Statistical Analysis for Revealing Defects in Software Projects: Systematic Literature Review

Alia Nabil Mahmoud[1], Vítor Santos[2]
Information Management" Systems Management and Information Technologies"
NOVA IMS – Information Management School, Universidade Nova de Lisboa, Lisboa, Portugal

***Abstract*—Defect detection in software is the procedure to identify parts of software that may comprise defects. Software companies always seek to improve the performance of software projects in terms of quality and efficiency. They also seek to deliver the soft-ware projects without any defects to the communities and just in time. The early revelation of defects in software projects is also tried to avoid failure of those projects, save costs, team effort, and time. Therefore, these companies need to build an intelligent model capable of detecting software defects accurately and efficiently. The paper is organized as follows. Section 2 presents the materials and methods, PRISMA, search questions, and search strategy. Section 3 presents the results with an analysis, and discussion, visualizing analysis and analysis per topic. Section 4 presents the methodology. Finally, in Section 5, the conclusion is discussed. The search string was applied to all electronic repositories looking for papers published between 2015 and 2021, which resulted in 627 publications. The results focused on finding three important points by linking the results of manuscript analysis and linking them to the results of the bibliometric analysis. First, the results showed that the number of defects and the number of lines of code are among the most important factors used in revealing software defects. Second, neural networks and regression analysis are among the most important smart and statistical methods used for this purpose. Finally, the accuracy metric and the error rate are among the most important metrics used in comparisons between the efficiency of statistical and intelligent models.**

***Keywords*—Defects; software projects; statistical model; linear regression; logistic regression**

I. Introduction

Software companies aim to improve the quality of software projects in terms of their accuracy and efficiency. Software companies consume from 50% to 75% of the total budget of software projects in finding and fixing defects in those projects [1]. In the CHAOS report, many software projects vary in size (small, medium, and large projects) and, therefore, cost. These projects use many software development methods such as waterfall and agile. Several software projects failed due to the development and testing phase, as shown in Table I. A standard software development cycle has six phases, namely, planning, analysis, design, implementation, testing, and maintenance. In the development phase, developers modify source code that may lead to many defects in a software project. In modifications, developers should be careful not to produce any new defects in these projects. The testing phase is crucial to software projects. It is responsible for delivering the final project or product efficiently to customers without any defects and in time. Many factors, such as McCabe and Halstead, help developers find and fix defects in those projects, as shown in Table II. Nevertheless, there is difficulty in using these factors in medium and large-scale projects. Thus, developers need a statistical or intelligent model capable of predicting defects in software projects accurately and efficiently.

Many reasons lead to the failure of software development projects. These are the lack of experience of the project team, lack of knowledge of the code language, insufficient experience in the field, etc. Software defects in the development phase are among the most critical problems facing software companies because the many defects lead to those projects' failure. The avoidance of software defects is to gain clients' trust by providing a quality product. According to the CHAOS report, many software projects still fail because of the many reasons that have been mentioned earlier [2]. However, the direct reason for these projects' failure is the emergence of many software defects, as shown in Table I [2].

It was performed a compressive study about the relevant related work using PRISMA methodology. The PRISMA explanation gives the minimum set of items for detailing a precise audit. It comprises the four-phase flow diagram, which permits us to utilize the Clarification and Elaboration document to go through cases and clarifications and find the meaning and method of reasoning for each item on the checklist. For a clear understanding of PRISMA, perusing the Clarification and Elaboration document is unequivocally recommended. The PRISMA Stream Graph delineates the stream of data through the diverse stages of a Precise Audit. It maps out the number of records recognized, included, and prohibited and the reasons for avoidances.

TABLE I. CHAOS Report by Agile Versus Waterfall [2]

Size	Method	Successful	Challenged	Failed
All Size Projects	Agile (Scrum)	39%	52%	9%
	Waterfall	11%	60%	20%
Large Size Projects	Agile (Scrum)	18%	29%	53%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile (Scrum)	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile (Scrum)	58%	38%	4%
	Waterfall	44%	45%	11%

TABLE II. SOFTWARE METRICS OF MCCABE AND HALSTEAD TO REVEAL SOFTWARE DEFECTS [7]

Factor ID	Factor	Description
1	Loc	McCabe's line count of code
2	v(g)	McCabe ''cyclomatic complexity''
3	eV(g)	McCabe ''essential complexity''
4	Iv(g)	McCabe ''design complexity''
5	N	Halstead total operators + operands
6	V	Halstead ''volume''
7	L	Halstead ''program length''
8	D	Halstead ''difficulty''
9	I	Halstead ''intelligence''
10	E	Halstead ''effort'': effort to write program
11	B	Halstead ''Number of Delivered Bugs''
12	T	Halstead's time estimator: time to write program
13	LOCode	Halstead's line count
14	LOComment	Halstead's count of lines of comments
15	LOBlank	Halstead's count of blank line
16	LOCodeAndComment	Halstead's count of lines which contain both code and comments
17	uniq_Op	Unique operators
18	uniq_Opnd	Unique operands
19	total_Op	Total operators
20	total_Opnd	Total operands
21	branchCount	Of the flow graph
22	defects	Module has/has not one or more reported defects

Many researchers, such as [3] and others [4]; [5] have suggested many factors to detect software defects. However, to date, there is no formal study to determine the critical factors to help software companies detect software defects with a reasonable degree of accuracy. Most researchers such as [6] and others also used scientific methods and models to detect software defects, but these models were weak in accuracy and results. Thus, software companies need a formal study to determine the critical factors to build a statistical model capable of detecting software defects with high results and accuracy.

The paper is organized as follows. Section 2 presents the materials and methods, PRISMA, search questions, and search strategy. Section 3 presents the results with an analysis, and discussion, visualizing analysis and analysis per topic. Section 4 presents the methodology. Finally, in Section 5, we discuss the conclusion.

II. MATERIALS AND METHODS

The methodology is composed of three steps. First, PRISMA was used to find appropriate manuscripts in our research based on the manuscript title and the experimental results of the manuscripts. Second, bibliometric analysis was used to find the common terms that influence the revealing of software defects in terms of critical factors, performance metrics, and intelligent and statistical methods. Finally, the manuscripts were analyzed in detail to extract the most important factors and statistical methods used in detecting software defects and linking them to the results of the bibliometric analysis.

The systematic literature survey presents an evaluation of the scientific community's contributions to the topic of revealing software defects by using a rigorous and auditable methodology based on the PRISMA approach.

The PRISMA method is composed of five phases, as follows:

- Identification of relevant manuscripts of the domain or domains.
- Screening of titles, abstracts, papers without experiments, and position papers.
- Eligibility analysis.
- Full-text screening exclusion.
- Final papers to be analyzed in detail.

It was also adopted a bibliometric map; the bibliometric map is used to find the relationships between common software defects domain terms [8]. To this end, three phases were followed, evaluating the following quantities:

- Words frequency.
- Most common words.
- Frequency of these common words in the final manuscripts of the study.

By following PRISMA [9].this section is structured in the following way: (1) our research questions, (2) followed paper search strategy, (3) bibliometric map, (4) inclusion and exclusion criteria, and (5) final paper selection.

A. Research Questions

The study aims to provide a state-of-the-art review of current research efforts in revealing software projects. It was started by introducing the reader to specific topics concerning research objectives and employed methods. Particularly, the survey addresses the following research questions, aiming to identify the adoption techniques that have been applied in the overall domain of revealing software defects:

RQ1: What kinds of metrics have been adopted in software defects (SD)?

RQ2: Which statistical or intelligent techniques have been adopted for SD?

RQ3: What performance metrics have been adopted in the literature in the prediction of SD?

B. Search Strategy

A literature survey, generally, recommends searching several available journal and conference paper repositories to determine if similar work has already been performed, aiding in locating potentially relevant studies. The papers counted were searched in two electronic repositories, Scopus, and Web of Science. This study's covered topics were multidisciplinary, including, Software, Computer Science, Engineering, Mathematics, Environmental Science, Telecommunications, and Multidisciplinary Sciences. However, both repositories were used. The analysis showed that most of the publications from Web of Science were in Scopus as well. A repeated search process was performed to identify publications that have in their titles, abstracts, or keywords the following expressions: "software-defects" (or software defects, or defect or projects defects), and "machine learning" in Fig. 1.

Phase 1, the search string was applied to all electronic repositories looking for papers published between 2015 to 2021, which resulted in 627 publications.

Phase 2 followed a 5-step approach. In step 1, we excluded manuscripts based on titles (e.g., software defects, regression, and machine learning), which narrowed the set to 211 publications. In step 2, we excluded manuscripts based on abstracts screening, which resulted in 117 publications. In the following step 3, we excluded manuscripts reporting research without experiments, resulting in 83 publications.

Subsequently, in step 4 of phase 2, we excluded position manuscripts which gave us the final figure of 29 publications, as shown in Fig. 2.

(software-defects OR defect OR projects) AND (OR “data mining” OR forecasting OR “machine learning” OR “neural network" OR “clustering" OR “artificial intelligence” OR “prediction” OR “predictive” OR “statistical” OR analysis”)

Fig. 1. Search query for Scientific Manuscripts to Extract the Best Studies in Software Defects.

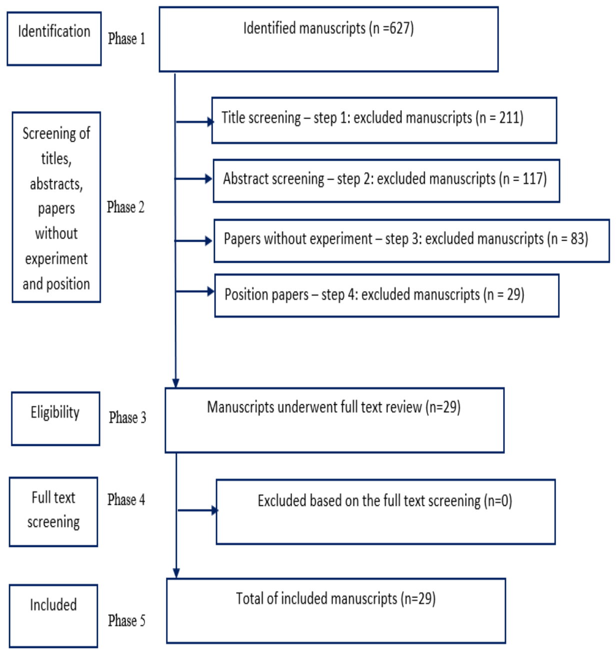
Identification
Phase 1
Identified manuscripts (n =627)
Screening of titles, abstracts, papers without experiment and position
Phase 2
Title screening – step 1: excluded manuscripts (n = 211)
Abstract screening – step 2: excluded manuscripts (n = 117)
Papers without experiment – step 3: excluded manuscripts (n = 83)
Position papers – step 4: excluded manuscripts (n = 29)
Eligibility
Phase 3
Manuscripts underwent full text review (n=29)
Full text screening
Phase 4
Excluded based on the full text screening (n=0)
Included
Phase 5
Total of included manuscripts (n=29)

Fig. 2. Scientific Steps for Analyzing the Proposed Manuscripts “PRISMA Flow Chart”.

In phase 3, manuscripts underwent a full-text reading and review, which lead to no exclusions (the result of phase 4).

As a result of our paper selection approach, the final list included 29 manuscripts (phase 5), analyzed in detail in this paper. These were further divided into the following four categories, as shown in Tables III and IV.

- Regression analysis studies to reveal Software Defects.
- Studies of Software Defects Prediction.

TABLE III. REGRESSION ANALYSIS STUDIES TO REVEAL SOFTWARE DEFECTS

No	Ref	Application	Dimensions	Method of Solution and Performance Metrics
1	S.N. Umar[10]	Software testing defect prediction model-a practical approach	Total number of test cases executed, test team size, allocated development effort, test case execution effort, and the total number of components delivered	Multiple linear regression. R square and standard error
2	(Dhiauddin & Ibrahim, 2012)[11]	A Prediction Model for System Testing Defects using Regression Analysis	Software complexity, test process, errors, the severity of the defect, and validity of defect	Multiple linear regression. Adjusted R square
3	E. A. FELIX and et al [12]	Integrated Approach to Software Defect Prediction	Defect acceleration, namely, the defect density, defect velocity, and defect introduction time	Statistical analysis. Adjusted R square and correlation coefficient
4	D. VERMA and et al [13]	Prediction of defect density for open source software using repository metrics	software size, number of developers, commits, and the total number of defects	Multiple linear regression. R square
5	D. Sharma and et al [14]	Identification of latent variables using factor analysis and multiple linear regression for software fault prediction	Coupling between object classes, depth of inheritance tree, lack of cohesion of methods, and weighted methods per class	Multiple linear regression. R square and Adjusted R square
6	O. Sari and et al [15]	Use of Logistic Regression Analysis for Bug Prediction	Weighted method count, depth of inheritance tree, lack of cohesion in methods, number of attributes, and number of methods	Logistic regression. Standard error
7	G. MAUSA and et al [16]	Software Metrics as Identifiers of Defect Occurrence Severity	Software size, number of code lines, and the total number of defects.	Correlation coefficients and logistic regression. Error rate

8	Peng H. and et al [17]	presented a model for predicting defects in software projects	Software size, number of code lines, and the total number of defects.	Logistic regression. Standard error
9	M. Dhillon and et al [18]	An empirical model for fault prediction on the basis of regression analysis	Weighted method count, depth of inheritance tree, lack of cohesion in methods, number of attributes, and number of methods	Logistic regression. Precision, recall, and f1 measure
10	X. Chen and et al [19]	Multi-Objective Effort-Aware Just-in-Time Software Defect Prediction	diffusion [Number of modified subsystems], size [line of codes], history [The number of unique changes to the modified files], and finally, experience [Developer experience].	Logistic regression. Accuracy

TABLE IV. STUDIES OF SOFTWARE DEFECTS PREDICTION

No	Ref	Application	Dimensions	Method of Solution and Performance Metrics
1	A. H. Yousef [7]	Extracting software static defect models using data mining	McCabe and Halstead metrics	Data mining techniques. Accuracy, Precision, Recall, and F1 score
2	Karuna P and et al [20]	Statistical analysis of metrics for software quality improvement	Violation of programming standards, error in data representation, error in design logic, and assorted error type	Statistical analysis. Mean and standard deviation
3	Sukanya. V and et al [21]	An enhanced evolutionary model for software defect prediction	McCabe and Halstead metrics	Enhanced genetic algorithm, genetic algorithm, and neural network. Precision
4	Y. Koroglu and et al [22]	Defect prediction on a legacy industrial software: a case study on software with few defects	Product and process metrics	Data mining techniques. AUC
5	L. KUMAR and et al [23]	An effective fault prediction model developed using an extreme learning machine with various kernel methods	Complexity, coupling, cohesion, and inheritance in the code	Extreme learning machine with various kernel methods (e.g., Linear kernel, Polynomial kernel, and Sigmoid kernel). Accuracy
6	F. Zhang and et al [24]	Towards building a universal defect prediction model	The weighted method programming language, issue tracking, total lines of code, total number of files, the total number of commits, and the total number of developers	K-mean clustering. AUC
7	A. Marandi and et al [25]	An approach of statistical methods for improving software quality	Post-delivery rework effort, actual effort, cost of the appraisal, cost of prevention, and cost of failure	Statistical analysis. Standard error
8	G. RajBahadur and et al [26]	The impact of using regression models to build defect classifiers	Object-oriented metrics	Linear regression, logistic regression, random forest, support vector machine, and neural network. AUC
9	S. Rathore and et al [27]	Predicting the number of faults in a software system using genetic programming	Total number of modules, number of lines of code, and number of faulty modules	Genetic programming. Recall and error rate
10	M. Sirshar and et al [28]	Comparative Analysis of Software Defect Prediction Techniques	Product and process metrics	Neural Network, Naive Bayes, Deep Forest technique. Error rate
11	M. Rawat and et al [29]	Software defect prediction models for quality improvement: a literature study	Object-oriented code, product, and process metrics	Regression models. Accuracy
12	S. Feng and et al [30]	Complexity-based Oversampling Technique to alleviate the class imbalance problem in software defect prediction	Line of code, number of children, and weighted method per class	Complexity-based Oversampling. Error rate
13	S. Patil and et al [31]	Predicting software defect type using concept-based classification	Interface, syntax, and standard [build-config-install]	Concept-based Classification. F1 score
14	J. Jiarpakdee and et al [32]	The impact of automated feature selection	inconsistent and correlated	Automated Spearman correlation. Error rate

		techniques on the interpretation of defect models		
15	A. Bangash and et al [33]	On the time-based conclusion stability of cross-project defect prediction models	Time, types of the projects, software development process	Mathews Correlation Coefficient. F-score
16	S. Morasca and et al [34]	On the assessment of software defect prediction models via ROC curves	Lines of code and complexity	Receiver Operating Characteristic. Error rate

III. Results, Analysis, and Discussion

This section introduces two main parts, which are bibliometric analysis and analyzing previous works in detail. The first part shows the relationships between common terms in intelligence, statistical techniques, and performance metrics used in the previous study. The second part seeks to find the scientific gap between proposed manuscripts in this study to build a novel model to overcome the issues for revealing defects in software projects.

A. *Visualizing Analysis*

It was used VOS viewer ("VOS viewer," n.d.), a Visualizing bibliometric network, to find common terminology in two areas: software defects and statistical techniques, across the 29 manuscripts under analysis. This tool supported the study with visual information enabling us to explore the relations between the domains of software defects and statistical techniques. Moreover, it helped to find the most common dimensions, clustering, and variety techniques able to answer the research questions.

Fig. 3 represents the visualization of a network map that displays the relations between the most popular terminology, how it is linked. The larger node represents the popular terminology in manuscripts, and the size of it represents the number of times these words appeared in manuscripts. VOS viewer splits the terminology into clusters according to the relevance concerning each other.

It was performed the analysis on the title and abstract using a binary counting method of 759 examined keywords with a minimum threshold of 2 occurrences, resulting in 57 terminologies, as shown in the figure. The largest nodes representing the important nodes of each cluster in the network map are determined as" Regression" (red), "cluster" (yellow), "software engineering" (green), "neural network" (blue), and finally "software defect prediction" (purple).

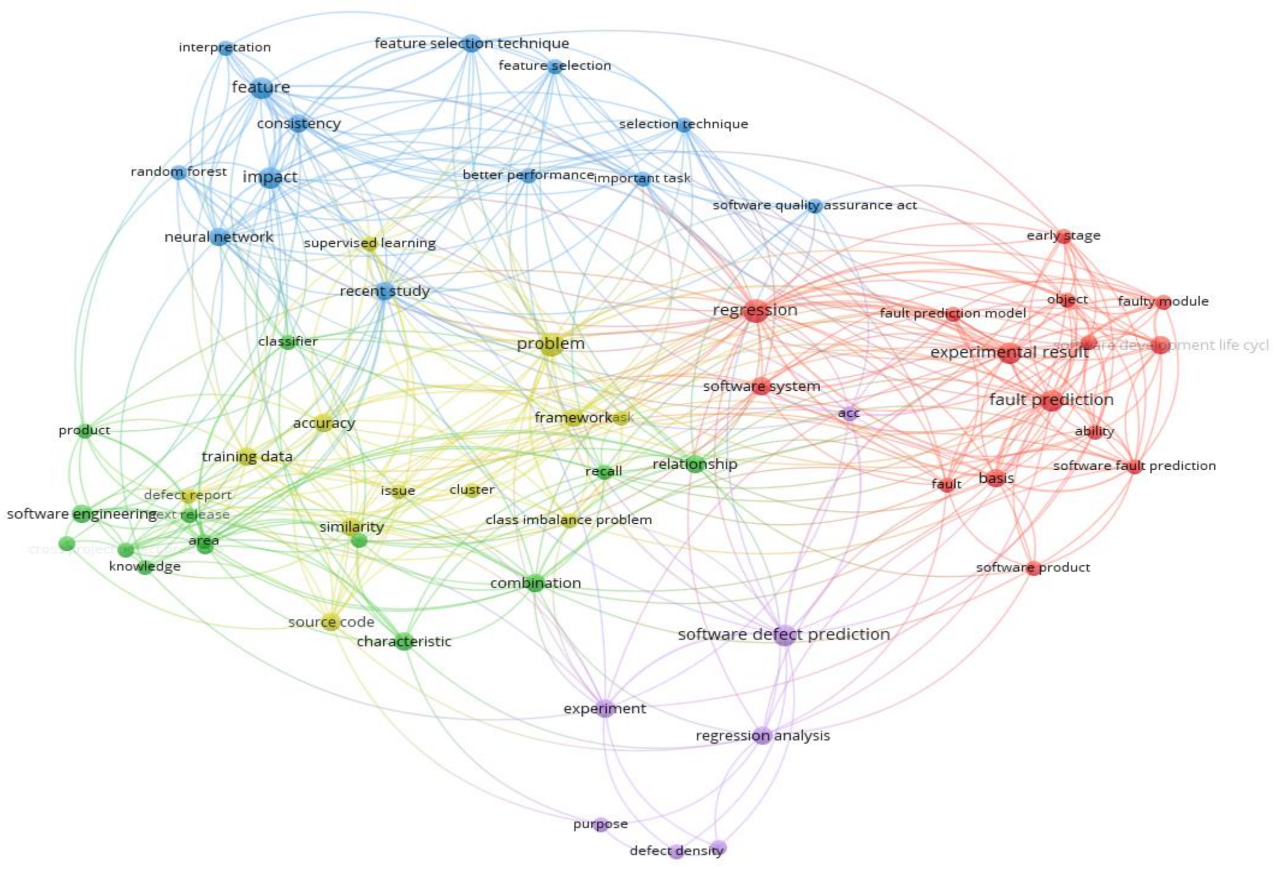

Fig. 3. The Relationships between the Common Terms using the Bibliometric Map.

Looking closer at the network map in Fig. 1, we can see that the 5 clusters are connected between them; for instance, the "regression" term is connected to "fault prediction model" in the same red cluster, it connected to "cluster" and "accuracy" in the yellow cluster, it is also connected to "software engineering" and "recall" in the green cluster. Finally, it is also connected to "neural network" and "feature selection" in the blue cluster; it is also connected to "software defect prediction "and "defect density." Besides, the term "software engineering" in the green cluster is connected to "cluster" in the yellow cluster, "regression" in the red cluster, and "neural network" in the blue cluster. Moreover, the terms "random forest" and "feature selection" are connected to "neural network" in the blue cluster, "recall" and "software engineering" in the green cluster, "cluster" in yellow cluster, "regression" and "fault prediction model" in the red cluster and "software defect prediction" and "defect density" in the purple cluster.

Finally, by analyzing the network map in Fig. 1 was possible to identify the important terms in each cluster, as follows:

- In the red cluster: "regression" and "software prediction model."
- In the yellow cluster: "cluster" and "accuracy."
- In the green cluster: "recall" and "software engineering."
- In the blue cluster: "random forest", "feature selection" and "neural network"
- In the purple cluster: "software defect prediction" and "defect density."

B. Analysis per Topic

RQ1 drove to look for metrics, data sources, and critical factors able to reveal software defects. Our review of papers S1 to S26 allowed us to extract such critical factors. Dimensions such as software status [No. of defects], OOP [Depth of Inheritance Tree and No. of Methods], McCabe Metrics [Line Count of Code], and Halstead Metrics [Effort to Write Program and Time to Write Program] seem to be highly considered when studying the revealing of software defects in software companies. Table V shows the variety of metrics used in predicting defects in software projects. The studies of S1, S4, and S16 relied on team dimension (team size and the number of developers) to predict software defects in software projects. The studies of S2, S3, S4, S7, S8, S12, S15, and S26 relied on software status dimensions (software complexity, number of defects, and software size) to detect defects in those projects. Moreover, the studies of S5, S6, S9, S15, S16, S18, and S21 relied on the OOP dimension (coupling between object classes, depth of inheritance tree, number of methods) also to reveal defects in those projects. Also, the studies of S7, S8, S10, S11, S13, S16, S19, S22, and S26 relied on McCabe metrics (line count of code, cyclomatic complexity, essential complexity, and design complexity) to find the optimal intelligent techniques to predict defects in software projects. Finally, the studies of S1, S3, S11, S13, S16, S17, S25 relied on Halstead Metrics (total operators + operands, effort to write the program, number of delivered bugs, count of lines of comments, and time to write a program) to forecast defects in various software projects. We observed that four factors are the most used in predicting defects in software projects. These are the number of defects, depth of inheritance tree, number of methods, and line count of code.

TABLE V. MAJOR FACTORS IN SOFTWARE DEFECT PROJECTS

Dimensions																			
		Team		**Software status**			**OOP**			**McCabe Metrics**				**Halstead Metrics**					**Other Factors**
		Size	No. Developers	software complexity	No. of Defects	Software Size	Coupling between Object classes	Depth of Inheritance Tree	No. of Methods	Line Count of Code	Cyclomatic Complexity	Essential Complexity	Design Complexity	Total Operators + Operands	The effort to Write Program	Number of Delivered Bugs	Count of Lines of Comments	Time to Write Program	
S.N. Umar [10] S1	Software testing defect prediction model-a practical	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	✓
M.D. Suffian and et al S2 [11]	A Prediction Model for System Testing Defects using Regression Analysis	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓

E. A. FELIX and et al. S3 [12]	Integrated Approach to Software Defect Prediction	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓
D. VERMA and et al. S4 [13]	Prediction of defect density for open source software using repository metrics	-	✓	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
D. Sharma and et al. S5 [14]	Identification of latent variables using factor analysis and multiple linear regression for software fault prediction	-	-	-	-	-	✓	✓	✓	-	-	-	-	-	-	-	-	-	✓
O. Sari and et al. S6 [15]	Use of Logistic Regression Analysis for Bug Prediction	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-
G. MAUSA and et al. S7 [16]	Software Metrics as Identifiers of Defect Occurrence Severity	-	-	-	✓	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-
Peng H. and et al. s8 [17]	presented a model for predicting defects in software projects	-	-	-	✓	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-
M. Dhillon and et al s9 [18]	An empirical model for fault prediction on the basis of regression analysis	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	✓
X. Chen and et al. s10 [19]	An empirical model for fault prediction on the basis of regression analysis	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓
A. H. Yousef s11 [7]	Extracting software static defect models using data mining	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Karuna P and et al. s12 [20]	Statistical analysis of metrics for software quality improvement	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓

Sukanya.V and et al s13 [21]	An enhanced evolutionary model for software defect prediction	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	-
Y. Koroglu and et al s14 [22]	Defect prediction on a legacy industrial software: a case study on software with few defects	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
L. KUMAR and et al. s15 [23]	An effective fault prediction model developed using an extreme learning machine with various kernel methods	-	-	✓	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-
F. Zhang and et al. s16 [24]	Towards building a universal defect prediction model	-	✓	-	-	-	-	-	✓	✓	-	-	-	-	-	-	✓	-	-
A. Marandi and et al s17 [25]	An approach of statistical methods for improving software quality	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	✓
G. RajBahadur and et al s18 [26]	The impact of using regression models to build defect classifiers	-	-	-	-	-	✓	✓	✓	-	-	-	-	-	-	-	-	-	-
S. Rathore and et al. s19 [27]	Predicting the number of faults in a software system using genetic programming	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
M. Sirshar and et al. s20 [28]	Comparative Analysis of Software Defect Prediction Techniques	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
M. Rawat and et al. s21 [29]	Software defect prediction models for quality improvement: a literature study	-	-	-	-	-	✓	✓	✓	-	-	-	-	-	-	-	-	-	✓

S. Feng and et al. s22 [30]	Software defect prediction models for quality improvement: a literature study	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓
S. Patil and et al. s23 [31]	Software defect prediction models for quality improvement: a literature study	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
J. Jiarpakdee and et al s24 [32]	Software defect prediction models for quality improvement: a literature study	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
A. Bangash and et al. s25 [33]	Software defect prediction models for quality improvement: a literature study	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓
S. Morasca and et al s26 [34]	Software defect prediction models for quality improvement: a literature study	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-

While addressing RQ2, we examined the techniques applicable in predicting defects in software projects. With this goal, we analyzed manuscripts S1 to S26 and noticed that techniques such as multiple linear regression, logistic regression, and machine learning are the most adopted, as shown in Table VI. Moreover, multiple linear regression was adopted by 23% of the analyzed manuscripts, whereas statistical analysis and data mining were the choices in 27% of manuscripts. Logistic regression accounted for 27% of the revised manuscripts. Also, machine learning techniques accounted for 19% of the revised manuscripts. Finally, the remaining 4% corresponded to the other intelligent techniques. We noticed four points.

Firstly, the studies (S1, S2, S4, S5, and S21) relied on multiple linear regression where S1 presented a model to predict defects in software projects to enhance the quality of software testing. This study seeks to find a suitable model to predict software defects to save effort, costs, and software companies' time. The results of this study show that R square and standard errors are 0.91 and 5.90%, respectively. S2 presented a model for predicting defects in software projects to improve the testing process in those projects. Besides, the adjusted R square in multiple linear regression is 90%. S4 presented a framework to predict defect density in open-source software projects. The results of this study show that the R square in multiple linear regression is 0.86. S5 presented a model to predict faults in software projects. Furthermore, the results of this study show that R square and adjusted R square are 83% and 80%, respectively. S21 presented a review study to detect defects in a software project. It also seeks to find an optimal model to detect defects efficiently to save costs and time. Also, this study confirmed that regression models have achieved high results in terms of accuracy in detecting defects of software projects.

Secondly, the studies (S6, S7, S8, S9, and S10) relied on logistic regression, where S6 presented an approach to improve the quality of software projects by detecting bugs in software projects efficiently. Also, the standard error in the proposed statistical technique is 0.24. S7 presented a study to detect defects in software projects in the early stage to save effort, money, and time. This study also depends on statistical techniques such as correlation coefficients and logistic regression. The results show that the accuracy in logistic regression is 91.2%, and the correlation coefficient is 0.95. S8 presented a model for predicting defects in software projects. The result of this study shows that the standard error in logistic regression is 0.19. S9 presented an empirical model to predict fault in software projects. This study also depends on the binary logistic regression technique to predict defects in software projects. The results also show that the precision, recall, and f1 measures are 0.65, 0.9, and 0.79. S10 presented a study to predict software defects by using logistic regression just in time. The results of this study show that the proposed technique is better than the state-of-the-art methods in terms of accuracy. The accuracy of the proposed technique is 0.73.

TABLE VI. INTELLIGENT AND STATISTICAL TECHNIQUES IN SOFTWARE DEFECT PROJECT

NO	Multiple Linear Regression	Logistic Regression	Statistical Analysis	Data Mining	Machine Learning	Other
S1	✓	-	-	-	-	-
S2	✓	-	-	-	-	-
S3	-	-	✓	-	-	-
S4	✓	-	-	-	-	-
S5	✓	-	-	-	-	-
S6	-	✓	-	-	-	-
S7	-	✓	-	-	-	✓
S8	-	✓	-	-	-	-
S9	-	✓	-	-	-	-
S10	-	✓	-	-	-	-
S11	-	-	-	✓	-	-
S12	-	-	✓	-	-	-
S13	-	-	-	-	✓	✓
S14	-	-	-	✓	-	-
S15	-	-	-	-	✓	✓
S16	-	-	-	-	✓	-
S17	-	-	✓	-	-	-
S18	✓	✓	-	-	✓	-
S19	-	-	-	-	-	✓
S20	-	-	-	-	✓	-
S21	✓	✓	-	-	-	-
S22	-	-	-	-	-	✓
S23	-	-	-	-	-	✓
S24	-	-	✓	-	-	-
S25	-	-	✓	-	-	-
S26	-	-	-	-	-	✓

Thirdly, the studies (S3, S11, S12, S14, S17, S24, S25) relied on statistical analysis and data mining techniques where S3 presented an approach to forecasting defects in software projects. It also depends on statistical regression such as multiple linear regression to predict defects in those projects. Besides, the adjusted R square in statistical regression is 98.6%, and the correlation coefficient is 0.98. S11 presented a model to extract software static defects by using data mining techniques. The results of this study show that the accuracy in Association Rules, Decision Tree, Naive Bayes, and Neural Network is 77.2%, 76.6%, 73.2%, and 73.2%, respectively. Thus, Association Rules is better than Decision Tree, Naive Bayes, and Neural Network in terms of accuracy. S12 presented a study to improve the quality of software projects using statistical analysis. The results of this study were evaluated in terms of projection of errors (total errors) and cumulative projection of severity errors (e.g., series, moderate and minor). It also shows that total errors in 2016 are more than in 2015 by 1.5%.

Moreover, most severity errors are minor types. S14 presented a study to predict defects in legacy industrial software using data mining techniques. The results of this study show that the area under the curve (AUC) in Random Forest, Logistic Regression, Decision Tree, Naive Bayes, and a combination of Random Forest + Logistic regression is 0.73, 0.72, 0.66, 0.67, and 0.75. Thus, a combination of Random Forest + Logistic regression is better than Random Forest, Logistic Regression, Decision Tree, Naive Bayes. S17 presented an approach to improve software quality and cost minimization using statistical analysis. The results of this study were evaluated in terms of standard error. The standard error in the statistical model is 0.13. S24 presented a study to evaluate the impact of automated feature selection techniques on the interpretation of defect models. This study investigated 12 automated feature selection techniques in terms of consistency, correlation, performance, computational cost. By analyzing 14 publicly-available defect datasets, the results showed that the most important inconsistent metrics are highly correlated with the automated Spearman correlation of 0.85–1. S25 presented a study to predict defects in software models. This study applied the Mathews Correlation Coefficient-MCC to avoid defects in software models. MCC in F-score is less than 0.01. Therefore, the proposed technique is better than the state-of-the-art methods in terms of MCC.

Fourthly, the studies (S13, S15, S16, S18, S20) relied on machine learning techniques where S13 presented a model to predict software defects by using an enhanced genetic algorithm. The results of this study were evaluated in terms of precision. It also confirmed that precision in enhanced genetic algorithm, genetic algorithm, and neural network is 0.93, 0.81, and 0.80, respectively. Thus, the enhanced genetic algorithm is better than the genetic algorithm and neural network. S15 presented a model to predict effective faults in software projects using extreme learning machines with various kernel methods (e.g., Linear kernel, Polynomial kernel, and Sigmoid kernel). The results of this study were evaluated in terms of accuracy metrics. The accuracy in the linear kernel, Polynomial kernel, and Sigmoid kernel is 0.88, 0.93, and 0.91. Thus, an extreme learning machine using the Polynomial kernel is better than linear kernel and Sigmoid kernel. S16 presented a model to predict universal defects in software projects using clustering techniques. The results of this study were evaluated in terms of AUC. The AUC in K-mean clustering is 0.76. S18 presented a model to detect defects in a software project. This study depends on object-oriented metrics. It also relies on many intelligent techniques such as linear regression (LR), logistic regression (LG), random forest (RF), support vector machine (SVM), and neural network (NN). The results of this study were evaluated in terms of AUC. The AUC in LR, LG, RF, SVM and NN is 0.86, 0.94, 0.91, 0.90 and 0.90. Thus, LG is better than LR, RF, SVM, and NN. S20 presented a review analysis to predict defects in a software project. This study depends on many metrics, such as product and process metrics. It also introduced a comparative analysis between Neural Network, Naive Bayes, Deep Forest technique. This study relies on previous works in the analysis of these techniques. Besides, this study confirmed that Deep Forest is better than Neural Network, Naive Bayes in terms of error rate.

Fifthly, the studies (S19, S22, S23, and S26) relied on other intelligent and statistical techniques where S19 presented an approach to predict many faults in a software system by using a genetic algorithm. The results of this study were evaluated in terms of error rate and recall. The error rate and recall in the genetic algorithm are 0.11, 0.91, respectively. S22 presented a new technique in software defect prediction by Complexity-based Oversampling. This paper relied on three main factors: a line of code, number of children, and weighted method per class. By analyzing the results, the proposed technique is better than the other oversampling techniques under the statistical Wilcoxon rank-sum test and Cliff's effect size. S23 presented a framework to predict software defect type using concept-based classification. This paper's main objective is to minimize the labeled training data's dependence for automation of the software defect type classification task. The results show that the proposed framework outperforms the state-of-the-art semi-supervised [LeDEx] in terms of the F1 score. F1 score in the proposed framework and LeDEx is 63.16% and 62.30%, respectively. S26 presented a study to assess the software prediction model by using Receiver Operating Characteristic. The results showed that the proposed technique is better than all other state-of-the-art methods in terms of recall and accuracy by 0.4 and 0.8, respectively.

The literature study also analyzed the performance evaluation metrics in the scope of our RQ3. Results are shown in Table VII and Table VIII. 21% of the selected manuscripts (S10,11,15, 9, 13, and 21) adopted accuracy and precision. 21% of them (S9, 11, 19, 23, and 25) selected only recall and F1 score. The error rate was used by 30% of the analyzed manuscripts (S1, 6, 7, 8, 17, 19, 20, 22, 24, and 26). 15% of the manuscripts adopted the R Square measure (S1, 2, 3, 4, and 5). We also realized that 13% (S12 S14, S16, and S18) did not use any defined evaluation metric.

Our research helped us to determine several research gaps. It was only possible to identified a few manuscripts (S11 and S13) tackling specific metrics impacting defects in software projects. For example, some studies (S5, S6, S9, S18, and S21) are concentrated on the OOP metric in general, with no mention of the line count of code and the number of developers. There are only simple manuscripts (S14, S20, S23, and S24) regarding finding defects in all types of software projects (small, medium, and large projects). However, stakeholders in software companies seem to find this topic pertinent and are willing not only to enhance software efficiency in those projects but interested to predict early defects in software projects to save costs and money. The results of this survey also showed a significant gap in the field of "intelligent and statistical models," particularly relating to the automatic prediction of defects in software projects. Some of the most promising algorithms are not yet being utilized. Only a few studies (S18 and S21) tackle the application of "hybrid statistical and intelligent techniques, for instance, logistic regression with multiple linear regression and regression analysis with deep learning," which is a promising technique for forecasting defects in software projects. Moreover, there is a lack of official studies to identify critical factors that influence defects in software projects.

TABLE VII. SAMPLE OF PERFORMANCE METRICS RATE IN PREVIOUS WORK

	Performance Metrics	**Rate**
1	Accuracy and precision	21%
2	Recall and F1 Score	21%
3	Error Rate	30%
4	R Square Measure	15%
5	Other	13%

TABLE VIII. MAJORITY OF PERFORMANCE METRICS USED IN SOFTWARE DEFECT PROJECTS

NO	**Accuracy**	**Precision**	**Recall**	**F1 score**	**Error Rate**	**R-Square**	**Other**
S1	-	-	-	-	✓	✓	-
S2	-	-	-	-	-	✓	-
S3	-	-	-	-	-	✓	-
S4	-	-	-	-	-	✓	-
S5	-	-	-	-	-	✓	-
S6	-	-	-	-	✓	-	-
S7	-	-	-	-	✓	-	-
S8	-	-	-	-	✓	-	-
S9	-	✓	✓	✓	-	-	-
S10	✓	-	-	-	-	-	-
S11	✓	✓	✓	✓	-	-	-
S12	-	-	-	-	-	-	✓
S13	-	✓	-	-	-	-	-
S14	-	-	-	-	-	-	✓
S15	✓	-	-	-	-	-	
S16	-	-	-	-	-	-	✓
S17	-	-	-	-	✓	-	
S18	-	-	-	-	-	-	✓
S19	-	-	✓	-	✓	-	-
S20	-	-	-	-	✓	-	-
S21	✓	-	-	-	-	-	-
S22	-	-	-	-	✓	-	-
S23	-	-	-	✓	-	-	-
S24	-	-	-	-	✓	-	-
S25	-	-	-	✓	-	-	-
S26	-	-	-	-	✓	-	-

IV. Proposed Model

Proposal of a new proposed model based on a statistical model able to predict defects in software projects. This section presents an approach for a statistical model able to predict defects in software projects. The proposed model has been used in several scientific data science researches like is the case of [7]. As shown in Fig. 4, the detailed the proposed model will cover the following phases:

- State-of-the-art analysis: Review the literature to extract important metrics, data sources, mathematical and computational approaches used for predicting defects of software projects.
- Data collection: data is collected from the NASA data sets online. We have two reasons to select the NASA Data set. The first reason is it is too hard to collect huge data from software companies to reveal the defects in software projects. The second reason for selecting Nasa is based on its vast and high-quality data. It explains the static measures and other variables that are used to detect static defects in software projects. It also shows a binary variable indicating whether the module is defective or not.
- Data Analysis and Pre-Processing: Analyze the data in detail and, if necessary, transform it to expose its information content better. Different mathematical techniques may be used, namely, outlier removal, discretization, reduction of the number of variables, and/or dimensionality (adopting regression models).
- Feature selection: determine critical metrics and detect defects that will be adopted in the proposed IST study by using logistic regression and multiple linear regression. Create a mapping between logistic regression and multiple linear regression to determine the final list of critical metrics capable of predicting defects in software projects.
- Build a model: present a statistical model capable of predicting defects in software projects using multiple linear regression and logistic regression.
- Training and verification model: train the model with data set and verify its ability to predict defects in software projects.
- Also, we will present a comparison between logistic regression and multiple linear regression by using the final list of critical metrics to determine which one is better than the other in terms of accuracy, precision, recall, F1 measure, and error rate.

Following this holistic approach, we built a methodology composed of five phases, as shown in Fig. 4.

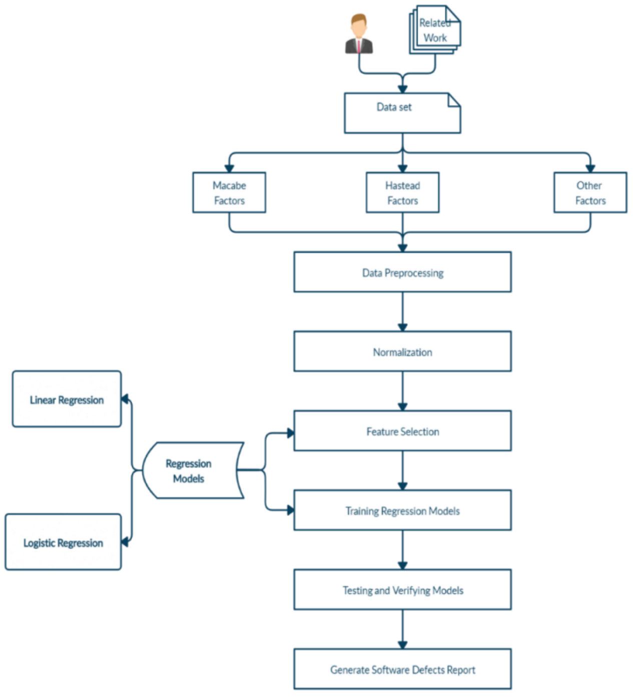

Fig. 4. A Proposed Statistical Model for Software Defects Prediction.

V. Conclusion

This paper presented a systematic review on the topic of revealing defects in software projects, concentrating on finding replies to our research questions, a diplomatic map was used to find the most used terminology in the statistical technique's software projects domains. By following a Prisma approach in our systematic review, we started by determining 627 papers and ended with VP analyses of 26 papers. The research questions covered three major points. Firstly, we identified the factors of our metrics that influence revealing defects in software projects. Secondly, we concentrated our research on identifying the production techniques used in the context. After, we determined the evaluation criteria used by those techniques. Thus, there is still a chance for enhancement regarding our topic to use statistical and intelligent techniques to reveal defects in software projects.

Finally, a new methodology based on a statistical model able to predict defects in software projects was proposed.

This study succeeded in identifying the critical factors that affect the detection of defects in the programs. Statistical analysis is executed by four methods, which are MLR-CDF, MLR-PLSDF, LR-CDF, and LR-PLSDF. LR-CDF outperforms on all the proposed methods in order to accuracy and standard error. In addition, LR-CDF outperforms on state-of-the-art methods (Association rule, Decision tree, Naive Bayes, and neural network) related to the accuracy by 9.1%, 10.3%, 13.1%, and 13.1%, respectively.

The study has some limitations. it was restricted by the search keywords selected and the time of the manuscripts (last six years). In addition, it utilized a fixed number of electronic sources. Furthermore, this study only handled English scientific papers, and we cannot warranty to have picked all the worthy substance for our review.

It is recommended as future work to utilize other techniques in terms of improving the model accuracy and identifying critical factors for revealing defects in software projects.

REFERENCES

[1] Y. Koroglu et al., "Defect prediction on a legacy industrial software: A case study on software with few defects," Proc. - Int. Conf. Softw. Eng., vol. 17-May-201, pp. 14–20, 2016, doi: 10.1145/2896839.2896843.

[2] A. Abdelaziz Mohamed, N. Ramadan Darwish, and H. Ahmed Hefny, "Towards a Machine Learning Model for Predicting Failure of Agile Software Projects," Int. J. Comput. Appl., vol. 168, no. 6, pp. 975–8887, 2017.

[3] M. Sirshar, "Comparative Analysis of Software Defect Prediction Techniques," no. December, p. 456頁、453頁、603頁, 2019.

[4] D. Sharma and P. Chandra, "Identification of latent variables using, factor analysis and multiple linear regression for software fault prediction," Int. J. Syst. Assur. Eng. Manag., vol. 10, no. 6, pp. 1453–1473, 2019, doi: 10.1007/s13198-019-00896-5.

[5] V. S. Sukanya and S. Saraswathy, "An Enhanced Evolutionary Model for Software Defect Prediction," vol. 7, no. 10, pp. 15323–15328, 2017.

[6] S. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming," Procedia Comput. Sci., vol. 62, no. Scse, pp. 303–311, 2015, doi: 10.1016/j.procs.2015.08.454.

[7] A. H. Yousef, "Extracting software static defect models using data mining," Ain Shams Eng. J., vol. 6, no. 1, pp. 133–144, 2015, doi: 10.1016/j.asej.2014.09.007.

[8] 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," Prof. la Inf., vol. 29, no. 1, pp. 1–20, 2020, doi: 10.3145/epi.2020.ene.03.

[9] D. Moher et al., "Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement," PLoS Medicine, vol. 6, no. 7. 2009, doi: 10.1371/journal.pmed.1000097.

[10] S. N. U., "Software Testing Defect Prediction Model - a Practical Approach," Int. J. Res. Eng. Technol., vol. 02, no. 05, pp. 741–745, 2013, doi: 10.15623/ijret.2013.0205001.

[11] M. Dhiauddin and S. Ibrahim, "A Prediction Model for System Testing Defects using Regression Analysis," Int. J. Soft Comput. Softw. Eng., vol. 2, no. 7, pp. 55–68, 2012, doi: 10.7321/jscse.v2.n7.6.

[12] E. A. Felix and S. P. Lee, "Integrated Approach to Software Defect Prediction," IEEE Access, vol. 5, pp. 21524–21547, 2017, doi: 10.1109/ACCESS.2017.2759180.

[13] D. Verma and S. Kumar, "Prediction of defect density for open source software using repository metrics," J. Web Eng., vol. 16, no. 3–4, pp. 294–311, 2017.

[14] D. Sharma and P. Chandra, "Identification of latent variables using, factor analysis and multiple linear regression for software fault prediction," Int. J. Syst. Assur. Eng. Manag., vol. 10, no. 6, pp. 1453–1473, 2019, doi: 10.1007/s13198-019-00896-5.

[15] O. Sari and O. Kalipsiz, "Bug prediction for an ATM monitoring software use of logistic regression analysis for bug prediction," ICEIS 2015 - 17th Int. Conf. Enterp. Inf. Syst. Proc., vol. 2, pp. 382–387, 2015, doi: 10.5220/0005382803820387.

[16] G. Mauša, T. G. Grbac, L. Brezočnik, V. Podgorelec, and M. Heričko, "Software metrics as identifiers of defect occurrence severity," CEUR Workshop Proc., vol. 2508, no. September, pp. 22–25, 2019.

[17] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," Inf. Softw. Technol., vol. 59, no. February, pp. 170–190, 2015, doi: 10.1016/j.infsof.2014.11.006.

[18] M. K. Dhillon, P. B. Singh, and P. J. Singh, "Empirical Model for Fault Prediction On the Basis of Regression Analysis," Int. J. Sci. Res., vol. 5, no. 6, pp. 163–168, 2016, doi: 10.21275/v5i6.nov164139.

[19] X. Chen, Y. Zhao, Q. Wang, and Z. Yuan, "MULTI: Multi-objective effort-aware just-in-time software defect prediction," Inf. Softw. Technol., vol. 93, pp. 1–13, 2018, doi: 10.1016/j.infsof.2017.08.004.

[20] N. Of, "S Tatistical a Nalysis of R Ainfall I Nsurance," vol. 89, no. 5, pp. 1248–1254, 2007, doi: 10.1111/j.1467-8276.2007.01092.x.

[21] V. S. Sukanya and S. Saraswathy, "An Enhanced Evolutionary Model for Software Defect Prediction," vol. 7, no. 10, pp. 15323–15328, 2017.

[22] Y. Koroglu et al., "Defect prediction on a legacy industrial software: A case study on software with few defects," Proc. - Int. Conf. Softw. Eng., vol. 17-May-201, pp. 14–20, 2016, doi: 10.1145/2896839.2896843.

[23] L. Kumar, A. Tirkey, and S. K. Rath, "An effective fault prediction model developed using an extreme learning machine with various kernel methods," Front. Inf. Technol. Electron. Eng., vol. 19, no. 7, pp. 864–888, 2018, doi: 10.1631/FITEE.1601501.

[24] F. Zhang, A. Mockus, I. Keivanloo, and Y. Zou, "Towards building a universal defect prediction model with rank transformed predictors," Empir. Softw. Eng., vol. 21, no. 5, pp. 2107–2145, 2016, doi: 10.1007/s10664-015-9396-2.

[25] A. K. Marandi and D. A. Khan, "An approach of statistical methods for improve software quality and cost minimization," Int. J. Appl. Eng. Res., vol. 12, no. 6, pp. 1054–1061, 2017.

[26] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, "The impact of using regression models to build defect classifiers," IEEE Int. Work. Conf. Min. Softw. Repos., pp. 135–145, 2017, doi: 10.1109/MSR.2017.4.

[27] S. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming," Procedia Comput. Sci., vol. 62, no. Scse, pp. 303–311, 2015, doi: 10.1016/j.procs.2015.08.454.

[28] M. Sirshar, "Comparative Analysis of Software Defect Prediction Techniques," no. December, p. 456頁、453頁、603頁, 2019.

[29] M. S. Rawat and S. K. Dubey, "Software Defect Prediction Models for Quality Improvement: A Literature Study," Int. J. Comput. Sci. Issues, vol. 9, no. 5, pp. 288–296, 2012.

[30] S. Feng et al., "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction," Inf. Softw. Technol., vol. 129, no. September 2020, p. 106432, 2020, doi: 10.1016/j.infsof.2020.106432.

[31] S. Patil and B. Ravindran, "Predicting software defect type using concept-based classification," Empir. Softw. Eng., vol. 25, no. 2, pp. 1341–1378, 2020, doi: 10.1007/s10664-019-09779-6.

[32] J. Jiarpakdee, C. Tantithamthavorn, and C. Treude, "The impact of automated feature selection techniques on the interpretation of defect models," Empir. Softw. Eng., vol. 25, no. 5, pp. 3590–3638, 2020, doi: 10.1007/s10664-020-09848-1.

[33] A. A. Bangash, H. Sahar, A. Hindle, and K. Ali, "On the time-based conclusion stability of cross-project defect prediction models," Empir. Softw. Eng., 2020, doi: 10.1007/s10664-020-09878-9.

[34] S. Morasca and L. Lavazza, "On the assessment of software defect prediction models via ROC curves," Empir. Softw. Eng., vol. 25, no. 5, pp. 3977–4019, 2020, doi: 10.1007/s10664-020-09861-4.