IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 15 September 2023, accepted 26 October 2023, date of publication 2 November 2023, date of current version 8 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329732

SURVEY

An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management

RAZVAN BOCU[1,2], ALEXANDRA BAICOIANU[1,2], AND ARPAD KERESTELY[2]
[1]Department of Mathematics and Computer Science, Transilvania University of Braşov, 500091 Braşov, Romania
[2]Department of Research and Technology, Siemens Industry Software, 500227 Braşov, Romania

Corresponding author: Razvan Bocu (razvan.bocu@unitbv.ro)

ABSTRACT Bug reports are generated in large numbers during the software development processes in the software industry. The manual processing of these issues is usually time consuming and prone to errors, consequently delaying the entire software development process. Thus, a properly designed bug triage and management process implies that essential operations, such as duplicate detection, bug assignments to proper developers, and determination of the importance level, are sustained by efficient algorithmic models and implementation approaches. Designing and implementing a proper bug triage and management process becomes an essential scientific research topic, as it may significantly optimize the software development and business process in the information technology industry. Consequently, this paper thoroughly surveys the most significant related scientific contributions analytically and constructively, distinguishing it from similar survey papers. The paper proposes optimal algorithmic and software solutions for particular real-world use cases that are analyzed. It concludes by presenting the most important open research questions and challenges. Additionally, the paper provides a valuable scientific literature survey for any researcher or practitioner in software bug triage and management systems based on artificial intelligence and machine learning techniques.

INDEX TERMS Bug report, bug prioritization, bug assignment, bug triaging, classification, machine learning.

I. INTRODUCTION

Large software development projects rely on bug triaging as an important part of software testing. Thus, it supports the software bug management processes, while allowing relevant decisions, which are related to the software bug fixing, to be made. Relevant tasks are represented by properly assigning bugs to adequate developers, prioritizing bugs, and detecting duplicate bugs. Nevertheless, manual bug triaging appears as an essentially time consuming and tedious task, considering that a significant part of software development requires a lot of time and other types of resources. Considering old statistics from August 2009, the Mozilla bug database contained over 500,000 bug reports, and the Eclipse bug database had over 250,000 bug reports. The average number of bug reports created daily amounts to 170 for the Mozilla database and 120 for the Eclipse database between January and July 2009. The dynamics of software systems development have constantly increased during the past fifteen years. Therefore, the problem that is approached in this paper becomes increasingly more relevant.

The process of bug triage involves that a triager makes a decision regarding the bugs entered in the respective bugs repository through an analysis, which involves two variants. Thus, the repository-oriented decisions involve that the reported bug does not represent a duplicate, the person that triages checks it for validity, which means that the bug is assessed whether it is genuine. This mediates the removal of bug reports that do not require a resolution. The remaining bug reports are investigated to support the development-oriented decisions, which involve that the triager assesses the severity and priority levels of the bugs. These levels are modified if inappropriate values are observed, so sufficient resources are allocated to resolve critical bugs. Consequently,

The associate editor coordinating the review of this manuscript and approving it for publication was Xinyu Du.

the person that addresses the manual bug triaging process writes down the required remarks for the bug and assigns this bug report to the suitable developer. Therefore, the need to design and implement an efficient bug triaging and management system becomes clear.

In this context, the task of bugs classification, which implies the determination of priorities, appears as a very important effort package relative to very large software development projects, and also open source projects, considering that the efficiency of the development process is usually quantitatively assessed considering the number of open bug reports, and the average resolution time. The manual determination of bug priorities also introduces inherent human errors to the process, which essentially depends on the bug triager's subjective perspective and experience. Consequently, a significant number of bug reports may have been assigned incorrect priority levels while other bug reports remain unaddressed. The implied economic and operational consequences are easily discernible. Therefore, an automatic bug triaging and management strategy, which uses a certain automated approach, is required. Thus, the related scientific literature includes various machine learning approaches, such as Decision Tree (DT), Support vector machine (SVM) classification algorithms, Naive Bayes (NB) classifiers, Information Retrieval (IR), and Random Forest (RF) models. This paper concentrates on a logically structured survey, which aims to analyze and suggest the optimal bug triaging and management approaches. The scientific objectives of this research are the following.

- To survey the existing literature to determine shortcomings and propose optimal solutions.
- Identification of the most relevant studies in bug classification.
- Prioritizing studies relevant to bug sorting according to various criteria: citations, scientific relevance, objectives achieved, reproducibility of experiments, genericity of solutions, etc.
- Classification by various criteria of frameworks dedicated to the open problem.
- To determine the relevant scientific research trends.
- To identify the relevant research problems.
- To define the scientific relevance of the corresponding content of research.
- To propose the conceptual and practical relevance of automatic bug triaging in the management processes.

The rest of the paper is structured according to the following sections. The next section presents the structured research methodology, which has been considered. Following, the most relevant artificial intelligence models are described and analyzed. Moreover, the relevant performance evaluation methods and related metrics are surveyed and assessed. Furthermore, the most relevant scientific challenges and open research questions are discussed. Consequently, the essential research questions, which determined this extended survey process, are analyzed, and the degree of this paper's accomplishment is objectively assessed. The last section concludes the paper.

TABLE 1. Reference scientific literature databases and academic search engines.

Database	Public URL
Science Direct-Elsevier - DL	http://www.sciencedirect.com/
Scopus - SE	http://www.scopus.com/
IEEE Xplore - DL	http://ieeexplore.ieee.org/
ACM Digital library - DL	http://dl.acm.org/dl.cfm
Web of Science - SE	https://www.webofknowledge.com/
Wiley online library - DL	https://onlinelibrary.wiley.com/
Google Scholar - SE	https://scholar.google.ro/
Sensors - DL	https://www.mdpi.com/
Springer - DL	https://www.springer.com/
ResearchGate - Scientific social networking	https://www.researchgate.net/
Edinburgh library database - DL	https://my.napier.ac.uk/Library/
RefSeek - SE	https://www.refseek.com/
Bielefeld Academic Search Engine - SE	https://www.base-search.net/

II. RESEARCH METHODOLOGY

The survey methodology relates to a systematic review (SR) approach, which is determined by the methodology that is referred to as ''Preferred Reporting Items for Systematic Reviews and Meta-Analysis'' (PRISMA) [1]. More precisely, the scientific methodology is based on the following phases: specification of research questions, identification and survey of proper papers, and specification of the relevant inclusion and exclusion criteria.

A. RESEARCH QUESTIONS

The literature review relates to the following research questions.

- What is the related significant literature, which approaches conceptual problems, and reports adequate solutions?
- What are the relevant scientific research trends?
- What are the determined research questions and shortcomings?
- What is the reviewed research scope's conceptual, scientific, and real-world importance?
- What are the principal algorithmic and machine learning models that specify and implement automatic bug triaging and management approaches?

The following subsection describes the logical structure of the proper research process.

B. RESEARCH PROCESS

The reference sources that were considered in order to collect the proper scientific literature are described in Table 1. Here, *DL* means Digital Library, and *SE* means Search Engine.

The next subsection presents the exclusion and inclusion criteria, which have been used to filter the scientific contributions objectively.

TABLE 2. Inclusion criteria.

Inclusion criteria
Articles should be indexed by at least one of the selected scientific literature databases.
Scientific papers are published from 2010-2023, and significant historical papers are also selected.
Articles should meet at least one of the search terms, relative to this review article's title, abstract, and keywords.
Papers should be published in indexed journals, conference proceedings, mainstream technical journals, or books and chapters issued by top tier publishing houses.
Reviewed articles should clearly analyze and answer specified research questions.
A search that relates to the title, abstract, and the full text is sufficient.

TABLE 3. Exclusion criteria.

Exclusion criteria
Articles that are not written in English.
Duplicated articles, which are identified using more than one of the specified scientific literature databases.
Articles with full texts that cannot be retrieved.
Articles that are only marginally relevant for studying automatic bug triaging and management, and related artificial intelligence models.

C. EXCLUSION AND INCLUSION CRITERIA

The appropriateness of the surveyed papers, and, consequently, the scientific adequacy of this review paper, are also determined by certain inclusion criteria (IC), and exclusion criteria (EC). More precisely, contributions that do not meet the specified EC are disregarded. The IC-related filtering model relates to a logical process based on the following steps.

- Step 1. Abstract-related filtering: irrelevant articles are disregarded considering the information acquired from the abstract, and also based on the keywords. More precisely, articles that fulfill at least 50% of the relevance threshold are considered.
- Step 2. Full text-related filtering: articles that concern only a small part of the scientific scope, as specified by the abstract and the keywords, are disregarded.
- Step 3. Quality analysis-related filtering: the rest of the papers were additionally filtered considering that at least one of the following conditions are unmet:
  <The paper describes a functional solution concerning the automatic bug triaging and management models.> AND <The article fully presents the implemented technical solution.> AND <The article surveys related relevant contributions.> AND <The article presents and assesses the outcomes of the experimental process.>

Furthermore, the inclusion criteria are presented in table 2. Moreover, the exclusion criteria are described in table 3.

The following sections extensively review the large number of articles, which were selected considering the principles of this scientific survey methodology.

III. ARTIFICIAL INTELLIGENCE MODELS FOR BUGS TRIAGING

The relevant scientific literature presents various approaches concerning software bug triaging (SBT). Thus, SBT models are grouped into six fundamental categories relative to the considered AI technologies. The six technological approaches relate to machine learning (ML), information retrieval (IR), social network analysis (SNA), recommender systems (RS), mathematical modeling and optimization (MMO), and deep learning (DL). The features of these approaches are described in Table 4. This section surveys the most relevant scientific contributions identified in each category.

A. BUG TRIAGING MODELS BASED ON MACHINE LEARNING

Machine Learning (ML) models represent the natural solution to implement an automatic software bug triaging (SBT) system. Proper machine learning-based models [2], [3] are frequently used in this respect (Softmax classifier, Support Vector Machine, Multinomial Naive Bayes, K-Nearest Neighbors, J48, Random Forests, Artificial Neural Networks), along with clustering [4] and association rule mining [5]. SBT is regarded as a multiclass, single-label classification problem [6], which considers the software developer as a class. Thus, it is immediately discernible that the proper classification techniques are frequently used relative to machine learning-based bug triaging techniques. The performance metrics, such as accuracy, precision, recall, and F1-measure, are used in order to assess the described approaches, typically on the top 5 or 10 best outcomes. Thus, the accuracy gets as high as 40% - 50%. It is relevant to note that a comparative review concerning a handful of machine learning models for software bug triaging is presented in article [7], and is also conducted by Goyal and Sardana [8]. While other techniques like information retrieval can beat plain ML-based SBTs, they are relatively simple to model, and there are proper libraries, which implement efficient Application Programming Interface (API) support. Nevertheless, the enhancement of the computational performance [9] represents a goal that continues to motivate scientific research efforts to enhance the existing approaches [10], and consequently assess modern approaches, which are based on artificial intelligence models. The surveyed experimental analysis contributions suggest that plain machine learning-based approaches under-perform deep learning-based models, but they are comparable with information retrieval-based models.

B. BUG TRIAGING MODELS BASED ON INFORMATION RETRIEVAL

The general algorithmic process of software bug triaging may also be perceived as a problem of information retrieval (IR), which presumes that the relevant data determines that a developer is fetched from the set of software developers relative to the newly created bug reports. Thus, IR-related models, such as LSA (Latent Semantic Analysis), and LSI (Latent Semantic Indexing) [11], [12], are also used together with other relevant models. The accuracy scores generated using IR-related SBT models range from 63.2%

TABLE 4. Features of various bug triaging techniques.

Bug triaging class	Features
ML	Training data and classification
IR	Text similarity and topic modeling
SNA	Developers network and cross repository analysis
RS	An ordered list of developers
MMO	Optimization and computational mathematics
DL	Layered stack of training and classification

to 96%. Moreover, a recall value of 95% is reported by the work described in article [13]. Additionally, TF-IDF (Term Frequency-Inverse Document Frequency) represents the most usual algorithmic model relative to IR-related models for software bug triaging. Several articles describe relevant changes to TF-IDF-related solutions related to the scientific efforts presented in [14] and [15]. These resources consider bug location information, term weighting in TF-IDF, and time metadata concerning the TF-IDF presentation. Additionally, software bug triaging uses both text mining, which is presented in articles [16], [17], [18], [19], and also text similarity models, which are described in articles [20], [20], [21], [22], [23], [24], [25]. The topic is also approached in other related studies.

Moreover, large software repositories determine an interesting scope of scientific research, which is interestingly approached in [26]. Also, the concept of topic modeling is approached in several articles, such as [12], [27], [28], [29], and [30], with a clear emphasis on software bug triaging. Moreover, Latent Dirichlet Allocation (LDA) determines an important probability-related algorithmic approach, which is described in paper [31]. Other similar approaches consider this algorithmic model for automatic software bug triaging, presented in [32] and [33]. Certain papers regard specialized variants concerning topic modeling in connection to automatic software bug triaging. Thus, the multi-feature topic model (MTM) is proposed in article [34]. At the same time, the Entropy Optimized Latent Dirichlet Allocation is described in article [35], along with the Multiple LDA concept proposed in paper [29]. Notably, the maximum generated accuracy through the utilization of topic modeling-based bugs triaging approaches is 98.31%, which is suggested by the work presented in article [36].

The extensive scientific literature that was surveyed suggests that, in a similar fashion to machine learning-based models, information retrieval-based models provide acceptable application programming interface (API) features. Furthermore, the implied algorithmic models are easy to model and implement in a suitable programming language. Despite the obvious conceptual and practical advantages, the relevant approaches related to information retrieval and topic modeling present computational performance issues in certain real-world scenarios, and they may also have difficulties generating proper terms relative to the topics produced by the respective topic modeling approaches. Thus, the topics that are produced by related topic modeling approaches may provide a certain degree of randomness, which may impact the overall data analysis process, as it is suggested by the work that is reported in article [37]. Nevertheless, it is relevant to state that the surveyed scientific literature suggests that the main problem, which should be addressed, is represented by the insufficient level of computational performance that manifests in certain real-world scenarios that are described in article [32]. This may affect the real-time processing of relevant software bug data.

There is a clear similarity between ML-related and IR-related automatic bug triage and management techniques. The fundamental difference is determined by the fact that IR-related approaches essentially relate to the textual data that are stored by the software bug repositories. Similarly to ML-related approaches, algorithmic and software modeling is also relatively easy with IR-related models, which require the least computational and data storage resources to implement automatic software bug triaging systems. The surveyed literature also suggests that these IR-related approaches benefit from consistent support relative to all modern programming languages and application programming interfaces. Some drawbacks, such as computational scalability and real-time software bug triaging, are shared between IR-related and ML-related approaches.

C. BUG TRIAGING MODELS BASED ON SOCIAL NETWORK ANALYSIS

Relative to software bug triaging, social networks designate the developers' network, which is used by the enrolled software developers in order to sustain the implied software systems development processes. The activity of bug resolution implies the existence of particular skills. Nevertheless, developers use third-party or external support sources, such as StackOverflow, or GitHub, which may provide useful technical information. The extraction of useful information from several sources in order to enhance the resolution of software bugs is referred to as Crowdsourcing. Thus, this idea is explored in certain papers, such as [38], [39], and [40].

The relevant technical information, as it is fetched from several repositories, may be integrated together with the identifying data of the software bug repository, which may help identify the software developers with expert skills. This approach is generally designated as cross-repository analysis and is approached in an interesting manner in article [41]. The synergistic combination of crowdsourcing and social networks-related analysis models creates a functional advantage regarding determining the relationship between developers and their technical skills. This generally supports an enhanced bug assignment process to the proper developers, which may support the implied automatic bug management systems. The surveyed scientific literature suggests that this type of approach determines a problematic aggregation and integration of the acquired and existing data, which is particularly derived from the consideration of multiple data sources. The implied problem is studied and reported in

articles [38], [39], and [40], which also analyze the generated and existing relationship graphs. Furthermore, the surveyed literature suggests that automatic software bug triage and management approaches, which are based on software network analysis, are relatively difficult to design due to the implied processing of the graph data structures that model the developer-bug relationships. Consequently, the implied data processing routines require more computational time and resources. Nevertheless, the surveyed contributions claim that this is compensated by the overall enhancement of the bug assignment or reassignment, while further algorithmic and implementational improvements are likely to assure the required computational scalability.

D. BUG TRIAGING MODELS BASED ON RECOMMENDER SYSTEMS

Recommender systems represent a fundamental concept in the scope of machine learning scientific research. Thus, the contributions that are described in articles [23], [43], [44], and [45] suggest that proper developers may be efficiently recommended and assigned to the newly created bug reports. The consideration of recommender systems (RS) mediates the creation of a list of software developers, and a list of the most suitable k developers is generated, according to their assessed technical skills. Thus, certain historical contributions are reported in articles [46], [47], and [48]. The approach that is described in article [48] involves that a ranking of the software developers is created considering using the mechanism of developer prioritization. It is interesting to note that certain performance metrics are described in the surveyed literature. Thus, in article [48], the authors propose Accuracy@K (Acc@K), Precision@K (P@K), and Recall@K (R@K). Here, K designates the most suitable K software developers relative to the list, which was created using the implied recommender system. There is a majority of the articles that were surveyed, which consider the Eclipse and Mozilla Firefox bug trackers, and Recall@10 is obtained at a level of up to 90%.

Including an RS-related algorithmic module mediates the generation of a list of ranked developers, which are sent over to the automatic bug triaging and management components. Consequently, the system generates a higher-quality software developer suggestion, who is available and adequate to fix the respective bugs. Nevertheless, it is important to note that there are certain conceptual and practical problems, which are reported in the surveyed literature. Thus, the issue of cold start is analyzed in article [49], while the performance and data sparsity are approached in paper [50]. Thus, the issue of cold start and data sparsity are determined by insufficient data relative to a certain data item or category, which provokes an overall degradation of the recommender system's computational performance. More precisely, an RS-related bug triaging system may not properly identify the software developer, if sufficient data are not available regarding the potentially suitable developers.

E. BUG TRIAGING MODELS BASED ON MATHEMATICAL MODELING AND OPTIMIZATION

Several mathematical models are relevant for general bug triaging and management processes. Thus, fuzzy sets are assessed and experimentally analyzed in articles [51], [52], [53], [54], and [55]. Moreover, the Knapsack programming was analyzed in paper [55], in connection to bug triaging. Additionally, a bug triaging process, which is based on genetic algorithms models, is proposed in article [56]. It is relevant to mention the probabilistic optimization models that are based on the behaviour of ants, which are approached in paper [57]. Most surveyed contributions pertain to the Eclipse bug tracker, and the implemented models generate a maximum accuracy of 86%. In this case, the researcher focuses on the development of the mathematical models, and also on the specification of the objective functions, which are used to process the bug reports data.

As an example, relative to the fuzzy modeling-related techniques, a fuzzy model of software bugs is necessary, along with the membership functions that are specified between software bug terms and respective developers for bug triaging purposes. Considering the optimization models for software bug triaging, the relevant optimization constraints are specified relative to the number of bugs that are resolved by a certain developer during a given period of time. This is particularly important, as time is an important parameter, especially relative to high priority and security bugs. Consequently, the specified and implemented mathematical model mediates the selection of the proper developer to resolve the processed bug reports.

The software bug triage techniques (SBT), which are based on the optimization concept, imply the adequate mathematical modeling of the SBT problems relative to the terms of the implied optimization model. As an example, an SBT software system that is based on the mechanism of Ant Colony Optimization (ACO) [58], presumes that the bug tossing graphs are generated using the historical bug resolution data. Furthermore, the ants are allowed to circulate through these tossing graphs, with the goal to detect the optimum paths, which determine the suggested developers for the resolution of the reported bugs. Additionally, an ACO-related model is based on certain calibration parameters, such as the number of ants, the number of iterations, the configuration of the developer's network, and the intensity of the ants' pheromone. Several knapsack optimization-related SBT techniques [59], involve that the bug fixing and developer metadata are transformed into the respective knapsacks. The capacity of knapsacks is determined by the time limit, which is allocated for the resolution of the software bugs. Thus, the items and knapsacks determine the number of bugs and developers, respectively.

Genetic algorithms-related optimization techniques [60] for software bug triaging imply that fitness functions are specified relative to the list of words that label and describe bugs. The calculation of similarity scores is performed for clustered centers, and the fitness functions pertain to the

developers' data. The maximum value of the similarity suggests a higher-grade membership in the cluster of developers, which resolve the provided software bugs. Moreover, greedy search-related optimization models [61] for software bug triaging are based on the creation of a search space by considering the data of particular developers, which are available for a specific timeframe in order to resolve the existing bugs. Moreover, concerning the bugs triaging process, the distance functions are utilized to compute the distance to all the available developers, and consequently assign the open bugs to the developers that are featured by the shortest distance.

The surveyed articles suggest that the mathematical-based models mediate efficiently identifying the necessary constraints. The computational performance of bug triaging represents the main issues of these techniques, and the relevant problematic is approached in the articles [41], [62], and [63]. The efficient real-time behaviour of the implemented models may be affected by particular situations, such as the removal of the developer from the respective project, or the possibility for the developer to leave the company.

The fuzzy logic-related bug triaging models imply that membership functions are specified to represent relationships between the newly reported software bugs, and the proper software developers. The relevant problematic is approached in articles [60], [61], and [63]. The creation of new software bugs implies that the similarity between the terms of newly reported bugs relative to the existing bug terms is computed. Following, fuzzy logic allows for the membership values to be computed in connection to the newly created bugs, to assign the proper developers. Thus, a greater value of the membership indicates a higher possibility of properly resolving the bugs. It is relevant to note that several recent studies, such as [60], [62], and [63], propose fuzzy logic models relative to multi-criteria decision-making and analysis. The surveyed contributions suggest that these are used in the realm of efficient software bug triaging systems.

The main advantage of fuzzy logic models is the implied simplicity, and also the reasonable amount of computation resources that are required [63]. Considering a varied set of skilled software developers, the consideration of fuzzy membership mediates the efficient selection of the optimal developer. The disadvantage of SBT approaches that are based on fuzzy logic models is represented by the relatively difficult development of the most relevant and logical membership function. Moreover, the extensive scientific survey that was conducted suggests that only a few relevant fuzzy logic-related models are reported. Consequently, the degree of scientific generality is reduced, in this case, and the possibility of considering this type of mathematical and algorithmic model in all real-world use cases is problematic.

F. BUG TRIAGING MODELS BASED ON DEEP LEARNING

Deep Learning represents a machine learning technique, which considers the natural "learn by example" strategy. Therefore, its algorithmic and computational apparatus may also be applied to software bug triaging (SBT). Thus, deep learning techniques, which are based on Convolutional Neural Networks (CNN) are described in articles [2], [64], and [65]. Moreover, models that are based on Recurrent Neural Networks (RNN) are discussed in articles [2] and [5]. These papers report contributions that are consistently used relative to software bug triaging. It is relevant to note that Convolutional Neural Networks and Recurrent Neural Networks represent popular approaches relative to deep learning models, which are considered to implement software bug triaging systems. The essential difference between CNN and RNN is determined by the densely connected feed-forward network, which determines CNN, while RNN considers a feed-backward network, which processes the data from the previous iteration in order to improve the weights. Thus, bug summary and description represent significant attributes, which are usually processed in general bug triaging processes. These are text-based attributes. Consequently, word embedding models, such as Word2Vec and Glove, are frequently used to implement software bug triagers. The surveyed articles report experiments that are conducted on various datasets, and the obtained accuracy values belong to the range 57% to 87% relative to the assessed DL-related models. The Word2Vec approach [66] is a word embedding technique that is frequently considered in the related scientific literature, as compared to the Glove model [67].

The considered CNN models are based on convolutional layers and pooling layers. The rationale behind these layers is represented by feature extraction. Classification is conducted during the last stage. Thus, relative to SBT, each software developer determines a category. Generally, DL-related models offer consistent computational performance, scalability, and learning rates, which is the time required for the model's training, relative to other AI-related models. Nevertheless, computational time and the mandatory features of the computing infrastructure represent key challenges. Thus, DL-related models require greater training time relative to traditional machine learning or information retrieval-related approaches [68]. This computational behaviour is determined by the multiple data processing layers, which have to be visited [69]. Therefore, organizations that cannot afford the more expensive computational infrastructures, may be forced to consider alternative solutions.

Regular multilayered CNN architectures, such as the one that is presented in article [64], consider DL-related software architectures for SBT, which are structured according to word vector representation layers, convolutional layer, pooling layer, and activation functions, which are necessary in order to aggregate the generated output values. The convolution layer conducts the training of the data samples, and also the training of the input data samples. The functional relevance of the pooling layer is to determine and extract the data samples from the feature space, which are relevant to the processed task. Moreover, it is relevant to note that Max pooling techniques, which are sample-based discretization processes, are

considered and proposed in the surveyed studies, considering that they provide consistent computational performance.

The surveyed papers also describe personalized deep learning models, such as graph recurrent convolutional neural networks (GRCNN) models [68], and reinforcement deep learning (RDL) models [69]. The graph recurrent convolutional neural network (GRCNN) model generates F1 scores of 86.74%, and 75.64% relative to the Eclipse and Mozilla projects bug trackers. Furthermore, the surveyed deep reinforcement learning approaches generate 52%, 54%, 68%, and 78% top-5 accuracy values considering the OpenOffice, NetBeans, Mozilla, and Eclipse bug trackers.

IV. PERFORMANCE EVALUATIONS

Performance determines a significant problematic for the assessment of any bug triaging approach, which was naturally approached by this extended study. Consequently, the relevant performance metrics suggest the real-world appropriateness of the proposed algorithmic approaches. As an example, relative to a classification-related bug management and triaging model, accuracy (ACC), precision (P), recall (R), and F-measure (F1) are defined through a confusion matrix [70].

The number of correct predictions divided by the total number of predictions makes up the classification accuracy measure, which is possibly the most straightforward to use and implement. The formula for the accuracy is presented in equation (1), where TP, TN, FP, and FN represent the true positives, true negatives, false positives, and false negatives, respectively.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

A multitude of studies consider this metric to validate the model [31], [94], [95], [96], [97], [98]. However, some of the mentioned studies also use other metrics for parallel comparisons, and/or validations, precisely because the accuracy itself presents some downfalls. Sometimes, it may be preferable to choose a model with lower accuracy, if it has a stronger ability to anticipate the outcome of the situation. For instance, if the real-world use case presents a significant class imbalance, a model can predict the value of the majority class for all predictions, and obtain a high level of classification accuracy. Nevertheless, the model may not be applicable to the problem at hand.

Therefore, the next time a classification problem is presented, one should be careful not to merely choose accuracy as a metric, and start creating the model straight away. Naturally, experimenting with the model is enticing, but it's necessary to take some time to understand the types of issues that should be solved, and the most suitable metrics. After clearing that up, one may be confident that the model that is created will be appropriate for the task at hand. This paradox is known as the Accuracy Paradox, and for such issues, extra measurements are needed to evaluate a classifier.

Further on, precision represents the ratio of relevant software developers relative to all the developers determined by the respective machine learning algorithm. Continued, the recall represents the ratio of relevant software developers relative to all the relevant software developers determined by the machine learning algorithm. The formulas for these metrics are presented in equation (2).

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN} \tag{2}$$

Both of these values should ideally be as high as they can be. That might not be possible, though. Precision will fall off when recall rises, and vice versa. Therefore, one must choose which indicators are particularly relevant while training the machine learning model. One crucial and time-consuming aspect of software maintenance is bug triaging. The model may be required not to provide a false result if it was used for an organization that wanted to determine whether the programmer is a good fit or not for an urgent bug. As a result, one would rather label suitable programmers as inappropriate programmers (false negative) than suitable programmers as inappropriate (false positive). In other words, a false negative is preferable to a false positive (recall).

Additionally, F1-measure (3) is a metric that combines precision and recall. The F1 measure is the best option if it is required to choose a model based on a balance between precision and recall.

$$F1 = \frac{2 \times P \times R}{P + R} \tag{3}$$

It is interesting to note that these metrics may be computed directly, or they can be computed using a confusion matrix [99].

In addition, other metrics have proved valuable for measuring the quality of results, metrics that are perhaps less well-known but of certain interest. For example, the authors of article [71] conduct a relatively consistent analysis of existing bug triaging techniques. Thus, the article surveys 74 studies, which are related to software bug triaging, and it includes a presentation of the metrics that were used in order to evaluate the analyzed software bug triaging models. Thus, the article emphasizes the metrics, which are considered in order to rank the available software developers. These metrics are top-K accuracy, Precision@K (P@K), Recall@K (R@K), mean reciprocal rank (MRR), and mean average precision (mAP). Here, K designates the number of recommended or available software developers. The top-K accuracy relates to the number of effective developers that have been tasked with the resolution of existing bugs, out of the K developers that are recommended. This is computed as the ratio of recommended developers divided by the total number of developers. Although there are numerous studies that use this metric, such as [101] and [102], the article [100] presents an experimental analysis concerning the trade-offs in top-k classification accuracies, and the losses related to the problematic of deep learning.

Furthermore, P@K is calculated as the ratio between the relevant (effective) recommended developers and the total number of recommended developers. Moreover, R@K is calculated as the ratio between the relevant (effective) recommended developers and the total number of relevant developers. Thus, P@K and R@K are computed according to the formulae described in equation (4), F1@K is represented in equation (5), and the average precision (AP) is described in equation (6).

$$\begin{aligned} P@K &= \frac{TP@K}{TP@K + FP@K}, \\ R@K &= \frac{TP@K}{TP@K + FN@K} \end{aligned} \tag{4}$$

$$F1@K = 2 \times \frac{P@K \times R@K}{P@K + R@K} \tag{5}$$

$$\begin{aligned} AP = \Sigma(&Recall@K - Recall@(K-1)) \\ &\times Precision@K \end{aligned} \tag{6}$$

Furthermore, DCG@K and NDCG@K represent two additional metrics, which are considered in order to evaluate recommender systems. DCG is an acronym for Discounted Cumulative Gain, and NDCG is the acronym for Normalized Discounted Cumulative Gain. These metrics are considered in order to assess the quality of the developers ranking during the software bug triaging process. The calculation of DCG@K and NDCG@K is conducted using the formulae described in equation (7). Here, rel_i represents the relevance of the i^{th} developer recommendation, while the IDCG determines the Discounted Cumulative Gain in ideal conditions.

$$\begin{aligned} DCG@K &= \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)}, \\ NDCG@K &= \frac{DCG@K}{IDCG@K} \end{aligned} \tag{7}$$

The surveyed articles suggest that, although contributions like [71] imply that mean average precision mAP represents the optimal quantitative assessment metric related to software bug triaging, the quantitative and qualitative analysis is also determined by the type of AI-related model, which is selected for the bug management and triaging process. Thus, if the algorithmic model is based on recommender system techniques, then DCG@K and NDCG@K are the optimal metrics to conduct the performance evaluation [48].

Furthermore, relative to machine learning and information retrieval-oriented models, accuracy, precision, recall, and F-measure are the recommended performance evaluation metrics [70].

V. SCIENTIFIC CHALLENGES AND OPEN RESEARCH QUESTIONS

The types of metrics, algorithmic and numerical approaches, which were surveyed in the previous sections, suggest that although numerous techniques exist for the design and implementation of automatic bug triaging and management techniques, there are also conceptual and practical challenges, which are discussed in this section, along with the proper suggested solutions. This section considers several categories of conceptual and real-world problems, which are relevant for future research efforts. Additionally, the extensive surveyed literature determines significant open research questions, which are discussed.

A. GENERATION OF DEVELOPERS VOCABULARY

The generation of a precise software developers' vocabulary represents a fundamental activity relative to the software bug triaging process. Thus, the design and development of software bug triaging techniques imply that the defining bug data and technical abilities of the software developers are taken into account during the automatic bug assignment process. Nevertheless, apart from the technical abilities of the involved software developers, recent articles, such as [72] and [73], consider other relevant aspects. Thus, there are certain software engineers or developers that can be classified as ''experts'', and may be consequently considered during the optimization of the developer vocabulary generation process, which relates to upcoming software bug triaging techniques.

B. DATA REDUCTION MODELS

The contribution that is described in article [74] reports an approach for filtering and eliminating invalid bugs from the processed bug trackers. This optimizes the amount of bug triaging data, and consequently ameliorates the necessary time and energy, which are necessary for the proper management and effective bug triaging processes. Moreover, the paper defines four feature groups, which are the experience of the bug reporter, the related collaboration network, the degree of technical completeness, and the defining bug text. Consequently, a Random Forest classifier was designed to assign the invalid bug to the proper category. Additionally, the paper has the merit to conduct a rather useful overview on interesting approaches for software bug triaging. It is also relevant to mention the contribution reported in paper [75], which aims to detect and properly manage the non-reproducible bugs, in which identification and handling of respective bugs are conducted relative to the processed bug tracker. This may sensibly reduce the processing time of non-reproducible bugs, which are generally difficult to manage in an adequate manner.

C. MODELS REGARDING BUG PRIORITIZATION

The contribution that is described in article [40] concerns the prioritization of security bugs, which is regarded as an essential process in the scope of software development. Thus, the implementation of security patches determines a mandatory effort in the process of software development, as it implements the required security mechanisms. The reported work is based on reducing the size of the training data, which is logically enriched and further improved with a transfer learning model. The article demonstrates that this type of

approach may be considered relative to the relevant software bug triaging processes.

D. MODELS RELATED TO FEATURES SELECTION AND AGGREGATION

Feature selection, aggregation, and ranking [76] determines a task that is usually performed during the preprocessing phase of the software bug triaging process. Thus, article [77] describes an ontology for an efficient location-based knowledge extraction relative to the implementation phase of software development. The relevant ontology may be considered for bug triaging during the subsequent phases of the software bug triaging process. Furthermore, the generation of user summaries, which are useful to recommend similar projects during the software development is proposed in article [27], which can be aggregated and used together with other algorithmic model to implement the software bug triaging process. Additionally, another contribution was reported in article [78], which described an approach for the generation of a text summary. This is used to assess the opportunity to escalate a ticket, to automatically generate the title and content of the ticket, and also to assign the ticket to a properly available developer. This approach brings obvious advantages to the bug triaging process. Furthermore, aside from the classical feature selection models, feature enhancement is also conducted relative to software bug triaging in article [38], which also presents further useful features and metrics relative to the software bug triaging process.

E. RELEVANT METRICS

The comprehensive contributions that are reported in articles [51], [79], and [80] discuss four quality metrics, which influence the quality and computational efficiency of bug triaging processes. Thus, the relevant quality metrics are complexity, cohesion, inheritance, and coupling, which were analytically assessed in the mentioned articles. The conceptual and real-world analysis that determines this scientific survey implies that future research efforts should concentrate on other software quality features, such as maintainability, reusability, and refactoring.

F. ANALYSIS OF LARGE BUG DATASETS

The majority of surveyed papers [77] consider public bug trackers, such as Mozilla, Apache, Google, and Git, in order to assess the validity and computational efficiency of the proposed software bug triaging models. Consequently, we suggest that the reference performance assessment datasets should be diversified, while the automatic software bug triaging models should be algorithmically enriched with up-to-date techniques, such as big data analytics.

G. MODELS RELATED TO THE STUDY OF NETWORKED AND GRAPH STRUCTURES

The authors of article [38] suggest the consideration of semantic, multiplex, and multimode networks, which are useful to model the relationships between the components of bug items components. This supports the design and development of future enhanced software bug triaging models. Moreover, paper [81] describes the creation of a visual analysis tool that is related to a Directed Acyclic Graph (DAG), which may be considered to model the logical relationship between the involved developers. Article [82] reports a model that is called iTriage, which creates a sequential model that supports the processing of textual features, and also the relevant bugs tossing sequences. The relevant conceptual and practical suggestions should be considered for the creation of future enhanced software bug triaging systems, which properly implement the bug assignment and management routines. It is relevant to note that article [41] proposes a logically related approach that considers the GitHub bugs tracking repository, which presumes that semantically related issues are properly recorded in the analyzed GitHub bugs dataset. The reported scientific developments may be considered to efficiently identify similar bugs, and consequently determine relevant information for the resolution of critical and high-priority bugs. Furthermore, the logical mechanism of bug dependency, and also dependency graphs may also be considered for the enhancement of software bug triaging routines, as it is demonstrated in article [83].

H. MODELS THAT CONSIDER CROSS REPOSITORY ANALYSES

Several papers consider the experimental and performance analysis mechanism of cross-repository analysis. Thus, paper [84] proposes an automatic commit message generation model, which pertains to version control systems. Thus, the text of the commit message was automatically generated considering previous software commit data, which was stored in a comprehensive repository. Moreover, a further relevant model is described in article [85], which proposed a customized recommender system that is related to the processing of open-source repositories. Additionally, an interest measurement mechanism, which is based on the processing of GitHub data, is specified relative to the developers that work on a project that is linked, considering the common technical abilities of relevant software developers. These and similar existing contributions propose conceptually relevant algorithmic models, which should be considered in order to design future software bug triaging models.

I. MODELS THAT CONSIDER DEEP LEARNING TECHNIQUES

Deep Learning (DL) models are created and reported by several contributions that pertain to software bug triaging. Thus, the Deep Learning-related Convolutional Neural Networks (CNN) model is utilized, together with the Word2vec word embedding model, in order to implement the bug triaging processes [64]. Apart from the described algorithmic model, DL determines a consistent algorithmic and functional apparatus, which can be used in future

software bug triaging approaches. Variants of CNN, such as bio-inspired spiking CNN (SCNN), seem to provide superior computational performance than classical CNN networks. Consequently, SCNN networks can be considered in order to implement relevant computational routines for software bug triaging [46]. Moreover, article [44] analyzes the technological profile recommendations relative to various document embedding models, while paper [86] addresses the problematic of the software developers' skills prediction considering a multi-label classification of available resumes data, which is based on the usage of CNN networks with model predictions.

J. CONTEMPORARY MACHINE LEARNING AND BIG DATA APPROACHES

Contemporary machine learning models like ensemble learning and transfer learning are usually coupled with big data analytics methods, which can be considered in order to further improve the software bug triaging approaches. Thus, the model of label prediction in bug repositories is studied in [17], while the automatic identification of technically skilled software engineers are discussed, designed, and implemented in paper [73]. Moreover, various software fault prediction models are explored in article [87], while an automated issue assignment model is proposed by the contribution reported in paper [88]. Furthermore, it is interesting to note that relevant blockchain-related models are also analyzed and described in article [62], with a clear emphasis on the software bug triaging processes.

K. EFFICIENT MANAGEMENT OF IMBALANCED DATASETS

The subject of imbalanced classes represents a theoretical and practical problem in the general field of artificial intelligence. Relative to software bug triaging, class imbalancing manifests when the bug items are not assigned to the developers in a balanced manner. More precisely, the abnormal situation occurs when a few software engineers fix a majority of the reported bugs. This situation may determine a class imbalancing in the training dataset, which is considered relative to the bug triaging process. Thus, several papers explore the relevant problematic from several perspectives, including the defect prediction algorithmic models [89], [90], [91], [92]. The surveyed papers also approach the topics of bug fixability prediction [93], and also the general problem of bugs classification [23], [24]. Nevertheless, there are only a few systematic studies concerning the relevant research aspects and open questions in the scope of automatic bug triaging and management. Consequently, this survey contributes to filling a gap, as it systematically approaches the essential aspects and existing contributions, which are related to software bug triaging and management. The issue of imbalanced datasets may be further approached using deep learning and soft computing techniques, which combine artificial intelligence models, and also natural selection approaches.

VI. RESEARCH QUESTIONS AND DEGREE OF ACCOMPLISHMENT

Artificial intelligence is perceived and practically demonstrated as a novel approach to the problematic of design and implementation of precise and automated software bug triaging systems. The vast array of analyzed scientific contributions suggests that artificial intelligence and machine learning can be considered effective tools in order to design and implement accurate and computationally efficient automated software bug triaging systems. Nevertheless, the continuous technological developments, and also certain conceptual and practical issues, which were addressed in the previous sections, suggest that there is still enough room for further optimizations and functional extensions. Consequently, the following paragraphs discuss the three fundamental research questions, which guided this scientific survey effort and also analyze the degree of scientific achievement of this paper.

A. RQ1: IS AI CAPABLE TO EFFECTIVELY ENHANCE AND AUTOMATE SOFTWARE BUG TRIAGING?

Regarding the thorough scientific review that was conducted, several pertinent remarks can be made concerning the consideration of AI-related models to design and implement software bug triaging approaches. Thus, the discussion considers three perspectives relative to software bug triaging systems, which this section considers. The first perspective pertains to AI-related models for bug triaging. The identified scientific contributions suggest that machine learning models have been a popular choice during the past ten years for software bug triaging. Following, as this paper already demonstrated, information retrieval-related models were considered, chronologically followed by approaches that are based on recommender systems. Furthermore, contemporary studies have started to consistently rely on deep learning-related solutions. The significant changes regarding the conceptual and practical paradigms are easily discernible considering the scientific literature, which this article surveys. Thus, deep learning-related approaches have been demonstrated to enhance related software bug triaging routines, both considering the accuracy, as it is measured by various metrics and also the computational efficiency. Nevertheless, future developments are expected to address the remaining issues, which are also connected to the accuracy and computational efficiency of relevant bug triaging processes. These currently hamper the consideration of some deep learning-related approaches in the context of certain large or structurally complex real-world software bug triaging use cases.

It is immediate to observe that automated software bug triaging will minimize the time that is necessary for software development processes, and concomitantly, the software development costs will be reduced. Consequently, the economical viability of the software development projects and companies will be strengthened. This assertion is comprehensively approached and demonstrated in the surveyed

literature. Therefore, it can be inferred that AI-related models are not only relevant, but essential for the implementation of automated software bug triaging systems.

B. RQ2: IDENTIFICATION OF SIGNIFICANT PERFORMANCE PARAMETERS AND METRICS CONSIDERED BY EXISTING CONTRIBUTIONS

The second perspective relates to the comprehensive analysis of performance evaluation techniques, parameters, and metrics, which are considered for AI-related software bug triaging models. The relevance of the second analysis perspective is determined by the fact that the selection of performance parameters should be carefully conducted, in accordance with the technological model, which is used to design and implement the relevant software bug triaging techniques. More precisely, relative to a bug triaging technique that is based on classification algorithms, accuracy, precision, recall, and F-measure are specified through the consideration of a confusion matrix. Furthermore, developer ranking algorithms consider performance metrics like top-K accuracy, Precision@K, Recall@K, mean reciprocal rank, and mean average precision, which is fully covered in this paper. Here, K designates the number of recommended software developers. Additionally, Discounted Cumulative Gain, and Normalized Discounted Cumulative Gain, which have already been covered, represent significant performance metrics, which are used in order to evaluate software bug triaging approaches, which are based on recommender systems.

Although certain contributions, such as the one that is reported in article [71], suggest that the mean average precision may be the optimal performance evaluation metric relative to the relevant software bug triaging processes, the assessment should consider the specific AI model that is used to implement the bug triaging process. As an example, in the case of bug triagers that are based on recommender systems, then Discounted Cumulative Gain, and Normalized Discounted Cumulative Gain was reported to be the proper performance evaluation metrics. Furthermore, in the case of bug triagers that are based on machine learning and information retrieval models, accuracy, precision, recall, and F-measure were reported to be the suitable metrics, both considering their validity for the intended type of measurement and their computational efficiency. It can be asserted that this scientific survey reached the goal to determine and comparatively analyze the most suitable performance parameters and metrics, in the context of software bug triaging.

C. RQ3: OPEN SCIENTIFIC PROBLEMS AND POSSIBLE FUTURE DEVELOPMENTS TO ENHANCE SOFTWARE BUG TRIAGING PROCESSES

The third perspective is defined by the open scientific research problems, which could define possible future research pathways in the scope of software bug triaging. The open research problems are presented in the previous section, along with clear indications regarding the possible future research pathways. Thus, eleven categories concerning the future possible research subjects are defined. These categories pertain to data reduction techniques, developers' vocabulary generation, feature selection and aggregation, bug prioritization, performance parameters and metrics, exploration of large bug data sets (repositories), networks and graph-related models, advanced deep learning-related models, cross repository analysis, contemporary machine learning and big data-related approaches, and the proper management of imbalanced datasets. The comprehensive scientific literature that was surveyed suggests that, in spite of the existing functional approaches that were proposed for automatic software bug triaging, there are numerous conceptual and practical aspects that need to be further addressed. Considering the semantics and perspectives that are determined by the comprehensive literature that was surveyed, it is possible to objectively assert that the most relevant open scientific problems, research gaps, and future research pathways were properly identified and discussed in this paper.

VII. CONCLUSION

Software bug triaging determines a significant research scope, considering its conceptual and real-world implications. Consequently, the relevant scientific literature encompasses various contributions, which have been thoroughly surveyed and presented. Thus, AI-related models are generally considered in order to implement automatic software bug triaging and management systems. The present article describes an extended survey, which systematically analyzes the most relevant contributions that are related to software bug triaging. The scientific survey is structured according to a systematic approach, which selects the relevant existing articles based on the principles of the PRISMA scientific review system. Consequently, this paper comprehensively surveys, classifies, and analyzes relevant software bug triaging and management approaches, which are reported in the existing literature. The identified papers are analytically and comparatively evaluated, and three fundamental research questions are defined and discussed. Consequently, a three-dimensional evaluation and comparative analysis are conducted relative to the identified software bug triaging contributions, which consider the defined research questions. Consequently, an evaluation of the identified performance parameters and metrics is conducted and included, as a separate section, in this paper.

Furthermore, relevant research questions, which are currently unsatisfactorily approached, are presented and analyzed, and possible future research and practical trends related to software bug triaging are discussed. Considering each surveyed bug triaging approach, the identified advantages and problems are discussed and evaluated, both considering their conceptual relevance and their importance for real-world software development processes. The relevance of the problematic approached in this survey paper is

further justified by the economic implications of the implied bug triaging and management processes relative to the overall software development efforts. Thus, the consideration of algorithmic models that relate to artificial intelligence and machine learning has essentially changed the paradigm of software bug triaging and management. The surveyed approaches significantly enhance the relevant software bug triaging and management processes, and the software development times and costs are consistently reduced. Nevertheless, several conceptual and practical issues remain, which are thoroughly presented. Therefore, future research efforts should carefully approach the remaining problems, in order to fully establish the automatic software bug triaging and management as an accurate and computationally efficient practical solution.

REFERENCES

[1] D. Moher, A. Liberati, J. Tetzlaff, and D. G. Altman, "Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement," *Ann. Internal Med.*, vol. 151, no. 4, pp. 264–269, 2009.

[2] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. ACM India Joint Int. Conf. Data Sci. Manage. Data*, Jan. 2019, pp. 171–179.

[3] H. Mohsin and C. Shi, "SPBC: A self-paced learning model for bug classification from historical repositories of open-source software," *Expert Syst. Appl.*, vol. 167, Apr. 2021, Art. no. 113808.

[4] J. A. Nasir, O. S. Khan, and I. Varlamis, "Fake news detection: A hybrid CNN-RNN based deep learning approach," *Int. J. Inf. Manage. Data Insights*, vol. 1, no. 1, Apr. 2021, Art. no. 100007.

[5] U. Koc, S. Wei, J. S. Foster, M. Carpuat, and A. A. Porter, "An empirical assessment of machine learning approaches for triaging reports of a Java static analysis tool," in *Proc. 12th IEEE Conf. Softw. Test., Validation Verification (ICST)*, Apr. 2019, pp. 288–299.

[6] M. Sharma, A. Tandon, M. Kumari, and V. B. Singh, "Reduction of redundant rules in association rule mining-based bug assignment," *Int. J. Rel., Qual. Saf. Eng.*, vol. 24, no. 6, Dec. 2017, Art. no. 1740005.

[7] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.*, 2004, pp. 1–6.

[8] A. Goyal and N. Sardana, "Machine learning or information retrieval techniques for bug triaging: Which is better?" *E-Informatica Softw. Eng. J.*, vol. 11, no. 1, pp. 117–141, 2017.

[9] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221.

[10] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "DREX: Developer recommendation with K-nearest-neighbor search and expertise ranking," in *Proc. 18th Asia–Pacific Softw. Eng. Conf.*, Dec. 2011, pp. 389–396.

[11] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *J. Softw., Evol. Process*, vol. 24, no. 1, pp. 3–33, Jan. 2012.

[12] S. Banerjee, Z. Syed, J. Helmick, M. Culp, K. Ryan, and B. Cukic, "Automated triaging of very large bug repositories," *Inf. Softw. Technol.*, vol. 89, pp. 1–13, Sep. 2017.

[13] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 2–11.

[14] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Improving automatic bug assignment using time-metadata in term-weighting," *IET Softw.*, vol. 8, no. 6, pp. 269–278, Dec. 2014.

[15] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "A time-based approach to automatic bug report assignment," *J. Syst. Softw.*, vol. 102, pp. 109–122, Apr. 2015.

[16] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *J. Softw.*, vol. 8, no. 9, pp. 2185–2190, Sep. 2013.

[17] J. M. Alonso-Abad, C. López-Nozal, J. M. Maudes-Raedo, and R. Marticorena-Sánchez, "Label prediction on issue tracking systems using text mining," *Prog. Artif. Intell.*, vol. 8, no. 3, pp. 325–342, Sep. 2019.

[18] P. Ardimento, N. Boffoli, and C. Mele, "A text-based regression approach to predict bug-fix time," in *Complex Pattern Mining*. 2020, doi: 10.1007/978-3-030-36617-9_5.

[19] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Softw. Eng.*, vol. 21, no. 2, pp. 368–410, Apr. 2016.

[20] A. Kaur and S. G. Jindal, "Text analytics based severity prediction of software bugs for apache projects," *Int. J. Syst. Assurance Eng. Manage.*, vol. 10, no. 4, pp. 765–782, Aug. 2019.

[21] L. Chen, X. Wang, and C. Liu, "Improving bug assignment with bug tossing graphs and bug similarities," in *Proc. Int. Conf. Biomed. Eng. Comput. Sci.*, 2011, pp. 421–427, doi: 10.1109/ICBECS.2010.5462287.

[22] L. Chen, X. Wang, and C. Liu, "An approach to improving bug assignment with bug tossing graphs and bug similarities," *J. Softw.*, vol. 6, no. 3, pp. 421–427, Mar. 2011.

[23] M. Chen, D. Hu, T. Wang, J. Long, G. Yin, Y. Yu, and Y. Zhang, "Using document embedding techniques for similar bug reports recommendation," in *Proc. IEEE 9th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2018, pp. 811–814, doi: 10.1109/ICSESS.2018.8663849.

[24] R. Chen, S.-K. Guo, X.-Z. Wang, and T.-L. Zhang, "Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 12, pp. 2406–2420, Dec. 2019.

[25] D. Hu, M. Chen, T. Wang, J. Chang, G. Yin, Y. Yu, and Y. Zhang, "Recommending similar bug reports: A novel approach using document embedding model," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2018, pp. 725–726.

[26] J. Jiang, D. Lo, J. Zheng, X. Xia, Y. Yang, and L. Zhang, "Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction," *J. Syst. Softw.*, vol. 154, pp. 196–210, Aug. 2019.

[27] M. R. Resketi, H. Motameni, H. Nematzadeh, and E. Akbari, "Automatic summarising of user stories in order to be reused in future similar projects," *IET Softw.*, vol. 14, no. 6, pp. 711–723, Dec. 2020.

[28] J.-W. Park, M.-W. Lee, J. Kim, S.-W. Hwang, and S. Kim, "CosTriage: A cost-aware triage algorithm for bug reporting systems," in *Proc. Nat. Conf. Artif. Intell.*, 2011, p. 139.

[29] D.-G. Lee and Y.-S. Seo, "Improving bug report triage performance using artificial intelligence based document generation model," *Hum.-Centric Comput. Inf. Sci.*, vol. 10, no. 1, Dec. 2020, Art. no. 26.

[30] T. S. Roopa, Y. Purna, and C. Krish, "A novel approach for bug triaging with specialized topic model," *Int. J. Innov. Technol. Exploring Eng.*, vol. 8, no. 7, pp. 1032–1038, 2019.

[31] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Trans. Softw. Eng.*, vol. 43, no. 3, pp. 272–297, Mar. 2017, doi: 10.1109/TSE.2016.2576454.

[32] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution," in *Proc. 8th Int. Conf. Predictive Models Softw. Eng.*, Sep. 2012, pp. 19–28.

[33] W. Zhang, S. Wang, and Q. Wang, "BAHA: A novel approach to automatic bug report assignment with topic modeling and heterogeneous network analysis," *Chin. J. Electron.*, vol. 25, no. 6, pp. 1011–1018, Nov. 2016.

[34] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.

[35] W. Zhang, Y. Cui, and T. Yoshida, "En-LDA: An novel approach to automatic bug report assignment with entropy optimized latent Dirichlet allocation," *Entropy*, vol. 19, no. 5, p. 173, Apr. 2017.

[36] G. Brookes and T. McEnery, "The utility of topic modelling for discourse studies: A critical evaluation," *Discourse Stud.*, vol. 21, no. 1, pp. 3–21, Feb. 2019.

[37] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proc. 6th IEEE Int. Work. Conf. Mining Softw. Repositories*, May 2009, pp. 131–140.

[38] I. Alazzam, A. Aleroud, Z. Al Latifah, and G. Karabatis, "Automatic bug triage in software systems using graph neighborhood relations for feature augmentation," *IEEE Trans. Computat. Social Syst.*, vol. 7, no. 5, pp. 1288–1303, Oct. 2020.

[39] J. Xuan, H. Jiang, H. Zhang, and Z. Ren, "Developer recommendation on bug commenting: A ranking approach for the developer crowd," *Sci. China Inf. Sci.*, vol. 60, no. 7, Jul. 2017, Art. no. 072105.

[40] S. Mostafa, B. Findley, N. Meng, and X. Wang, "Sais: Self-adaptive identification of security bug reports," *IEEE Trans. Depend. Sec. Comput.*, vol. 18, no. 4, pp. 1779–1792, Jul. 2021.

[41] W. Zhang, S. Wang, and Q. Wang, "KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity," *Inf. Softw. Technol.*, vol. 70, pp. 68–84, Feb. 2016.

[42] Y. Zhang, Y. Wu, T. Wang, and H. Wang, "A novel approach for recommending semantically linkable issues in GitHub projects," *Sci. China Inf. Sci.*, vol. 62, no. 9, pp. 1–3, Sep. 2019.

[43] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.

[44] P. Chamoso, G. Hernández, A. González-Briones, and F. J. García-Peñalvo, "Recommendation of technological profiles to collaborate in software projects using document embeddings," *Neural Comput. Appl.*, vol. 34, no. 11, pp. 8423–8430, Jun. 2022.

[45] A. Ray, M. H. Kolekar, R. Balasubramanian, and A. Hafiane, "Transfer learning enhanced vision-based human activity recognition: A decade-long analysis," *Int. J. Inf. Manage. Data Insights*, vol. 3, no. 1, Apr. 2023, Art. no. 100142.

[46] S. F. A. Zaidi, F. M. Awan, M. Lee, H. Woo, and C.-G. Lee, "Applying convolutional neural networks with different word representation techniques to recommend bug fixers," *IEEE Access*, vol. 8, pp. 213729–213747, 2020.

[47] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 25–35.

[48] G. Shani and A. Gunawardana, "Evaluating recommendation systems," in *Recommender Systems Handbook*. Springer, 2011, pp. 257–297, doi: 10.1007/978-0-387-85820-3_8.

[49] N. Mishra, S. Chaturvedi, A. Vij, and S. Tripathi, "Research problems in recommender systems," *J. Phys., Conf. Ser.*, vol. 1717, no. 1, Jan. 2021, Art. no. 012002.

[50] M. K. Najafabadi, A. H. Mohamed, and M. N. Mahrin, "A survey on data mining techniques in recommender systems," *Soft Comput.*, vol. 23, no. 2, pp. 627–654, Jan. 2019.

[51] R. Kumar, A. I. Khan, Y. B. Abushark, M. M. Alam, A. Agrawal, and R. A. Khan, "A knowledge-based integrated system of hesitant fuzzy set, AHP and TOPSIS for evaluating security-durability of web applications," *IEEE Access*, vol. 8, pp. 48870–48885, 2020.

[52] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging (NIER track)," in *Proc. 19th ACM SIGSOFT Symp., 13th Eur. Conf. Found. Softw. Eng.*, 2021, pp. 884–887.

[53] A. Goyal and N. Sardana, "Empirical analysis of ensemble machine learning techniques for bug triaging," in *Proc. 12th Int. Conf. Contemp. Comput. (IC3)*, Aug. 2019, pp. 1–6.

[54] M. Wei, S. Guo, R. Chen, and J. Gao, "Enhancing bug report assignment with an optimized reduction of training set," in *Proc. Int. Conf. Knowl. Sci., Eng. Manag.*, in Lecture Notes in Artificial Intelligence, vol. 11062, 2018, pp. 36–47.

[55] Y. Kashiwa and M. Ohira, "A release-aware bug triaging method considering developers' bug-fixing loads," *IEICE Trans. Inf. Syst.*, vol. E103.D, no. 2, pp. 348–362, 2020.

[56] J. Lee, D. Kim, and W. Jung, "Cost-aware clustering of bug reports by using a genetic algorithm," *J. Inf. Sci. Eng.*, vol. 35, no. 1, pp. 175–200, 2019.

[57] V. Akila, V. Govindasamy, and S. Sharmila, "Bug Triage based on ant system with evaporation factor tuning," *Int. J. Control Theory Appl.*, vol. 9, no. 2, pp. 859–863, 2016.

[58] Md. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in *Proc. 3rd Int. Symp. Empirical Softw. Eng. Meas.*, Oct. 2009, pp. 439–442.

[59] S. G. Jindal and A. Kaur, "Automatic keyword and sentence-based text summarization for software bug reports," *IEEE Access*, vol. 8, pp. 65352–65370, 2020.

[60] R. R. Panda and N. K. Nagwani, "Classification and intuitionistic fuzzy set based software bug triaging techniques," *J. King Saud Univ., Comput. Inf. Sci.*, vol. 34, no. 8, pp. 6303–6323, Sep. 2022.

[61] P. M. Vu, T. T. Nguyen, and T. T. Nguyen, "Fuzzy multi-intent classifier for user generated software documents," in *Proc. ACM Southeast Conf.*, Apr. 2020, pp. 292–295.

[62] C. Gupta and M. M. Freire, "A decentralized blockchain oriented framework for automated bug assignment," *Inf. Softw. Technol.*, vol. 134, Jun. 2021, Art. no. 106540.

[63] R. R. Panda and N. K. Nagwani, "Multi-label software bug categorisation based on fuzzy similarity," *Int. J. Comput. Sci. Eng.*, vol. 24, no. 3, pp. 244–258, 2021.

[64] S. Guo, X. Zhang, X. Yang, R. Chen, C. Guo, H. Li, and T. Li, "Developer activity motivated bug triaging: Via convolutional neural network," *Neural Process. Lett.*, vol. 51, no. 3, pp. 2589–2606, Jun. 2020.

[65] Y. Liu, J. X. Huang, and Y. T. Ma, "An automatic method using hybrid neural networks and attention mechanism for software bug triaging," *J. Comput. Res. Develop.*, vol. 57, no. 3, p. 461, 2020.

[66] C. A. Choquette-Choo, D. Sheldon, J. Proppe, J. Alphonso-Gibbs, and H. Gupta, "A multi-label, dual-output deep neural network for automated bug triaging," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 937–944.

[67] M. A. Wani, F. A. Bhat, S. Afzal, and A. I. Khan, *Advances in Deep Learning*. Springer, 2020.

[68] H. Wu, Y. Ma, Z. Xiang, C. Yang, and K. He, "A spatial–temporal graph neural network framework for automated software bug triaging," *Knowl.-Based Syst.*, vol. 241, Apr. 2022, Art. no. 108308.

[69] Y. Liu, X. Qi, J. Zhang, H. Li, X. Ge, and J. Ai, "Automatic bug triaging via deep reinforcement learning," *Appl. Sci.*, vol. 12, no. 7, p. 3565, Mar. 2022.

[70] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques* (The Morgan Kaufmann Series in Data Management Systems), vol. 5, no. 4, 3rd ed. 2011, pp. 83–124.

[71] A. Sajedi-Badashian and E. Stroulia, "Guidelines for evaluating bug-assignment research," *J. Softw., Evol. Process*, vol. 32, no. 9, Sep. 2020, Art. no. e2250.

[72] E. Kalliamvakou, C. Bird, T. Zimmermann, A. Begel, R. DeLine, and D. M. German, "What makes a great manager of software engineers?" *IEEE Trans. Softw. Eng.*, vol. 45, no. 1, pp. 87–106, Jan. 2019.

[73] P. L. Li, A. J. Ko, and A. Begel, "What distinguishes great software engineers?" *Empirical Softw. Eng.*, vol. 25, no. 1, pp. 322–352, Jan. 2020.

[74] Y. Fan, X. Xia, D. Lo, and A. E. Hassan, "Chaff from the wheat: Characterizing and determining valid bug reports," *IEEE Trans. Softw. Eng.*, vol. 46, no. 5, pp. 495–525, May 2020.

[75] A. Goyal and N. Sardana, "An empirical study of non-reproducible bugs," *Int. J. Syst. Assurance Eng. Manage.*, vol. 10, no. 5, pp. 1186–1220, Oct. 2019.

[76] B. Alkhazi, A. DiStasi, W. Aljedaani, H. Alrubaye, X. Ye, and M. W. Mkaouer, "Learning to rank developers for bug report assignment," *Appl. Soft Comput.*, vol. 95, Oct. 2020, Art. no. 106667.

[77] J. R. Martínez-García, F.-E. Castillo-Barrera, R. R. Palacio, G. Borrego, and J. C. Cuevas-Tello, "Ontology for knowledge condensation to support expertise location in the code phase during software development process," *IET Softw.*, vol. 14, no. 3, pp. 234–241, Jun. 2020.

[78] M. Nayebi, L. Dicke, R. Ittyipe, C. Carlson, and G. Ruhe, "ESSMArT way to manage customer requests," *Empirical Softw. Eng.*, vol. 24, no. 6, pp. 3755–3789, Dec. 2019.

[79] L. Kumar, S. Tummalapalli, and L. B. Murthy, "An empirical framework to investigate the impact of bug fixing on internal quality attributes," *Arabian J. Sci. Eng.*, vol. 46, no. 4, pp. 3189–3211, Apr. 2021, doi: 10.1007/S13369-020-05095-0.

[80] S. Kumar, A. K. Kar, and P. V. Ilavarasan, "Applications of text mining in services management: A systematic literature review," *Int. J. Inf. Manage. Data Insights*, vol. 1, no. 1, Apr. 2021, Art. no. 100008.

[81] Y. Kim, J. Kim, H. Jeon, Y.-H. Kim, H. Song, B. Kim, and J. Seo, "Githru: Visual analytics for understanding software development history through git metadata analysis," *IEEE Trans. Vis. Comput. Graphics*, vol. 27, no. 2, pp. 656–666, Feb. 2021, doi: 10.1109/TVCG.2020.3030414.

[82] S.-Q. Xi, Y. Yao, X.-S. Xiao, F. Xu, and J. Lv, "Bug triaging based on tossing sequence modeling," *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 942–956, Sep. 2019.

[83] R. Almhana and M. Kessentini, "Considering dependencies between bug reports to improve bugs triage," *Automated Softw. Eng.*, vol. 28, no. 1, pp. 1–26, May 2021.

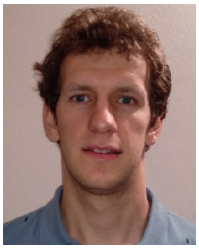
[84] Y. Huang, N. Jia, H.-J. Zhou, X.-P. Chen, Z.-B. Zheng, and M.-D. Tang, "Learning human-written commit messages to document code changes," *J. Comput. Sci. Technol.*, vol. 35, no. 6, pp. 1258–1277, Nov. 2020.
[85] C. Yang, Q. Fan, T. Wang, G. Yin, X.-H. Zhang, Y. Yu, and H.-M. Wang, "RepoLike: Amulti-feature-based personalized recommendation approach for open-source repositories," *Frontiers Inf. Technol. Electron. Eng.*, vol. 20, no. 2, pp. 222–237, Feb. 2019.
[86] K. F. F. Jiechieu and N. Tsopze, "Skills prediction based on multi-label resume classification using CNN with model predictions explanation," *Neural Comput. Appl.*, vol. 33, no. 10, pp. 5069–5087, May 2021.
[87] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artif. Intell. Rev.*, vol. 51, no. 2, pp. 255–327, Feb. 2019.
[88] E. U. Aktas and C. Yilmaz, "Automated issue assignment: Results and insights from an industrial case," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3544–3589, Sep. 2020.
[89] K. K. Bejjanki, J. Gyani, and N. Gugulothu, "Class imbalance reduction (CIR): A novel approach to software defect prediction in the presence of class imbalance," *Symmetry*, vol. 12, no. 3, p. 407, Mar. 2020.
[90] L. Gong, S. Jiang, and L. Jiang, "Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering," *IEEE Access*, vol. 7, pp. 145725–145737, 2019.
[91] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 46, no. 11, pp. 1200–1219, Nov. 2020.
[92] Z. Yuan, X. Chen, Z. Cui, and Y. Mu, "ALTRA: Cross-project software defect prediction via active learning and TrAdaBoost," *IEEE Access*, vol. 8, pp. 30037–30049, 2020.
[93] A. Goyal and N. Sardana, "NRFixer: Sentiment based model for predicting the fixability of non-reproducible bugs," *E-Informatica Softw. Eng. J.*, vol. 11, no. 1, pp. 109–122, 2017.
[94] H. A. Ahmed, N. Z. Bawany, and J. A. Shamsi, "CaPBug—A framework for automatic bug categorization and prioritization using NLP and machine learning algorithms," *IEEE Access*, vol. 9, pp. 50496–50512, 2021, doi: 10.1109/ACCESS.2021.3069248.
[95] P. Oliveira, R. M. C. Andrade, I. Barreto, T. P. Nogueira, and L. M. Bueno, "Issue auto-assignment in software projects with machine learning techniques," in *Proc. IEEE/ACM 8th Int. Workshop Softw. Eng. Res. Ind. Pract. (SER IP)*, Madrid, Spain, Jun. 2021, pp. 65–72, doi: 10.1109/SER-IP52554.2021.00018.
[96] M. Panda and A. T. Azar, "Hybrid multi-objective Grey Wolf search optimizer and machine learning approach for software bug prediction," in *Handbook of Research on Modeling, Analysis, and Control of Complex Systems*. 2020, doi: 10.4018/978-1-7998-5788-4.
[97] Y. Shi, Y. Mao, T. Barnes, M. Chi, and T. W. Price, "More with less: Exploring how to use deep learning effectively through semi-supervised learning for automatic bug detection in student code," *Int. Educ. Data Mining Soc*. [Online]. Available: https://par.nsf.gov/biblio/10340894
[98] A. Yadav, "Bug assignment-utilization of metadata features along with feature selection and classifiers," in *Applications of Artificial Intelligence and Machine Learning*. 2021, doi: 10.1007/978-981-16-3067-5_7.
[99] Z. J. Szamosvölgyi, E. T. Váradi, Z. Tóth, J. Jász, and R. Ferenc, "Assessing ensemble learning techniques in bug prediction," in *Computational Science and Its Applications—ICCSA 2021* (Lecture Notes in Computer Science), vol. 12955. Springer, 2021, doi: 10.1007/978-3-030-87007-2_26.
[100] A. Sawada, E. Kaneko, and K. Sagi, "Trade-offs in top-k classification accuracies on losses for deep learning," 2020, *arXiv:2007.15359*.
[101] V. Nath, D. Sheldon, and J. Alphonso-Gibbs, "Principal component analysis and entropy-based selection for the improvement of bug triage," in *Proc. 20th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Pasadena, CA, USA, Dec. 2021, pp. 541–546, doi: 10.1109/ICMLA52953.2021.00090.
[102] S. F. A. Zaidi, H. Woo, and C.-G. Lee, "A graph convolution network-based bug triage system to learn heterogeneous graph representation of bug reports," *IEEE Access*, vol. 10, pp. 20677–20689, 2022, doi: 10.1109/ACCESS.2022.3153075.

RAZVAN BOCU received the B.S. degree in computer science, the B.S. degree in sociology, and the M.S. degree in computer science from the Transilvania University of Brasov, Brasov, Romania, in 2005, 2006, and 2007, respectively, and the Ph.D. degree from the National University of Ireland, Cork, in 2010.

He is currently with the Department of Mathematics and Computer Science, Transilvania University of Brasov, where he is also a Research and Teaching Staff Member. He is with Siemens Industry Software, Romania. He is the author or coauthor of more than 60 technical articles, together with six books and book chapters. In this capacity, he supervises research projects with strategic business value. He is an editorial reviewing board member of 28 technical journals in the field of information technology and biotechnology, which includes prestigious journals, such as *Journal of Network and Computer Applications*, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, *International Journal of Computers Communications and Control*.

ALEXANDRA BAICOIANU received the Ph.D. degree from Babeş-Bolyai University, Cluj-Napoca, in 2016.

She has been a Lecturer with the Transilvania University of Braşov, Braşov, Romania, since 2017, teaching various courses and seminars, where she is currently with the Department of Mathematics and Computer Science. She is also with Siemens Industry Software, Romania. She is a Research Engineer of informatics. She supervised tens of graduation and dissertations thesis, programming training courses, programming summer schools, and code/tech camps, some of them in collaboration with IT companies. She is a member of the Department's Machine Learning and Quantum Computing Research Group, founded in 2018. She has published more than 30 scientific articles and is the coauthor of six books. She was a part of various scientific projects, among them it is important to mention Advanced Technologies for Intelligent Urban Electric Vehicles, Powerful Advanced N-level Digital Architecture (PANDA), Intelligent Motion Control under Industry4.E (IMOCO4E), Artificial Intelligence and Earth Observation for Romania's agriculture (AI4AGRI), Digital Technologies and Artificial Intelligence (AI) Solutions projects (DiTArtIS), and New Modular Electrical Architecture and Digital platforM to Optimize Large Battery Systems on SHIPs (NEMOSHIP). Her research interests and expertise are in the field of machine learning, formal languages and compilers, algorithms, remote sensing and Earth observation data, autonomous driving, and electric and hybrid vehicles.



ARPAD KERESTELY is currently with Siemens Industry Software, Romania. He is also a Research and Development Engineer. He was a part of various scientific projects, among them it is important to mention Advanced Technologies for Intelligent Urban Electric Vehicles, Powerful Advanced N-level Digital Architecture (PANDA), Intelligent Motion Control under Industry4.E (IMOCO4E), Artificial Intelligence and Earth Observation for Romania's Agriculture (AI4AGRI), Digital Technologies and Artificial Intelligence (AI) Solutions projects (DiTArtIS), and New Modular Electrical Architecture and Digital platforM to Optimize Large Battery Systems on SHIPs (NEMOSHIP). His research interests and expertise are in the field of machine learning, formal languages and compilers, and algorithms.