on members of the project team.

Fig. 7 shows Tab 4 with *Allocation 1*. The combinations of hazard function and test activities successfully fit to the data are shown in the top left pane. The user can select one or more models to use as the basis for test activity recommendations. In practice, effort allocation should be performed with the models that best fit the data. For example, models can be chosen based on the goodness-of-fit measures considered in Tab 3.

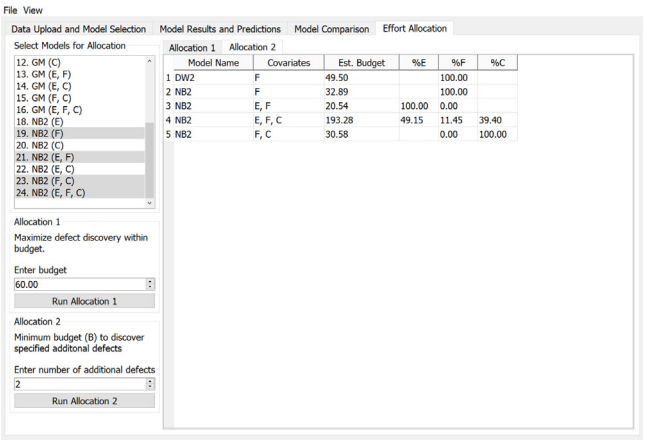

Fig. 8. Tab 4: Effort allocation 2.

Below the list of models, the user can define the parameters for *Allocation 1* and *Allocation 2*. The budget constraint for *Allocation 1* is specified in the spin box labeled *Enter budget*. Once the desired parameters and models have been selected, the user can click the *Run Allocation 1* button, after which the *Allocation 1* table is populated. The *Allocation 2* controls are located below the controls for *Allocation 1*, in the spin box labeled *Enter number of additional defects*, where the user can define the number of defects to uncover. Clicking the *Run Allocation 2* button solves the second allocation problem and displays the results in the *Allocation 2* table.

The five best fitting models according to the critic method in Fig. 6 were chosen for comparison. The right pane of Fig. 7 shows the results of *Allocation 1* on the five models using a budget of 60. The column *Est. Defects* column indicates the predicted number of additional defects that would be detected if the specified budget is allocated according to the percentages in the columns to the right. For example, row three of the table, suggests the negative binomial with covariates E, F, and C predicts 1.80 additional defects if 48.96% of the budget is allocated to E, 11.72% of the budget is allocated to F, and 39.32% of the budget is allocated to C. Note that the predictions are decimal values because the mean value function of the model provides a continuous approximation to the discrete process of discovering defects. The results of *Allocation 2* when specifying one additional defect are shown in Fig. 8. Here, the negative binomial with covariates E, F, and C, predict the minimum budget to uncover two additional defects is 193.28, with 49.15% of the budget allocated to E, 11.45% of the budget allocated to F, and 39.40% of the budget allocated to C. Models with a single activity allocate 100% of testing to that activity, but are included in the tables because the right pane also reports model predictions of the estimated defects or budget respectively. Allocation can aid decisions, but human judgement must be considered. Therefore, a practical approach is to iteratively allocate, collect data, and repeat until a desired target is achieved.

3. Impact

Past research has overemphasized nonhomogeneous Poisson process models that do not consider the underlying test activities driving defects or vulnerabilities discovered as well as simplified optimization problems that only consider the amount of time or effort to be allocated and optimal release problems that do not fully account for the details of modern software development practices.

Like the SFRAT (Software Failure and Reliability Assessment Tool), the C-SFRAT (Covariate Software Failure and Reliability Assessment Tool) offers to promote communication between software reliability researchers and practitioners. Specifically, the open source nature of the tool promotes the inclusion of additional hazard functions and measures of goodness-of-fit as well as the opportunity to extend the tool with additional test allocation problems. Thus, researchers contributing to the tool can make their results available to their target audience. Similarly, practitioners can communicate practical software engineering challenges that enable new modeling research for their benefit, including how to effectively apply models throughout the testing process. The open source approach will also encourage model comparison, reproducibility of research, and data sharing.

The C-SFRAT will be of interest to government organizations concerned with the quality of software produced with taxpayer dollars, organizations that perform independent analysis, and private companies. The tool automates the most difficult aspects of applying covariate models, including symbolic differentiation of the objective function and optimization procedure to identify numerical parameters that best characterize input data. This encapsulation allows users to focus on the model predictions and practical allocation decisions. Thus, the C-SFRAT significantly reduces the learning curve required to apply covariate models to quantitatively assess software, promoting collection of accurate software metrics and test activity tracking in order to conduct quantitative process assessment and improvement efforts.

We do not require individuals to register prior to download and therefore cannot accurately state the full extent of its use. However, we welcome constructive feedback to better serve the research and user community.

4. Conclusion and future work

This paper presents the Covariate Software Failure and Reliability Assessment Tool, an open source tool to automatically apply covariate models to software test activity data. Model fitting can be performed with eight discrete hazard rate functions and model fitness compared quantitatively using five goodness-of-fit measures. The fitted models are used to predict future defects and to determine optimal allocation of future testing resources. The open source architecture of the C-SFRAT is intended to encourage collaboration among software reliability researchers and practitioners.

The tool has been successfully tested on data sets with up to 10 activities. In some cases, one or two activities are sufficient to improve predictions significantly. Future research will (i) formulate additional practical test activity allocation problems, (ii) implement stable and efficient algorithms to fit models documenting many activities, and (iii) develop a sequential procedure to identify a sufficient number of activities for prediction.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This material is based upon work supported by the National Science Foundation, United States 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] Lyu M, editor. Handbook of software reliability engineering. New York, NY: McGraw-Hill; 1996.
[2] Farr W, Smith O. Statistical modeling and estimation of reliability functions for software (SMERFS) users guide. Tech. rep. NAVSWC TR-84-373, Rev. 2, Dahlgren, VA: Naval Surface Warfare Center; 1984.
[3] Nagaraju V, Shekar V, Steakelum J, Luperon M, Shi Y, Fiondella L. Practical software reliability engineering with the software failure and reliability assessment tool (SFRAT). SoftwareX 2019;10:100357. http://dx.doi.org/10.1016/j.softx.2019.100357.
[4] Yamada S, Ohtera H, Narihisa H. Software reliability growth models with testing-effort. IEEE Trans Reliab 1986;35(1):19–23.
[5] Rinsaka K, Shibata K, Dohi T. Proportional intensity-based software reliability modeling with time-dependent metrics. In: IEEE International conference on computer software and applications, vol.1, 2006. p. 369–76.
[6] Shibata K, Rinsaka K, Dohi T. Metrics-based software reliability models using non-homogeneous Poisson processes. In: IEEE International symposium on software reliability engineering, 2006. p. 52–61.
[7] Shibata K, Rinsaka K, Dohi T. M-SRAT: Metrics-based software reliability assessment tool. Int J Perform Eng 2015;11(4).
[8] Hudepohl J, Aud S, Khoshgoftaar T, Allen E, Mayrand J. Emerald: Software metrics and models on the desktop. IEEE Softw 1996;13(5):56–60.
[9] Li N, Malaiya Y. ROBUST: A next generation software reliability engineering tool. In: International symposium on software reliability engineering, 1995. p. 375–80.
[10] Ramani S, Gokhale S, Trivedi K. SREPT: Software reliability estimation and prediction tool. Perform Eval 2000;39(1):37–60. http://dx.doi.org/10.1016/S0166-5316(99)00057-7.
[11] Lyu M, Nikora A. CASRE: A Computer-Aided Software Reliability Estimation Tool. In: IEEE International workshop on computer-aided software engineering, 1992.p. 264–75.
[12] Shibata K, Rinsaka K, Dohi T. PISRAT: Proportional Intensity-Based Software Reliability Assessment Tool. In: Pacific rim international symposium on dependable computing, 2007. p. 43–52 https://doi.org/10.1109/PRDC.2007.18.
[13] Nagaraju V, Jayasinghe C, Fiondella L. Optimal test activity allocation for covariate software reliability and security models. J Syst Softw 2020;110643.
[14] Kleinbaum D, Kupper L, Nizam A, Muller K. Applied regression analysis and other multivariable methods. Applied Series, 4 th. Belmont, CA: Duxbury Press; 2008.