

A proposed model for detecting defects in software projects

Alia Nabil Mahmoud¹, Ahmed Abdelaziz^{1,2}, Vitor Santos¹, Mario M. Freire³

¹Nova Information Management School, Universidade Nova de Lisboa, Lisboa, Portugal

²Information System Department, Higher Technological Institute, Cairo, Egypt

³Department of Computer Science, University of Beira Interior Rua Marquês de Ávila e Bolama, Covilhã, Portugal

Article Info

Article history:

Received Dec 9, 2022

Revised Aug 15, 2023

Accepted Oct 25, 2023

Keywords:

Defects

Linear regression

Logistic regression

Software projects

Statistical model

ABSTRACT

Defective modules that cause software execution failures are common in large software projects. Source code for a significant number of modules may be found in several software repositories. This software repository includes each module's software metrics and the module's faulty status. Software companies face a considerable problem detecting defects in sizeable and complex programming code. In addition, many international reports, such as the comprehensive human appraisal for originating (CHAOS) report, have mentioned that there are countless reasons for the failure of software projects, including the inability to detect errors and defects in the programming code of those projects at an early stage. This research employs a statistical analysis technique to reveal the characteristics that indicate the faulty status of software modules. It is recommended that statistical analysis models derived from the retrieved information be merged with existing project metrics and bug data to improve prediction. When all algorithms are merged with weighted votes, the results indicate enhanced prediction abilities. The proposed statistical analysis outperforms the state-of-the-art method (association rule, decision tree, Naive Bayes, and neural network) in terms of accuracy by 9.1%, 10.3%, 13.1%, and 13.1%, respectively.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Alia Nabil Mahmoud

Nova Information Management School, Universidade Nova de Lisboa

1070-312 Lisboa, Portugal

Email: m20190508@novaims.unl.pt

1. INTRODUCTION

Using statistical analysis makes it possible to unearth many previously unknown software characteristics [1]. This procedure includes a long-standing interest in the success of software projects, assistance with software testing management, and the discovery of fault patterns [2], [3]. The program's quality is considered by the estimation and prediction approaches for software problems [4]. In addition, they are used after the product has been delivered to estimate the effort and expenditure associated with maintenance [5]. Maintainability is heavily influenced by software design metrics [6].

There is a clear relationship between software quality and process maturity [7]. Many studies [8], [9] have stressed the need for inspections, especially requirements and design inspections, to limit faults' density. During the coding phase, the number of defect densities decreased exponentially because faults were corrected as soon as they were identified and did not propagate to further stages of the process. However, because the code review process is labour-intensive, this peer review-based inspection technique becomes prohibitively expensive and unsustainable throughout the coding phase: it is possible to evaluate 8–20 LOC every minute, and this effort is repeated for each review team member.

The ability to predict, at the code level, which modules are most likely to encounter issues can assist in determining which ones require costly code reviews, inspections, and testing. Nevertheless, it is imperative

to exercise caution when utilizing this prediction method in order to avoid perpetuating the issue [9]. Ensuring that all parts of a system are accessible is crucial to prevent errors from being overlooked. To achieve this, various suggested static code measure-based fault detectors have been proposed. This approach is particularly relevant in software assessment, where certain regions are deemed mission-critical and require special attention [10].

In order to create and verify static defect models for several software projects or different versions of the same project, empirical data from software repositories has been collected and analysed [11]. Examples of such efforts include the NASA data sets available online [12], comprising five massive software projects, including hundreds of modules and five large software projects. For each set, the static measurements and other factors are mentioned and discussed in the first paragraph of the introduction. The data points represent statistical measures and a binary variable indicating whether or not the module is malfunctioning [13].

The primary emphasis of static measurements generated from source code is on identifying whether or not a module is problematic to model the causes of faulty modules [14]. Statistical analysis approaches are applied when identifying patterns indicating modules likely to fail [15]. Support vector machine (SVM) based service-oriented architecture and expert constructive cost model (COCOMO)-based service-oriented architecture are two other possibilities [16], [17]. Many studies from the literature review used NASA's projects [16], [17]. The data sets are arranged in descending order of module size. The bigger the number of modules, the better the forecast performance. Using these data, practitioners may develop a set of criteria for defect prediction that they can use in their work. For example, observations by some researchers [18], [19] revealed poor pre-prediction performance, which might be improved by following the suggestions. As things stand, these criteria have not been implemented into an existing software system or integrated with compilers, allowing developers to guarantee that they are followed, and code is written in a way that decreases the probability of defective modules being generated.

It is difficult, if not impossible, to compare techniques when there is no standard for defect prediction [20], [21] giving a publicly accessible data set of numerous software systems and a comprehensive comparison between well-known bug prediction methodologies, such as a regression model. The experimental method tried [22] to develop a process for evaluating software defect classification models and showed no significant performance variations among the top 17 classifiers. Moreover, this study sought [23] to provide an examination of several software defect models. In order to compare a broad variety of machine learning approaches [24], the PROMISE data sets are used. Unfortunately, predictive power was not a clear winner in this comparison, and no obvious winner emerged.

In general, the three issues described above may be summarised as follows: to begin, project managers and quality teams need realistic software tools that can be used to predict which new software modules will be problematic based on data from publicly available software repositories. This facet allows them to allocate their testing and debugging resources properly. Furthermore, these tools should use and integrate a variety of statistical analysis methodologies and compare and contrast the results to address the problem of predicting defective modules in software projects since there is no definite finding of the most effective statistical analysis methodology at this time.

The contributions made by this study are noteworthy for two reasons. First, software repositories may be employed to enhance software development, and this architecture provides the framework for doing so. It has already been mentioned that many references converted the software measurements into rules and knowledge that experienced developers may apply to their code. A software architecture that can be implemented and integrated with compilers is required to use this extracted knowledge to warn less experienced members of the software development team of potentially defective modules and allow the project manager to allocate more resources to these potentially defective modules. Second, more accurate results may be achieved when statistical analysis methods are used with bug-tracking databases and metrics derived from software source code. As a bonus, it provides an objective baseline for comparing the results of various statistical analysis algorithms for predicting faulty modules. Throughout this benchmark, the input dataset, the percentage of the training dataset, and the feature selection approach are all identical. In this study, it is advised that accuracy and error ratio rate values be combined to improve the prediction of software defects.

According to the following structure, the remainder of this paper will be: concerning extracting software fault models, section 2 analyses relevant work and emphasises the most significant techniques. Section 3 explains the research approach followed. Metrics, architecture, data gathering methods, and statistical analysis techniques are all part of this. Section 4 presents the scientific steps to building regression models. Section 5 presents the architecture solution to reveal defects in programming code. The results of using statistical analysis technologies on various projects are shown in section 6. Analysis and debate are given in detail. The paper comes to a close in section 7.

2. RELATED WORKS

Software defects are defined as a deficiency or problem that arises from utilising a software product that causes it to operate abnormally. Identifying and prioritising flaws has been the subject of various studies, including practical ones [25]. The ten best techniques (e.g., neural network and regression models) for reducing flaws were outlined in the programming code [26].

Defects may be seen as actual and visible progress before a new software modification is released to the public, whether the change is buggy or clean [27]. According to technical reports in software companies and following the Pareto principle, around 80% of the problems originate from 20% of the modules. In contrast, the other half of the modules are defect-free. Software defects cause operational failures. The more errors there are, the worse it becomes [28]. Faults during program execution may be discovered and corrected using the systemically performed unit and system tests. However, defects will inevitably be found once the program has been implemented. The cost of defect repair during operation is far higher than that of defect repair during development or testing. Factors of cost growth vary from 5 to 100 [29].

As a result of a vast number of issues that have been reported throughout time, models are employed in software development to predict how many defects may arise in the future. When using these dynamic models, the independent variable is the number of faults discovered during the early phases of the development process. These models, which estimate the total number of defects and the distribution of those problems over time, are referred to as defect prediction and estimation models [30]. Dynamic models are used to track the progress of issues after they have been identified. If a considerable number of faults exist in an application in development, they may result in difficulties in the final product that need a comprehensive investigation. This statistic is critical in determining whether a piece of software is suitable for consumer distribution. An older overview of these investigations, which is accessible online, may be found [31].

The Rayleigh and related curves are used in dynamic defect models [30], [31]. By beginning with the Rayleigh distribution, it demonstrated that the preceding assumption was no longer valid and argued for the use of a linear combination of Rayleigh curves to better fit the facts [30]. As discovered by the researchers [31], using COQUALMO software estimating models aided in properly allocating labour to satisfy quality objectives.

“Static defect models” are models for predicting faults in a software product or project based on the features and data collected about the product or project (i.e., measurements of software products) [25]. Software quality may be assessed by a range of parameters, including complexity, lines of code, volume, and size [21], [22]. Complexity, lines of code, volume, and size are only a few examples. As researchers [24] point out, many software metrics and statistical models rely primarily on size and complexity measures in order to predict and prevent issues. Their favoured approach for forecasting software defects was Bayesian belief networks (BBNs). Many researchers have subsequently utilised and improved Bayesian networks [28], [32]. A commercial software program that makes use of BBNs was shown. A framework for making software defect prediction was described [32] in order to assist project managers and act as a roadmap for future research into defect prediction.

Several types of defect model studies have been conducted, including regression models [19], [20], statistical models [11], [12], and machine learning-based models [17], [18]. To mention a few examples: artificial intelligence includes a variety of techniques such as neural networks and rule induction. A range of tactics is used in order to better predict unclear or missing data [28]. Regression and metric-based technologies, among other things, are examples of static models [19], [30]. With the help of the static analysis and dynamic instrumentation phases in SUDS, users may develop tools that take advantage of both paradigms [19], allowing them to construct dynamic bug detection tools.

It is possible to compensate for the lack of data on numerous parameters by employing approximation data sets [12], [13]. Researchers have [33] built many models that make use of (easily measurable qualities) in order to properly forecast the traits that are harder to measure. For example, to determine the most efficient mathematical models for a certain kind of software system or a given task, such as maintenance.

Models for defect detection are affected by noise. Using moving averages and exponential smoothing algorithms produces defect-occurrence predictions that are poorer than those obtained by just utilising the original data [22]. Researchers have [28] confirmed the presence of time-dependent variability in the accuracy of a bug prediction model.

The references show that the software defect prediction issue has two unresolved difficulties. This juncture is an excellent place to start with statistical models used to turn the available software repositories into knowledge and recommendations for the software development team. With the use of compilers and bug-tracking tools, the development team will be notified of any potentially problematic modules as soon as they are discovered. Second, the use of single detector models is not expected to lead to reliable predictions, according to this theory. To solve the issue of predicting malfunctioning software modules, a software system that incorporates several statistical analysis algorithms and compares them is needed.

3. METHOD

According to this new proposal, defects in software projects may be predicted using a statistical model. This section outlines a statistical model that may be used to anticipate software faults. Figure 1 depicts the suggested model’s detailed stages, which are as follows:

- i) Comprehensive literature searches for measurements, data sources, and mathematical and computational methodologies that may be used to anticipate software project failures at the field’s cutting edge.
- ii) The NASA data sets are accessed online, and the data is retrieved. For each of these reasons, we decided to use NASA data. Firstly, gathering large amounts of data from software businesses to identify software project flaws is difficult. Because of its enormous and high-quality data, NASA is the second choice. Software projects may benefit greatly from including static measurements and other factors in their testing. There is also a binary variable that indicates whether the module is broken.
- iii) In the pre-processing stage, it is vital to examine the data thoroughly and, if necessary, data transformation to correctly reveal the data’s valuable substance. Outlier removal, discretisation, handling missing values, a reduction in the number of variables, and/or a reduction in dimensionality are all techniques that may be used (adopting regression models).
- iv) Normalisation aims to convert dataset features to be on an identical scale. This step enhances the rendering and training constancy of the model (adopting the z-score method).
- v) Feature selection: in order to determine the most critical metrics and faults that will be utilised in the upcoming IST investigation, logistic regression and multivariate linear regression are employed. Make a logistic-to-multiple-linear regression mapping in order to obtain the final list of critical indicators that may be used to predict software project failures.
- vi) Build a statistical model (training stage) for anticipating software project flaws based on multiple linear and logistic regression, respectively.
- vii) Statistical model for testing and verification: it should be run over the whole dataset to determine whether the model accurately predicts software faults. It will also use various metrics such as accuracy, precision, recall, F1 measure, and the error rate to assess logistic and MLR and compare them to select the optimal one.
- viii) In the final stage, generate software defects report to help stakeholders in the software development field determine flaws in the software projects before delivering it to the clients finally, in order to save money, effort, and time to repair those defects that appear in the software after they are finally delivered.

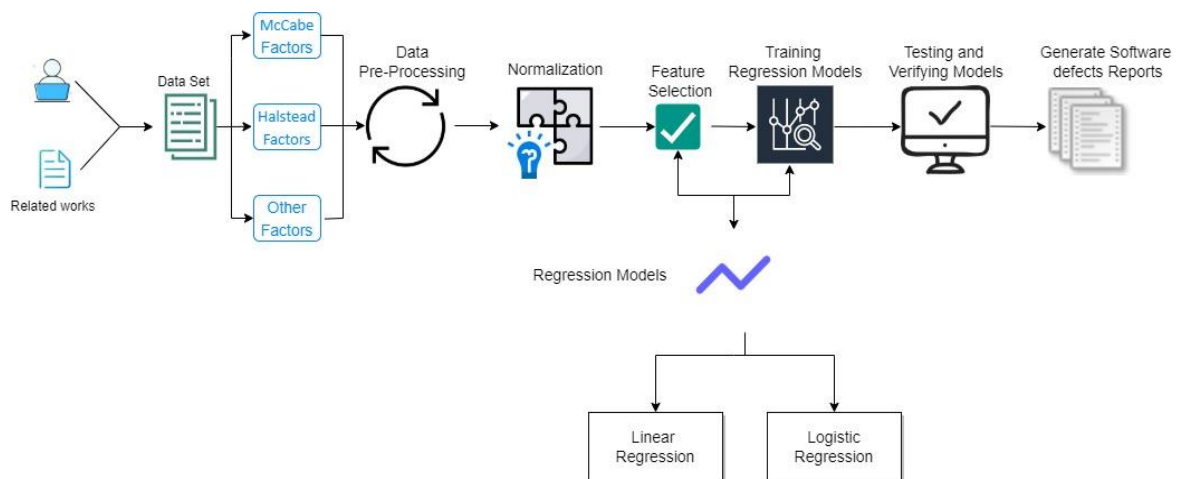


Figure 1. A proposed statistical model for detecting defects in software projects

4. THE PROPOSED ALGORITHMS OF STATISTICAL ANALYSIS

This section introduces two algorithms that are widely used in statistical analysis. These algorithms are binary logistic regression (BLR) and multiple linear regression (MLR). BLR is a technique that models the probability of a definite outcome, while MLR models the relationship between one dependent variable and multiple independent variables. Both algorithms have their strengths and weaknesses, and their selection depends on the nature of the problem and the data. The following sections will provide a detailed explanation of each algorithm and its application in different scenarios.

4.1. Algorithm of binary logistic regression

This part presents a BLR approach for finding the critical defect variables that impact software projects. The standard error (SE), the pseudo-R-squared (PRS), and the P-value (PV) are all crucial statistics in BLR. The SE is used to assess the accuracy of a sample distribution of the population [7]. The logistic regression model's power is shown via the usage of PRS [29]. Also, it determines the ratio of the effects of the independent factors on the dependent variable. A PV ratio shows an independent variable's impact on the dependent variable statistically. According to the following explanation, it is also a number between 0 and 1. The model rejects the null hypothesis if the PV is greater than 0.05 to put it in another method. Steps 3 to 14 represent the parameters of the BLR model. Steps 15 to 21 represent the construction of our model based on obtaining a little SE to ensure its accuracy. Steps 22 to 35 represent the evaluation of the proposed model based on PRS and PV. Finally, the final list of factors influencing software project defects has been selected. The following are the stages in the Algorithm 1, we have come up with:

Algorithm 1. BLR is an algorithm for identifying critical defects in software projects

1. Input: σ (the dependent variable degree to which software project fault variables have an impact)
x (independent variables Factors contributing to software project defects)
2. Output: β (list of the most common causes of software project defects)
3. BLR = Binary Logistic Regression
4. n = sample size
5. M_{full} = model with predictors
6. $M_{intercept}$ = model without predictors
7. $L(M)$ = estimated likelihood
8. \hat{A} = sample proportion
9. A_0 = Null hypothesis: percentage of the total population
10. F_i = the prediction value
11. \bar{Y} = the mean of Y_i
12. SE = Standard Error
13. PRS = Pseudo R- Square
14. PV = P-value
15. Construct the BLR model based on the group of α and σ
16. Start with random weights and biases: w_1, w_n, b
17. For every point (x_1, x_2, \dots, x_n) : do
18. For $i = 1, 2, \dots, n$ do
19. Update w_i' new weights
20. Update b' new biases
21. Repeat until the error is small
22. Evaluate BLR Model
23. Calculate the value of PRS. PRS is formulated as follows:
24.
$$PRS = 1 - \left\{ \frac{L(M_{full})}{L(M_{intercept})} \right\}^{2/n}$$
25. If (PRS is small)
26. Change the explanatory defect factors
27. Go to step 15
28. Else
29. Approve BLR Model
30. End If
31. Calculate the value of PV. PV is formulated as follows:
32.
$$PV = \frac{\hat{A} - A_0}{\sqrt{\frac{A_0(1-A_0)}{n}}}$$
33. If (PV > 0.05)
34. Refuse the other defect factors
35. Else
36. Approve β
37. End If
38. Return β

4.2. Multiple linear regression algorithm

This section offers a MLR analysis approach to detect the most crucial failure variables affecting agile software development initiatives. The SE, R-squared (RS), adjusted R-squared (ARS), and PV were introduced by multiple regression analysis. The SE first assesses the equation's ability to match the specimen data. The dependent variable's units of measurement are crucial. Regression coefficients RS measure the model's predictive power. ARS is a reworked version of RS that considers the model's additional predictors. Only if the innovative words improve the model more than expected do they increase in value. A predictor's impact

on a model falls if it boosts it less than expected. It's always lower than the RS, of course. The significance of statistical findings may be assessed with the use of PV.

One may think of it as something in the range of 0 to 1, and one can explain it like this: the null hypothesis is strongly refuted by a low PV (usually 0.05). We may conclude that the null hypothesis is false from the statistical data. Steps 3 to 14 represent the parameters of the MLR model. Step 15 represents the construction of our model based on the dependent variable and independent variables. Steps 16 to 31 represent the evaluation of the proposed model based on ARS and PV. Finally, the final list of factors influencing software project defects has been selected. The following are the stages in the algorithm we have developed. The Algorithm 2 stages are as follows:

Algorithm 2. MLR is an algorithm for identifying critical defects in software projects

1. **Input:** σ (the dependent variable degree to which software project fault variables have an impact)
 x (independent variables Factors contributing to software project defects)
2. **Output:** β (list of the most common causes of software project defects)
3. **MLR = Multiple linear regression**
4. SSR = Sum of Squares Regression
5. SST = Sum of Squares Total
6. n = The Number of Data Points
7. F_i = the prediction value
8. Y_i = the true value
9. \bar{Y} = the mean of Y_i
10. $SSR = \sum_i (F_i - \bar{Y})^2$
11. $SST = \sum_i (Y_i - \bar{Y})^2$
12. **SE = Standard Error**
13. ARS = Adjusted R- Squared
14. **PV = P-value**
15. Build the MLR model based on the set of α and σ
16. Estimate the MLR model
17. Check the value of SE. SE is calculated as follows:

$$SE = \sqrt{\frac{SSR}{n-2}}$$

18. Check the value of RS. RS is calculated as follows:

$$RS = 1 - \frac{SSR}{SST}$$

19. Check the value of ARS. ARS is calculated as follows:

$$ARS = 1 - \frac{SSR/(n-k-1)}{SST/(n-1)}$$

20. If (ARS < 0.5)
 21. Change the explanation of software defect factors
 22. **Go to step 15**
 23. **Else**
 24. **Approve MLR Model**
 25. **End If**
 26. Check PV for each variable to determine β
 27. **If (PV < 0.05)**
 28. Approve β
 29. **Else**
 30. Refuse the other failure factors
 31. **End If**
- Return β**

5. SOLUTION ARCHITECTURE

The proposed solution architecture is displayed in Figure 2. The repository data and software metrics are directed to statistical analysis techniques. The user should determine which software projects to include in the statistical analysis process and which to eliminate, based on their propinquity to the software project to be forecast. Two statistical techniques are launched, such as multiple linear and logistic regression. When the existing project development lifecycle begins, the source code in this project is estimated by the metrics estimator and directed with the bug tracking data to the predictor statistical analysis techniques. The outputs of applying the statistical analysis to the existing project data to forecast which modules are prospective to be faulted are directed to the development group. In addition, these faulted modules are passed to the programmer to fix them accurately. Finally, the programmer should deliver the software projects without any defects to save effort, team, money, and time.

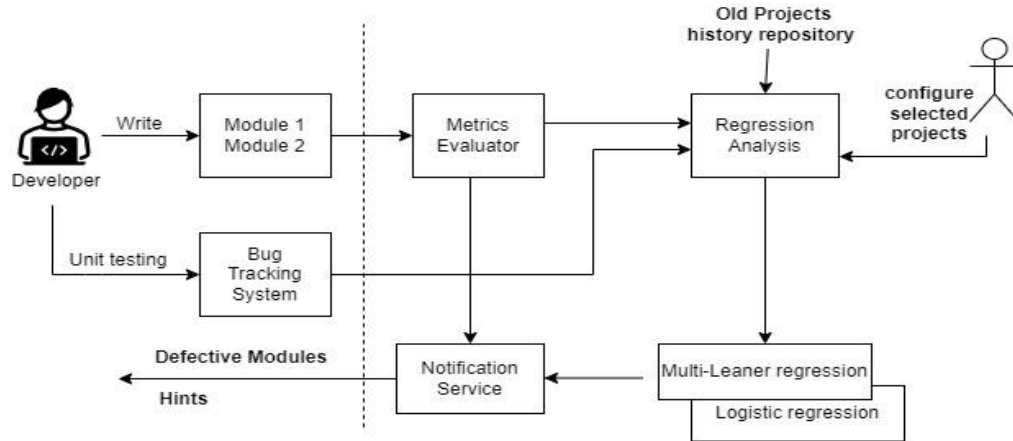


Figure 2. Solution architecture stage to detect defects in programming code

6. PERFORMANCE CRITERIA

The accuracy and SE rate are used to assess the performance of various statistical analysis models, as shown in (1) and (2) [9].

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

$$\text{Error Rate} = 1 - \text{Accuracy} \quad (2)$$

where:

TP=correctly identified data

TN=correctly rejected data

FP=incorrectly identified data

FN=incorrectly rejected data

7. RESULTS AND DISCUSSION

7.1. Multiple linear regression

We are undertaking research to determine which features may be beneficial in preventing software projects from failing. In this experiment, which uses MLR analyses, there are many independent variables (defect factors in software projects) and one dependent variable (the degree to which those factors affect the grade of those factors in software projects). In (3) [28] may be used to explain the MLR analysis in more detail:

$$Y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_{11}x_{11} + \varepsilon \quad (3)$$

where:

Y: is the degree of effecting the defeat factors in software projects

β_0 : is the y-intercept

β_i : is the regression coefficient

X_i : critical failure factors

ε : the random error term

A series of stages was used to carry out the application of the model in question. The requested data is broken down into dependent and independent variables as a starting point. In addition, it comprises 70% training and 30% testing data. The suggested model assumes a strong linear connection between the dependent and independent variables, and ARS is utilised to check this assumption. After that, the proposed model's quality is improved by L2 regularization.

This section offers a flowchart for discovering the essential aspects that affect software projects using MLR and LR analysis. Figure 3 depicts the suggested model's flowchart. The SE, RS, ARS, and PV were introduced in MLR and LR analysis (PV). SE supplies a first assessment of the equation's suitability for the data. The dependent variable's units of measure rely on it. RS shows the regression model's explanatory power. ARS is a reworked version of RS that considers the model's additional predictors. When relative terms

encourage the model more than would be anticipated, it rises to the surface. Predictors that promote the model less than predicted are less likely to have an impact on the model's accuracy. It is always lower than the RS, of course. MLR and LR have an ARS of 0.78. The significance of the statistical findings may be established by using PV.

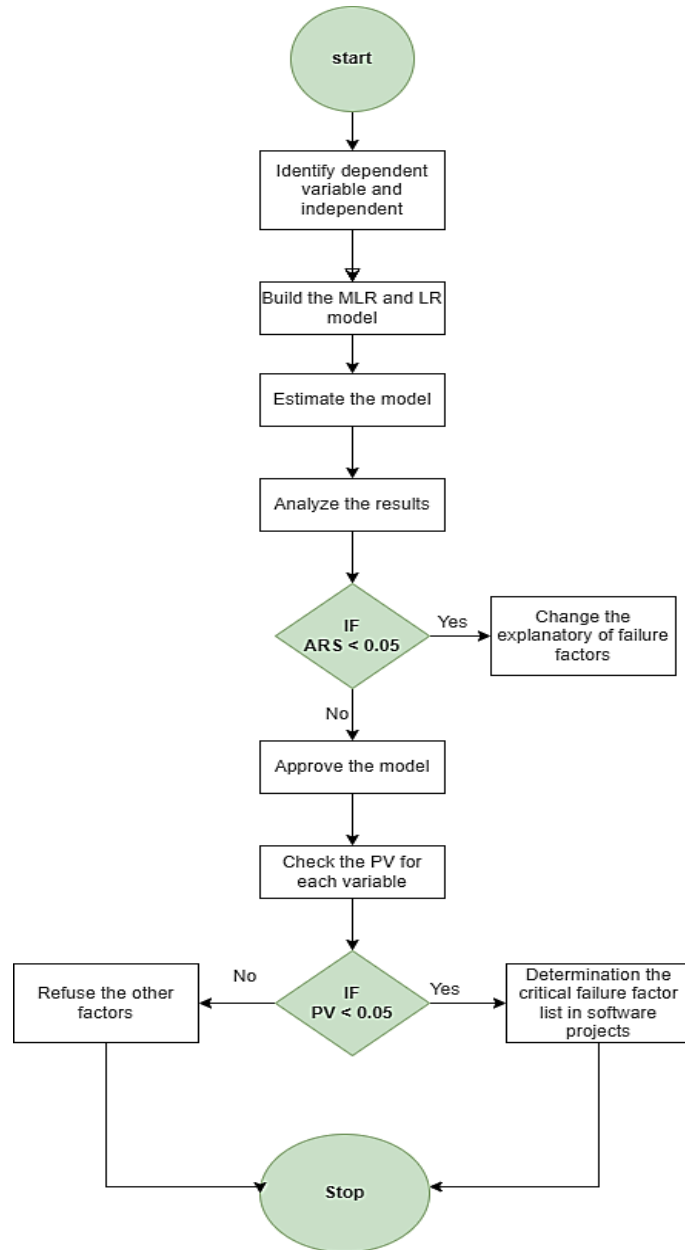


Figure 3. The proposed scientific steps for the MLR and LR model

According to the following formula, its value ranges from 0 to 1. Powerful evidence against the null hypothesis may be shown by the modest PV (usually less than 0.05). The statistical findings rule out the null hypothesis, the final list of critical factors that impact software projects in MLR, as shown in Table 1.

Accuracy, SE, and the model-based premier list of software defect factors (PLSDF) are given in Figures 4 and 5 for the MLR model based on critical defect factors (CDF). Two models have been created, which are MLR-PLSDF and MLR-CDF, and compared in terms of accuracy and SE rate. In addition, the accuracy in MLR-PLSDF and MLR-CDF is 0.79 and 0.82, respectively. Moreover, the SE rate in those models is 0.28 and 0.26, respectively. Therefore, the MLR-CDF model outperforms the MLR-PLSDF model in terms of accuracy and SE rate.

Table 1. The final list of critical factors that impact software projects in MLR

No	Factor ID		P-value
1	loc	✓	0
2	v(g)	✓	0
3	ev(g)	X	0.4341
4	iv	X	0.0537
5	n	✓	0
6	v	X	0.4804
7	l	X	0.2107
8	d	✓	0.0002
9	i	✓	0.0075
10	e	X	0.9454
11	b	X	0.7833
12	t	X	0.9454
13	IOCode	✓	0
14	IOComment	✓	0.0461
15	IOBlank	X	0.0809
16	locCodeAndComment	X	0.0667
17	Column1op	✓	0
18	Column2opnd	X	0.2169
19	Column1totalopnd	✓	0.0001
20	Column1totalop	✓	0.0003
21	Column1branch	✓	0

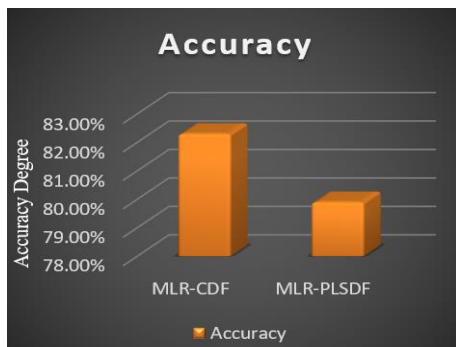


Figure 4. The comparison between MLR-CDF model and MLR-PLSDF in terms of accuracy

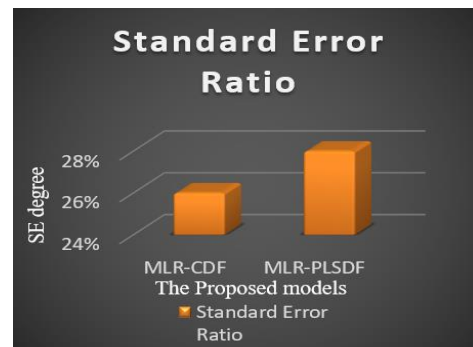


Figure 5. The comparison of the MLR-CDF model and MLR-PLSDF in terms of SE ratio

7.2. Logistic regression

During our experiment, we want to demonstrate the effectiveness of our suggested method. Logistic regression is used to carry out the strategy's implementation. The suggested model is implemented via a series of pre-processing processes. Defect factors in software projects are broken down into two different variables, one of which will be used as a dependent variable: the degree to which these factors affect the project. It is likewise divided into 70% training data and 30% test data. Third, the dependent variable was transformed from having two possible values (true or false) into just having two possible values (true or false) (0:1). For the fourth time, the independent variables were all set to range from 0 to 1. in this case, let's suppose that (as indicated in (4)) [28].

$$X_{\text{new}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \quad (4)$$

Logistic regression was used to identify the most important types of defects that have the greatest effect on software projects. It may be done in two ways. The first technique is based on software projects' LR-CDFs (logistic regression-critical defect factors). The second model is based on a list of the most common causes of software failure (LR-PLSDF). Because of this stable connection, the PLSDF approach may be used to analyse a wide range of software projects, even those with a relatively small number of defect factors in (5) [28]. Here is a breakdown of the P LR-CDFs:

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_{13} x_{13})}} \quad (5)$$

where:

P=degree of effecting defect factors in software projects

β_0 =P-intercept

β_i =regression coefficient

X_i =LR-CDF

There are critical statistical findings, as shown in Table 2. There are two significant findings in this article (ARS and PV). The ARS value is now 0.86. The PV highlights the main characteristics that impact the detection of flaws in software projects, as shown in Table 2. A degree must have a PV of more than 0.05 to be considered statistically insignificant. According to the PV of (v(g)=0.5970), this feature should be rejected since it is more than 0.05. In order to choose defect factors, those characteristics having a value level (PV less than 0.05) would be referred to as elects.

Table 2. The final list of critical factors that impact software projects in LR

No	Factor ID		P-value
1	loc	✓	0.0000
2	v(g)	X	0.5970
3	ev(g)	✓	0.0267
4	iv	✓	0.0447
5	n	X	0.1740
6	v	X	0.6973
7	l	✓	0.0003
8	d	✓	0.0072
9	i	✓	0.0084
10	e	X	0.9994
11	b	X	0.7338
12	t	X	0.9995
13	IOCode	✓	0.0001
14	IOComment	✓	0.0047
15	IOBlank	✓	0.0111
16	locCodeAndComment	X	0.0747
17	Column1op	✓	0.0032
18	Column2opnd	✓	0.0000
19	Column1totalopnd	✓	0.0016
20	Column1totalop	✓	0.0437
21	Column1branch	X	0.1599

Two models have been created, LR-PLSDF and LR-CDF, and compared in terms of accuracy and SE rate. In addition, accuracy in LR-PLSDF and LR-CDF is 0.83 and 0.86, respectively. Moreover, the SE rate in those models is 0.25 and 0.22, respectively. Therefore, the LR-CDF model outperforms the LR-PLSDF model in terms of accuracy and SE rate, as shown in Figures 6 and 7. The comparison between the proposed statistical models in terms of accuracy and SE ratio is in Figures 8 and 9. In addition, these figures indicate that the LR-CDF model outperforms MLR-PLSDF, MLR-CDF, and LR-PLSDF inaccuracy by 7%, 4%, and 3%, respectively, and the SE rate by 6%, 4%, and 3% respectively.

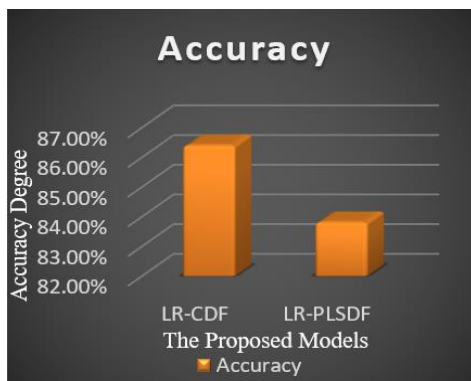


Figure 6. The comparison between a model of LR-CDF and a model of LR-PLSDF for accuracy

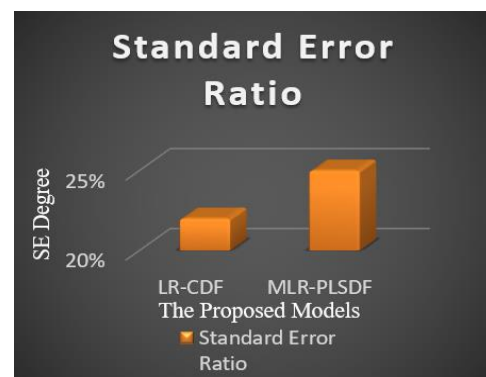


Figure 7. The comparison between a model of LR-CDF and a model of LR-PLSDF to SE ratio

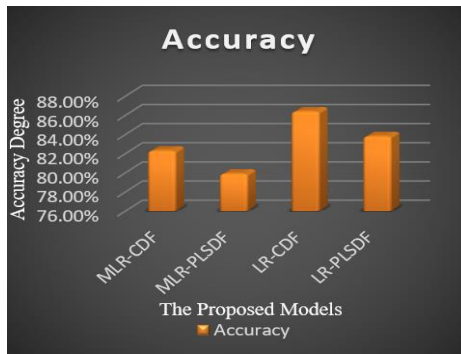


Figure 8. The accuracy comparison of all proposed statistical model

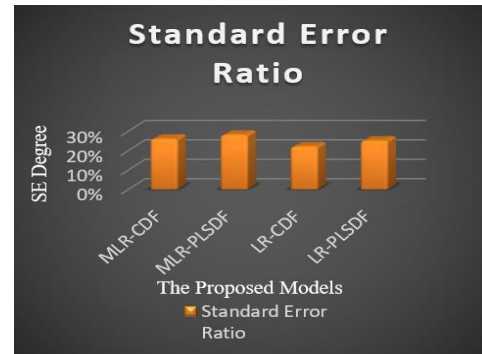


Figure 9. The SE ratio comparison of all proposed statistical model

It is demonstrated in Figure 10 that the LR-CDF model beats the state-of-the-art in prior studies in terms of accuracy. Sharma and Chandra [18] research, the LR-CDF model surpasses the intelligence approaches (association rule, decision tree, Naive Bayes, and neural network) in terms of accuracy by 9.1%, 10.3%, 13.1%, and 13.1%, respectively.

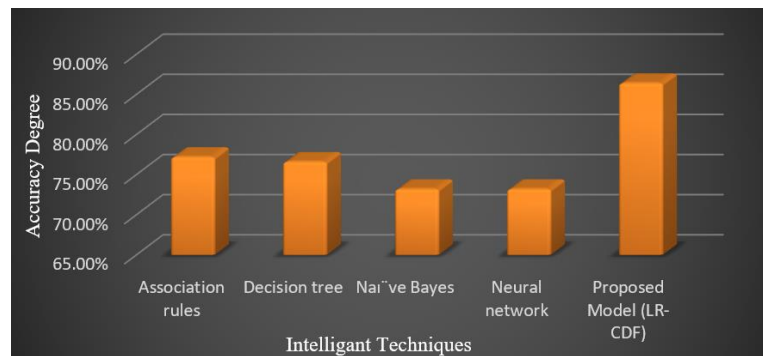


Figure 10. The comparison between the proposed model and the state-of-the-art method

8. CONCLUSION

Three important topics were addressed in our study, and they are as follows: to begin, we looked at the factors that influence the occurrence of errors throughout the software development process. Second, we concentrated on identifying the production procedures utilised in the case under consideration. The next stage was to determine the criteria for evaluating the techniques. Statistical and intelligent ways to uncover defects in software projects, which is this article's focus, have room for improvement. A statistical model was used to construct an altogether new method to predict defects in software projects, which is described in detail below.




This study demonstrated that some components have the greatest influence on finding software flaws. These four methodologies are utilised for statistical analysis: MLR-CDF, MLR-PLSDF, LR-CDF, and LR-PLSDF. In terms of accuracy and SE, it is evident that LR-CDF outperforms all the previous techniques that have been proposed. The LR-CDF outperforms existing techniques in terms of accuracy by 9.1%, 10.3%, 13.1%, and 13.1%, respectively, compared to the current approaches (association rule, decision tree, Naive Bayes, and neural network). The research presented in this study is limited to scientific articles published up to 2020, and does not account for new and innovative approaches that may be incorporated in 2021 and 2022. However, it is important to note that more intelligent approaches may lead to more accurate discovery of defects in diverse software projects. As a prospect for future research, it is suggested to investigate various approaches to enhance model accuracy and identify crucial factors that indicate defects in software projects. This direction of research would offer valuable insights into the optimization of software development processes and the improvement of overall software quality. These recent research trends, such as long short-term memory, convolution neural networks, and deep forests, have been found to improve the accuracy of research aimed at enhancing the proposed model and state-of-the-art method in previous works. This study proposes processing revealed defects in software projects using optimisation and deep learning techniques.

REFERENCES




- [1] R. Hewett, "Mining software defect data to support software testing management," *Applied Intelligence*, vol. 34, no. 2, pp. 245–257, Sep. 2011, doi: 10.1007/s10489-009-0193-8.
- [2] G. J. Liu and W. Y. Wang, "Research on an educational software defect prediction model based on SVM," in *Entertainment for Education. Digital Techniques and Systems: 5th International Conference on E-learning and Games*, 2010, vol. 6249 LNCS, pp. 215–222, doi: 10.1007/978-3-642-14533-9_22.
- [3] M. Jureczko and L. Madeyski, "Cross-project defect prediction with respect to code ownership model: an empirical study," *E-Informatica Software Engineering Journal*, vol. 9, no. 1, pp. 21–35, 2015, doi: 10.5277/e-Inf150102.
- [4] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4–5, pp. 531–577, Aug. 2012, doi: 10.1007/s10664-011-9173-9.
- [5] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, Jan. 2010, doi: 10.1016/j.jss.2009.06.055.
- [6] A. Abdelaziz, N. R. Darwish, and H. A. Hefny, "Towards a machine learning model for predicting failure of agile software projects," *International Journal of Computer Applications*, vol. 168, no. 6, pp. 20–26, Jun. 2017, doi: 10.5120/ijca2017914466.
- [7] M. Maddeh, S. Ayouni, S. Alyahya, and F. Hajje, "Decision tree-based design defects detection," *IEEE Access*, vol. 9, no. 2, pp. 71606–71614, 2021, doi: 10.1109/ACCESS.2021.3078724.
- [8] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on the interpretation of defect models," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 320–331, Feb. 2021, doi: 10.1109/TSE.2019.2891758.
- [9] A. A. Bangash, H. Sahar, A. Hindle, and K. Ali, "On the time-based conclusion stability of cross-project defect prediction models," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5047–5083, Sep. 2020, doi: 10.1007/s10664-020-09878-9.
- [10] S. Feng *et al.*, "COSTE: complexity-based over sampling technique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology*, vol. 129, p. 106432, Jan. 2021, doi: 10.1016/j.infsof.2020.106432.
- [11] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3590–3638, Aug. 2020, doi: 10.1007/s10664-020-09848-1.
- [12] S. Morasca and L. Lavazza, "On the assessment of software defect prediction models via ROC curves," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3977–4019, Aug. 2020, doi: 10.1007/s10664-020-09861-4.
- [13] S. Patil and B. Ravindran, "Predicting software defect type using concept-based classification," *Empirical Software Engineering*, vol. 25, no. 2, pp. 1341–1378, Feb. 2020, doi: 10.1007/s10664-019-09779-6.
- [14] J. A. Moral-Muñoz, E. Herrera-Viedma, A. Santisteban-Espejo, and M. J. Cobo, "Software tools for conducting bibliometric analysis in science: an up-to-date review," *Profesional de la Informacion*, vol. 29, no. 1, p. 20, Jan. 2020, doi: 10.3145/epi.2020.ene.03.
- [15] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, no. 5, p. 106287, Jun. 2020, doi: 10.1016/j.infsof.2020.106287.
- [16] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2107–2145, Aug. 2016, doi: 10.1007/s10664-015-9396-2.
- [17] A. H. Yousef, "Extracting software static defect models using data mining," *Ain Shams Engineering Journal*, vol. 6, no. 1, pp. 133–144, Mar. 2015, doi: 10.1016/j.asej.2014.09.007.
- [18] D. Sharma and P. Chandra, "Identification of latent variables using factor analysis and multiple linear regression for software fault prediction," *International Journal of System Assurance Engineering and Management*, vol. 10, no. 6, pp. 1453–1473, Oct. 2019, doi: 10.1007/s13198-019-00896-5.
- [19] M. Sirshar, K. Amir, H. Mir, and L. Zainab, "Comparative analysis of software defect prediction techniques," *Preprints*, 2019.
- [20] V. S. Sukanya and S. Saraswathy, "An enhanced evolutionary model for software defect prediction," *International Journal of Engineering Science and Computing*, vol. 7, no. 10, pp. 15323–15328, 2017.
- [21] D. Verma and S. Kumar, "Prediction of defect density for open source software using repository metrics," *Journal of Web Engineering*, vol. 16, no. 3–4, pp. 294–311, 2017.
- [22] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: multi-objective effort-aware just-in-time software defect prediction," *Information and Software Technology*, vol. 93, no. 2, pp. 1–13, Jan. 2018, doi: 10.1016/j.infsof.2017.08.004.
- [23] M. K. Dhillon, P. Singh, and P. Singh, "Empirical model for fault prediction on the basis of regression analysis," *International Journal of Science and Research (IJSR)*, vol. 5, no. 6, pp. 163–168, Jun. 2016, doi: 10.21275/v5i6.NOV164139.
- [24] A. N. Mahmoud and V. Santos, "Statistical analysis for revealing defects in software projects: systematic literature review," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 11, pp. 237–249, 2021, doi: 10.14569/IJACSA.2021.0121128.
- [25] E. A. Felix and S. P. Lee, "Integrated approach to software defect prediction," *IEEE Access*, vol. 5, no. 1, pp. 21524–21547, 2017, doi: 10.1109/ACCESS.2017.2759180.
- [26] P. He, Y. He, L. Yu, and B. Li, "An improved method for cross-project defect prediction by simplifying training data," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–18, Jun. 2018, doi: 10.1155/2018/2650415.
- [27] M. Jorayeve, A. Akbulut, C. Catal, and A. Mishra, "Machine learning-based software defect prediction for mobile applications: a systematic literature review," *Sensors*, vol. 22, no. 7, p. 2551, Mar. 2022, doi: 10.3390/s22072551.
- [28] N. R. Darwish, A. A. Mohamed, and A. S. Abdelghany, "A hybrid machine learning model for selecting suitable requirements elicitation techniques," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 14, no. 6, pp. 380–391, 2016.
- [29] A. Abdelaziz, N. R. Darwish, and H. A. Hefny, "Multiple linear regression for determining critical failure factors of agile software projects," *International Journal of Intelligent Engineering and Systems*, vol. 12, no. 3, pp. 244–255, Jun. 2019, doi: 10.22266/IJES2019.0630.24.
- [30] M. Pandit *et al.*, "Towards design and feasibility analysis of DePaaS: AI based global unified software defect prediction framework," *Applied Sciences (Switzerland)*, vol. 12, no. 1, p. 493, Jan. 2022, doi: 10.3390/app12010493.
- [31] A. Abdelaziz, N. R. Darwish, and S. A. Mazen, "A proposed approach for revealing failure of agile software projects," *Artificial Intelligent Systems and Machine Learning*, vol. 10, no. 10, 2018.
- [32] L. Q. Chen, C. Wang, and S. L. Song, "Software defect prediction based on nested-stacking and heterogeneous feature selection," *Complex and Intelligent Systems*, vol. 8, no. 4, pp. 3333–3348, Feb. 2022, doi: 10.1007/s40747-022-00676-y.
- [33] S. Zhang, S. Jiang, and Y. Yan, "A software defect prediction approach based on BiGAN anomaly detection," *Scientific Programming*, vol. 2022, no. 3, pp. 1–13, Apr. 2022, doi: 10.1155/2022/5024399.

BIOGRAPHIES OF AUTHORS






Alia Nabil Mahmoud    is an assistant lecturer in Information Management, teaching “Information Systems” and “Data Science” courses in Information Management and Information Systems Degrees. She is a researcher at NOVA IMS-Nova University of Lisbon, Portugal. She holds a B.Sc. in Management Information Systems and Technologies from the Higher Technological Institute, Egypt. She earned an M.Sc. in Information Systems and Technologies Management from Nova IMS Universidade Nova de Lisboa. She is pursuing a Ph.D. in Information Systems and Technologies Management from Nova IMS. She can be contacted at email: m20190508@novaims.unl.pt.






Ahmed Abdelaziz    received his B.Sc. degree in Computer Sciences and Information Systems from the Sadat Academy in 2007, Egypt, his M.Sc. in Information Systems from Sadat Academy in 2013, and his Ph.D. in Information Systems from Mansoura University in 2019. He is a data science researcher at NOVA IMS in Lisbon, Portugal. He is mainly interested in machine learning applications, data mining, cloud computing, and knowledge discovery in big data. He worked as a lecturer in the Information Systems Department at the Higher Technological Institute in Cairo, Egypt. He has several publications in reputed and high-impact journals published by IEEE, Elsevier, Springer, and others. Currently, he is participating in the energy consumption in public buildings project in Portugal between NOVA IMS and ADENE Agency in Lisbon, Portugal. He can be contacted at email: d20190535@novaims.unl.pt.



Vitor Santos    is an assistant professor at NOVA Information Management School (NOVA IMS)-Nova University of Lisbon and the European University, teaching “Information Systems” and “Artificial Intelligence” courses in Computer Science and Informatics Engineering Degrees. Before that, he was an invited Professor Trás os Montes e Alto Douro University (UTAD) and Minho University (UM). He integrates several national and international conference scientific committees and has authored several academic publications (~100) (>40 IS projects). He is an elected member of the order of engineers and the APDSI board. He was the Microsoft Portugal Academic Computer Science Program Manager. Before that, he occupied senior management positions at Santander Bank Group Companies and has developed computer engineering activities for about 15 years. He holds a Ph.D. in Science and Information and Technology Systems from the University of Minho, a B.Sc. in Informatics Engineering from Cocite, a postgraduate course in Computer Science from the Faculty of Science at the Lisbon University, an M.Sc. in Information Systems Science from the University of Minho, a D.E.A. from the University of Minho and a computer specialist title from polytechnic institutes Guarda, Castelo Branco and Viseu. He is working on a second Ph.D. in Culture and Communication. He can be contacted at email: vsantos@novaims.unl.pt.



Mário M. Freire    received a five-year B.Sc. degree in Electrical Engineering and a two-year M.Sc. in Systems and Automation in 1992 and 1994, respectively, from the University of Coimbra, Portugal. He obtained his Ph.D. in Electrical Engineering in 2000 and the *Habilitation title* in Computer Science in 2007 from the University of Beira Interior (UBI), Portugal. He is a full professor of Computer Science at UBI, which he joined in the Fall of 1994. In April 1993, he did a one-month internship at the Alcatel-SEL Research Centre (now Nokia Networks) in Stuttgart, Germany. His main research interests fall within the area of computer systems and networks, including network and systems virtualisation, cloud and edge computing and security and privacy in computer systems and networks. He is the co-author of seven international patents, co-editor of eight books published in the Springer LNCS book series, and co-author of about 130 papers in international journals and conferences. He serves as a member of the editorial board of the ACM SIGAPP applied computing review, serves as associate editor of the Wiley Security and Privacy Journal and of the Wiley International Journal of Communication Systems, and served as editor of IEEE Communications Surveys and Tutorials in 2007–2011. He served as a technical program committee member for several IEEE international conferences and co-chairs the track on Networking of ACM SAC 2024. He is a chartered engineer by the Portuguese Order of Engineers and a member of the IEEE Computer Society and the Association for Computing Machinery. He can be contacted at email: mario@di.ubi.pt.