Contents lists available at ScienceDirect

International Journal of Information Management Data Insights

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

An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review

Naresh Kumar Nagwani[a,*], Jasjit S. Suri[b]

[a] CSE Department, National Institute of Technology, Raipur, India
[b] Stroke Monitoring and Diagnostic Division, AtheroPoint™, Roseville, CA, United States

ARTICLE INFO

Keywords:
Software bug triaging
Bug assignment
Expert developer
AI
Recommender systems
Tossing graphs
Performance
and accuracy

ABSTRACT

The timely release of defect-free software and the optimization of development costs depend on efficient software bug triaging (SBT) techniques. SBT can also help in managing the vast information available in software bug repositories. Recently, Artificial Intelligence (AI)-based emerging technologies have been utilized excessively, however, it is not clear how it is shaping the design, development, and performance in the field of SBT. It is therefore important to write this well-planned, comprehensive, and timely needed AI-based SBT review, establishing clear findings. For selecting the key studies in SBT, Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) analysis was carried out, and 123 studies were selected for the AI-based review, addressing key research questions. Further, Cochrane protocol was applied for risk-of-bias computations for selecting AI techniques. We studied the six types of software bug triaging techniques (SBTT) that were analyzed. AI has provided the possibility of automating the time-consuming manual SBT process. Our study shows that AI-based architectures, developers for newly reported bugs can be identified more accurately and quickly. Deep learning (DL)-based approaches demonstrate capabilities for developing SBT systems having improved (i) learning rate, (ii) scalability, and (iii) performance as compared to conventional approaches. For evaluating the SBT techniques, apart from the accuracy, precision, and recall, the mean average precision (mAP) is suggested to be an effective metric. In the future, more work is expected in the direction of SBT considering additional information from developer's networks, other repositories, and modern AI technologies.

1. Introduction

The quality of the software depends on the working of the software features and thorough testing to find software bugs. If the bugs are detected, they are expected to be correctly fixed in a given amount of time and it can be done by assigning them to a suitable developer. The process of assigning newly reported bugs to the right fixer is known as software bug triaging (SBT) (Anvik et al., 2006). During the SBT process, the management of active developers is a complex task since it is labor-intensive, time-consuming, fault-prone (Anvik et al., 2006; Bhattacharya et al., 2012) and the process is cost-sensitive. As previously reported, "Average annual cost of software bug resolution is about $59.5 billion in 2002" (Bhattacharya et al., 2012), and the annual cost will be much higher in today's situation (Campos et al., 2019). Software bug management includes the task of bug prioritization (Uddin et al., 2017) also, where the priority of critical bugs is identified, it is also a key challenge in bug triaging. Thus SBT impacts the cost and reputation of any IT companies (Bhattacharya et al., 2012). If the assigned developer is not able to fix the bug, then it is re-assigned to the other developers. This activity is repeated till the bug is fixed encountering challenges. There are several reasons for bug re-assignments, a few of them are poor bug report quality, workload balancing, and difficulty to identify proper fix (P. J. Guo et al., 2011). A software development manager or triager has to deal with a lot of information available in software bug repositories. Managers or triagers utilize this information for identifying the most appropriate developer to fix the bugs identified in the software. So, automation of software bug triaging is the only option for better management of bug repository information. So, selection of an effective and *intelligent bug triaging approach* is the primary need of the information technology (IT) companies and thus is hypothesized as the key technology for minimizing the software bug fixing cost.

Software bug repositories consist of a wealth of bug information. Managing this high-volume information is a complicated task. Timely resolution of software bugs and managing the information of software bugs along with their status require a lot of effort. A software bug is not only providing the information of defects of software but along with it is also providing information about the status priority severity, and the information related to users involved in the particular module of software

* Corresponding author.
E-mail address: nknagwani.cs@nitrr.ac.in (N.K. Nagwani).

https://doi.org/10.1016/j.jjimei.2022.100153
Received 20 May 2022; Received in revised form 21 December 2022; Accepted 21 December 2022

related to the bug. For the large-scale software bug depositories consist of millions of software bugs, management of which requires specialized skills. The task of a software bug triager is to assign the bug to the appropriate software developer and get it fixed/resolved in a timely manner to make sure that the software release deadlines are maintained. The bug triager has to deal with a large set of information in bug repositories to identify the appropriate developers and to manage all the resolved and unresolved software bugs. Artificial intelligence can help in management of these large software bug depositories. AI techniques such as classification, topic modeling and NLP, clustering, and deep learning etc. are utilized effectively for helping software bug triaging process.

AI is a trending disruptive technology and has numerous applications including in process automation of businesses. Machine learning, deep learning, information retrieval, and natural language processing (NLP) are popular AI technologies for automating software bug triaging processes. The purpose of AI is to develop intelligence in machines (computers) to automate the business processes. SBT is an important process in software development. Project managers or triagers put a lot of effort into finding appropriate software developers to fix or resolve the reported bugs in the software. So, AI technologies can be utilized effectively for automation of software bug triaging processes. This work lists the major studies in the direction of software bug triaging using AI in a systematic and categorized manner to provide an insight on information management of software bugs including the bug assignment and resolution in large scale software bug repositories.

Recently AI has been utilized effectively in various domains particularly in process automation. SBT is an important process of software development. An effective automation of SBT can help the software/IT companies to maximize profit and reputation among the clients. In several recent studies the impact of using AI for businesses is presented by listing its advantages and challenges. AI and its ethical implication on digital technologies is studied by Ashok et al. 92022). They have mapped fourteen ethical considerations to seven digital type archetypes. In the study, a critical debate on ethical use of AI in digital technologies is presented. A review on AI in information system research is presented by Collins et al. (2021), where the authors have listed the contribution of AI, its practical implication and identifying new opportunities in information system. It was highlighted that lack of consensus exists around the clear definition of AI particularly when applied to information systems. It was concluded in the study that machine learning is an overwhelmingly prominent technology in AI and the most prominent reputed business that utilizes AI is process automation. Also, deployment of AI technologies like machine vision, NLP, robotics, and expert systems are dominating technologies. A study on how and why data science helps firms to achieve desired outcomes is presented by Boehmke et al. (2020). An illustration is also presented for the use of applied data science to a strategic sourcing problem via the use of open-source technology. A study on sustainable development using AI for various industries like construction, transportation, healthcare etc. is presented by Kar et al. (2022). It was identified that regression, reinforcement learning, and decision support system-based AI technologies are very popular for sustainability. The methods, challenges and issues associated with adopting AI for sustainable development are also discussed. A framework based on inductive-learning related to success & barrier shared on social media platform is presented by Kar & Kushwaha (2021) various factors like efficiency, innovation, business research, product novelty, manual intervention, adaptability, emotion support, experiential learning, fear of failure are identified and considered in the framework. The framework bridges the gap of adopting AI in businesses.

There are various challenges in adoption of AI technologies in any organization. Several recent studies have discussed the practical problems in adoption of AI technologies. A model for AI adoption in organizations is proposed by Uren and Edwards (Uren & Edwards, 2023), they have identified the parameters People, Processes, Technology lens, incorporating Data. The proposed model suggested that apart from technological readiness, people, process, and data readiness are also required for effective adoption of Borges et al. (2021) have discussed issues like practical use and lack of knowledge in effective utilization of AI. A conceptual framework is presented by the authors considering the four major parameters (i) decision support; (ii) customer and employee engagement; (iii) automation; and (iv) new products and services. Although these challenges exist, studies have proposed some strategies and solutions also. Considering the suitable strategy AI can be adopted for any organization for smooth implementation and automation of the process.

Applying AI in any business domain is not a straightforward job. There are various factors that affect the proper utilization of applying AI for business automation and applying AI for SBT is not an exception. There are many studies available in applying AI in various fields which have discussed various challenges and strategies for proper adoption of AI. Some of these studies are applying AI for human resource management (Votto et al., 2021), incorporating ontologies (Antunes et al., 2022), application of big data in emerging management disciplines (Kushwaha et al., 2021), transfer learning for recognition of human activities (Ray et al., 2023), deep learning in business analytics (Schmitt, 2023), applying reinforcement and deep learning for financial industries (Singh et al., 2022), AI for digital healthcare (Mahdi et al., 2023), AI for smart cities (Herath & Mittal, 2022), AI for detection of fake news (Nasir et al., 2021), and applications of text mining in services management (Kumar et al., 2021).

Managing the software development project is a crucial and time bound activity. It requires monitoring of various repositories, people management, and a lot of information management. The automation of software bug triaging helps the managers and triagers in identifying the most appropriate developers for fixing and resolving the bugs in a faster way. Automation of software bug triaging is performed by analyzing the information available on the software repositories and reduces the complexity of information management also. There are a few studies available on information management of software development. Some of the highlighted studies are summarized here.

Software maintenance is a complex and important topic in information management. Ketler and Turban (1992) have identified three classes of factors that impact software maintenance efforts. These factors are technological, administrative, and behavioral. Open-source software development is based on the approach of knowledge contribution, the knowledge lost once contributors leave a project. Rashid et al. (2019) have analyzed the factors related to knowledge loss in open-source software projects and identified that there is a lack of effective strategy for reducing the knowledge loss in these projects. They have also suggested key points to mitigate the knowledge loss. Deng and Robinson (2021) have analyzed the open-source development data for understanding the effects of how routines (functions or methods) changes impact the popularity of open-source software. Authors have identified the relationship between routine change, routine diversity, and project performance for open-source software development. They have identified that routine changes reduce the project's popularity. Akgün (2020) presented a study to demonstrate the impact of team wisdom in software project performance. The key managerial implications for different software project development projects are also discussed in the study. Automation of software development processes is the key objective of continuous software engineering. A case study on adoption of knowledge management by software companies for automating the development processes is presented by Colomo-Palacios et al. (2018). It is shown that knowledge management is the important enabler in the success of continuous software engineering.

Even though SBT is a key problem in the research area of mining software repositories, there is significant technological evolution in this field, and still, some open challenges exist in this area. Today there exist numerous automated techniques for SBT in the research area of mining software repositories (Kagdi et al., 2007). In the last decade, Artificial Intelligence (AI)-based techniques have been used for automating the SBT process. The software bug repositories are maintaining all the required information about the software bugs for resolving them.

To develop SBT techniques, the first step is to extract data from the bug repositories which are the primary source of software bugs. This data is pre-processed for making the bug data ready for triaging, and suitable AI architecture is applied for automating the triaging process (Bhattacharya et al., 2012; Koc et al., 2019).

Earlier bug triaging was done manually but as AI techniques emerged, it has played a vital role in automating bug triaging. It provides the advantages of machine learning (ML), information retrieval (IR), deep learning (DL), and other relevant technologies to semi-automate or automate the process of SBT more accurately thereby saving a lot of manual effort in software development. Since AI technologies are utilized for SBT (Lee & Seo, 2020; Mani et al., 2019; Uddin et al., 2017), an understanding of the design, development, and performance of these technologies is a challenge. So, an organized technology-wise review of existing SBT techniques is the need of the day.

The review presented in this paper explores the methodology and techniques of SBT. For SBT process automation scientists have proposed various solutions considering AI. The objective of the presented study is to identify the answers of the following three research questions:

RQ1: can AI improve and contribute effectively to automating the SBT process?

RQ2: Identify the major performance parameters used in the existing studies?

RQ3: List the challenges and issues and identify possible future work for further improving the SBT process?

To answer the above three research questions, a three-dimensional analysis is performed. The dimensions are trending research technologies in AI, performance evaluation techniques and, the future directions for SBT. This study provides a comprehensive narrative review of the important SBT techniques focusing on the quality publications and quality contributions made in the field of SBT. For proving the hypothesis, categorization of SBT techniques is implemented, and meta-analyses are carried out for each identified category. Performance evaluation parameters are extracted from each included study and summarized for analysis. The challenges and future scope of the existing studies are explored, and a categorized summary is presented. After selecting the appropriate studies as per the methodology, the studies are grouped into six major categories. These can be labeled as ML-based techniques (Bhattacharya et al., 2012; Bhattacharya & Neamtiu, 2010a; Goyal et al., 2015), IR-based techniques (Ahsan et al., 2009; Zhang et al., 2017), crowdsourcing and social network analysis (SNA)-based models (Huang & Ma, 2019; Mao et al., 2015), recommender systems (RS) (Chen et al., 2019a, Chen et al., 2019b; Florea et al., 2017b; Xie et al., 2012), mathematical modeling, and optimization (MMO)-based techniques (Goyal & Sardana, 2017c; Rahman et al., 2009; Tamrawi et al., 2011a), and DL-based techniques (Guo et al., 2020; Kukkar et al., 2019; Mani et al., 2019). All these six categories are a subset of AI techniques.

The layout of this study first presents the search strategy in section 2. Bug triaging pipeline is discussed in section 3. The heart of the study is focused on the AI-based analysis for SBT in section 4. This section also presents a brief review of all these SBT categories discussing their pros and cons. A suggestive matrix (Appendix-F) is also proposed to select a particular AI architecture under different constraints. The search strategy for selecting the relevant studies for the review is discussed in the next section. Section 5 presents the performance evaluation. Section 6 presents the discussions, and the study concludes in section 7.

2. Search strategy

For preparing the review of the SBT process, proper literature search queries and search keywords are identified. Searching is performed primarily on Scopus and Web-of-Science. Both bibliography databases cover a good collection of articles in various disciplines. Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) (Moher et al., 2009) analysis was followed for the selection of appropriate articles for this review. The PRISMA is a very standard protocol adapted study selection for the reviews and is used in several recent reviews in the field of AI covering different applications (Jena et al., 2021; Suri et al., 2021). Search keywords such as bugs triaging, triager, assignment, expert developer, triaging using AI, machine learning, deep learning, and their combinations were used for search queries in the online bibliography databases. The combinations of the keywords were used for searching in the title, abstract and keywords search topics. Searching was performed in multiple levels to finalize the best selection of the studies.

2.1. PRISMA analysis and search keywords

To identify the relevant research and to better explore the review work few popular Scopus and Web-of-Science indexed conference papers (Appendix-B Table B2) with higher citations in Google Scholar and Scopus are also considered. Journal papers (Appendix-B Table B1) in the field of software engineering with good impact factors and proper coverage of the bug triaging subject are considered. The process and summary of literature searching are graphically summarized in Fig. 1, from the initial findings of 375 studies, 123 studies are found to be relevant, recent, and eligible to be considered in the bug triaging review work. The combination of words, bug triaging, bug assignment, expert developer, software bug, issue management, change request management, and bug repository are used as query string for finding relevant studies. The combination of these keywords in searching the relevant papers is shown in Fig. 2. Several attributes like the publication year, citations, journal impact factor, the domain of the journal, hybrid approaches for triaging, optimization-based studies for triaging, performance improvement for bug assignment, and triaging as well as specific studies related to bug triaging and bug assignments are included.

To avoid any bias in the selected studies of SBT in the presented work, standard protocols are followed. The review is conducted as per the guidelines of the Cochrane given by Higgins et al. (2011) along with the PRISMA (Moher et al., 2009) analysis.

Three inclusion criteria for selecting the studies in SBT are considered. Research articles in journals indexed in Scopus and Web-of-Science, domain-specific international conference papers listed in Scopus with high citations, and latest international conferences i.e., published on or after 2018 are included irrespective of their citations as the inclusion criteria. Similarly, exclusion criteria are also created for the elimination of studies not considered for review. Papers other than the English language are not considered, Scopus indexed international conference papers with lesser citations published before 2017 are excluded from the studies, general work carried in mining software repositories not focused on SBT is not selected. Also, techniques of prediction of bugs and other matrices not focusing on SBT are not considered in this review.

2.2. Risk-of-bias assessment

Risk-of-bias (RoB) assessment is also carried out for the selected studies for narrative review. The Cochrane protocol analysis for assessment of the RoB is summarized in Table 1.

The RoB tool includes six domains of bias: selection bias, performance bias, detection bias, attrition bias, reporting bias, and other bias (Higgins et al., 2011). *Selection bias* refers to the bias that arises due to random selection and filtration of the articles for the study. It is eliminated by categorizing the bug triaging techniques into multiple categories so that participants of the selected studies can cover all the dimensions of the bug triaging techniques reported earlier. Other reasons for selection bias are allocation concealment. *Performance bias* is derived from the blinding of participating articles. Detection bias is due to the blinding of the outcome assessments. *Attrition bias* is due to the incomplete data outcome. *Reporting bias* arises due to reporting selectively findings from a group of articles. The *Other bias* is the bias not covered in the previous five types.

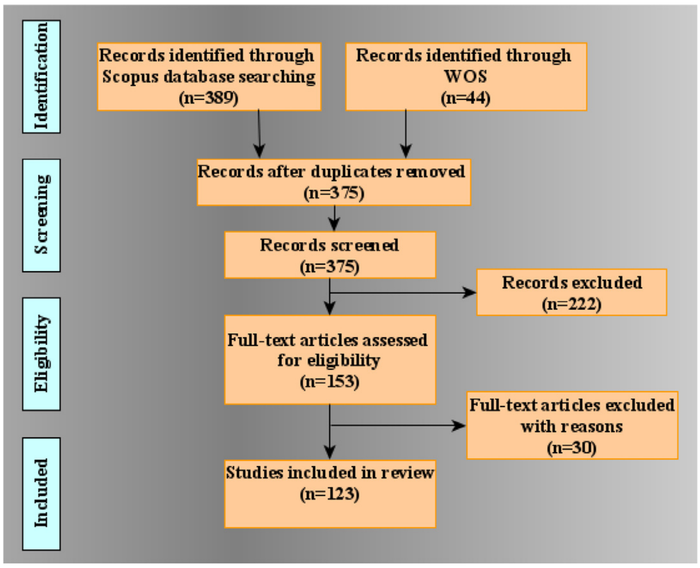
Identification
Screening
Eligibility
Included
Records identified through Scopus database searching (n=389)
Records identified through WOS (n=44)
Records after duplicates removed (n=375)
Records screened (n=375)
Records excluded (n=222)
Full-text articles assessed for eligibility (n=153)
Full-text articles excluded with reasons (n=30)
Studies included in review (n=123)

Fig. 1. PRISMA analysis for selection of studies.

Table 1
Cochrane protocol analysis for assessing risk-of-bias.

Type of Risk-of-Bias	Authors Judgement	Justification for Judgement
Selection Bias	Low	Eliminated by categorizing the bug triaging techniques into multiple categories so that participations of the selected studies should cover all the dimensions of the bug triaging techniques reported earlier
Performance Bias	High	All performance parameters related to bug triaging are listed and the study is already covered as a research question
Detection Bias	High	All performance parameters related to bug triaging are listed and the study is already covered as a research question
Attrition Bias	Low	Extraction of all major information from selected studies is performed based on the research objectives.
Reporting Bias	Low	All the selected studies are examined, and a comprehensive analysis of observations is done.
Other Bias	Low	All due care is taken to include the studies as per selection criteria and information focused on the research objectives is listed.

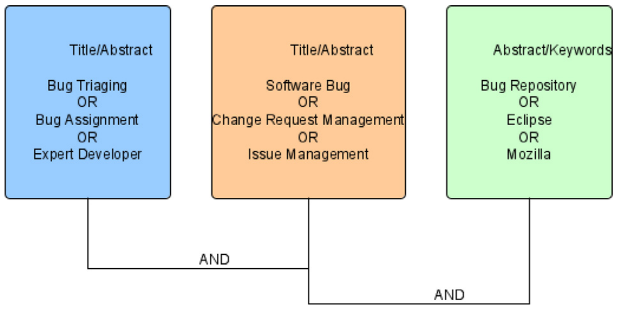

Fig. 2. Combination of search keywords.

2.3. Study proportions

From the selected 123 studies, the study proportionate as per the different AI-based technologies in this study is presented in Fig. 3. It is observed that ML-based and IR-based are utilized maximum for developing SBT techniques. The technological evolvement of SBT techniques is graphically summarized in Fig. 4.

From the technology evolvement point of view, it is seen that earlier ML and IR-based techniques were most commonly used in the direction of SBT, however, currently, advanced AI technologies like DL (Mani et al., 2019), optimization techniques (Goyal & Sardana, 2017c), and recommender systems (Kaur & Jindal, 2019; Mohsin & Shi, 2020) are emerging, and researchers have started utilizing these technologies for further improving the SBTT.

3. Bug triaging pipeline

Using AI-based intelligent bug triaging techniques the task of assigning the appropriate developers for newly reported bugs can be automated. The purpose of any bug triaging algorithm is to identify the *right fixers* for the bugs. A bug triaging algorithm takes the bug and developer information as input and generates a list of appropriate developers as the output. The generated output can be evaluated using various performance parameters.

The generalized process of software bug triaging is presented in Fig. 5. The process is summarized as follows. In the first step, new software bugs are identified during the testing phase of software development and reported by the testers or quality engineers through bug tracking tools. The bug triager extracts the relevant information like the nature of the bug, the domain of the bug, and the developer's information who can be tentative fixers from the bug tracking tools. After getting all this information and by using some suitable techniques, triager assigns the newly reported bugs to the right fixers (Bhattacharya et al., 2012). Sometimes triager creates a ranked list of developers for the newly reported bugs (Kagdi et al., 2007).

A detailed workflow diagram of the bug triaging process consisting of key steps of bug triaging using AI-based techniques is shown in Fig. 6. Pre-processing is the first step in which data is made ready for triaging. For textual data, stopping, stemming and lemmatization are performed, and noise is handled using several standard data mining operations. The choice of pre-processing technique depends on the type of data, noise handling mechanism, and type of data representation techniques.

Class imbalance is a common problem in any AI architecture, in which during the training phase some classes can have higher samples and others can have only a few. It can lead to biasing in training and lead to poor performance. Class imbalance problems in SBT can occur due to reasons like, only a few developers have fixed most of the bugs and other developers have fixed a few bugs only, so any SBT approach can always assign the developers who have fixed more bugs. Generally, the class imbalance can be handled with techniques like under-sampling (lowering the instances of the categories having majority) or oversampling (increasing the instances of the categories having fewer values).

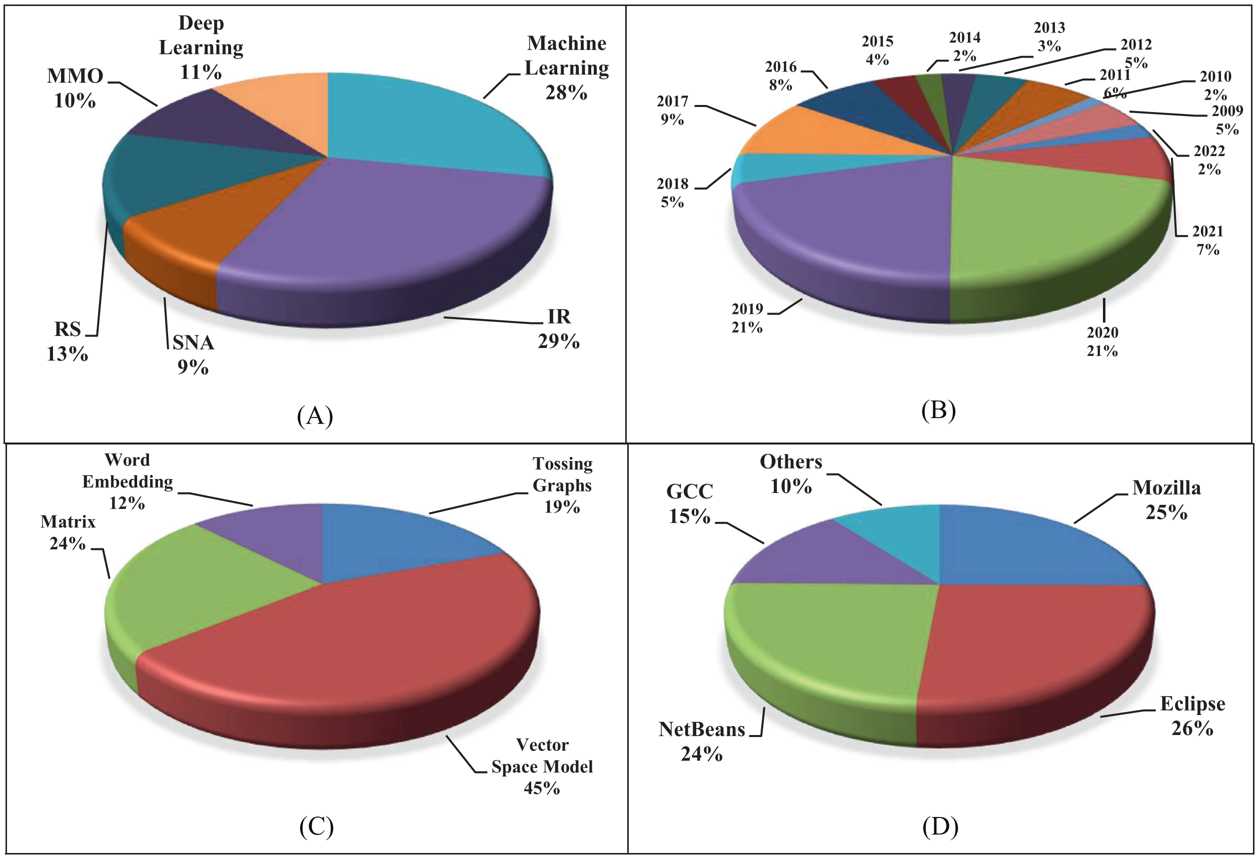

Fig. 3. Distribution of bug triaging techniques covered in the study (A) Technology-wise (B) Year-wise (C) Representation wise and (D) Repository wise.

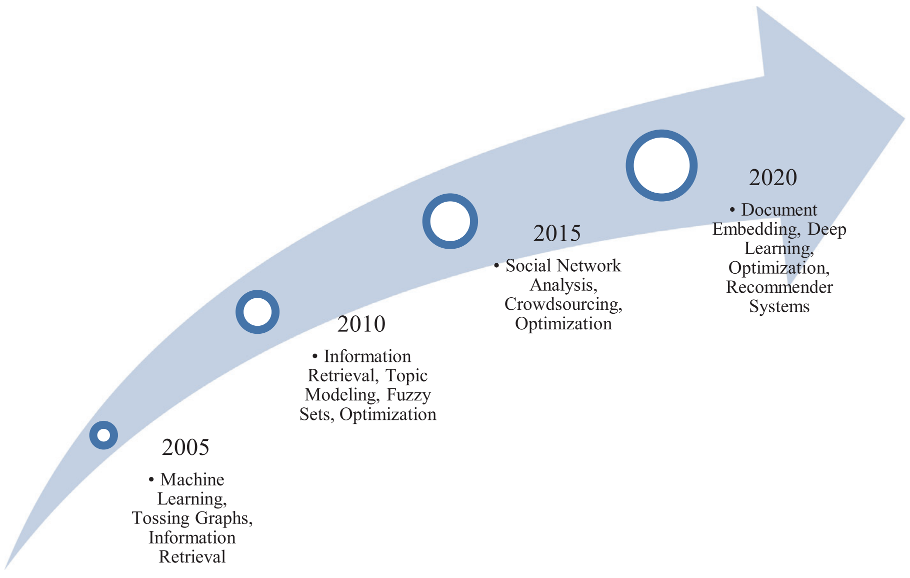
2005
• Machine Learning, Tossing Graphs, Information Retrieval
2010
• Information Retrieval, Topic Modeling, Fuzzy Sets, Optimization
2015
• Social Network Analysis, Crowdsourcing, Optimization
2020
• Document Embedding, Deep Learning, Optimization, Recommender Systems

Fig. 4. Technology evolvement in software bug triaging.

Although there is no dedicated work on class imbalance pertaining to SBT, it is applied in other mining software repositories such as defect prediction. The class imbalance for defect prediction using feature selection (Xia et al., 2015a), oversampling (Feng et al., 2021), and KNN under-sampling (Goyal, 2022) is reported in earlier studies. A theoretical framework is also presented recently, in which class-balance loss functions are proposed for handling class imbalance problems (Cui et al., 2019). Three loss functions namely, SoftMax cross-entropy, sigmoid cross-entropy, and focal loss are proposed for deriving the weighting factors for the classes to handle the class imbalance problem.

The bug representation technique is the important step after pre-processing, the bug information is transformed into a suitable representable format for taking it as input to the triaging algorithms. Most algorithms use bug information representation as vector (Anvik, 2006; Xuan et al., 2017a), matrix (Ahsan et al., 2009), tossing graphs (Bhattacharya et al., 2012; Jeong et al., 2009; Bhattacharya & Neamtiu, 2010b; Chen et al., 2010; Chen et al., 2011) and network (relationship matrix) representation (Zhang et al., 2016a) techniques, and these techniques are explained in section-4. The AI-based algorithm is then selected on the transformed bug report information and output

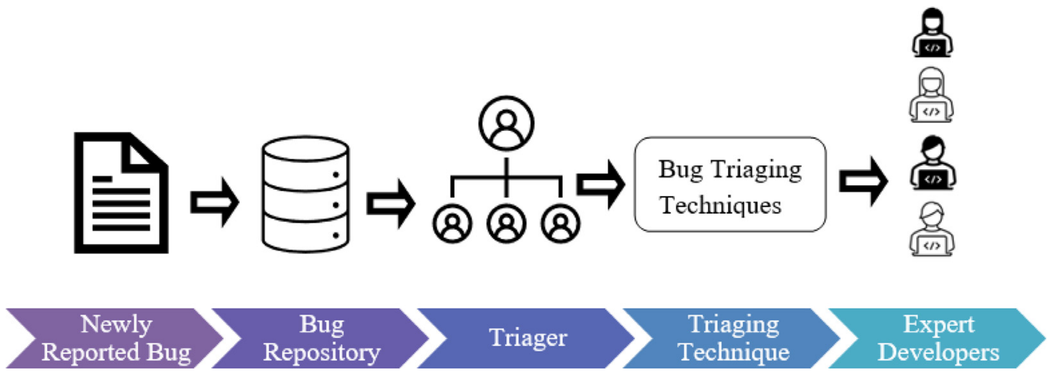

Fig. 5. General software bug triaging process.

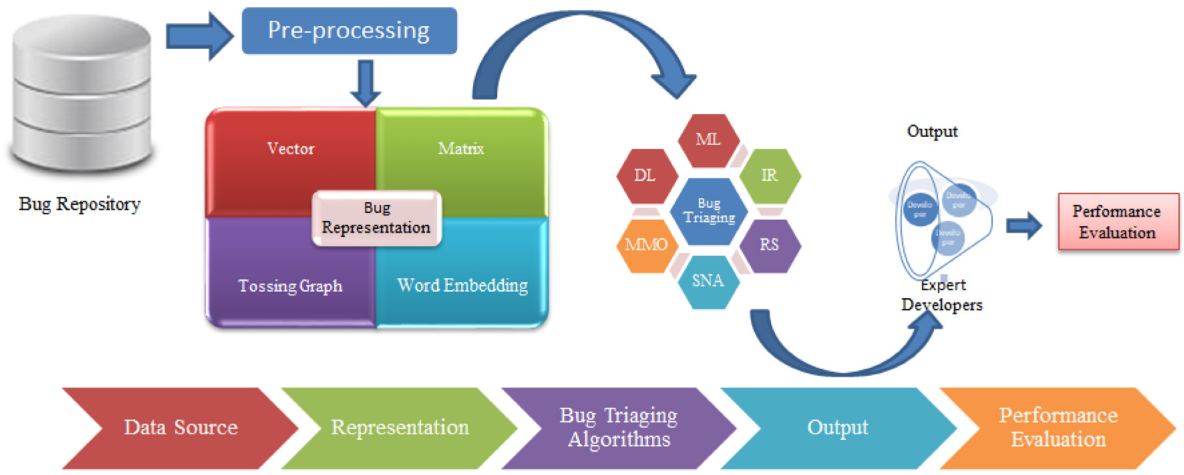

Fig. 6. Workflow diagram of software bug triaging process.

from the algorithms is generated as the list of expert developers for the newly reported bugs. Finally, performance evaluation is performed for the generated output, where the performance metrics are calculated to measure the efficacy of the selected algorithm. Standard performance metrics for evaluating bug triaging are covered in section-4.1.

3.1. Data collection and popular repositories

During the software development, the bug data is collected and maintained in bug tracking systems or issue tracking systems. There are many popular bug tracking systems available today, which are utilized widely during software development. The quality engineers or software testers identify defects in the software and report them as bugs in software. The project managers or triagers have full access to these tracking systems and from there they can collect the bug data. For open-source software the bug data is available publicly and utilized widely in most of the research works related to software bug triaging.

As per the summary of literature presented it is observed that most of the research work was carried using open-source bug repositories like Eclipse (Anvik, 2006; Zaidi et al., 2020; Anvik & Murphy, 2011; Kanwal & Maqbool, 2012; Tamrawi et al., 2011b, 2011a), Mozilla Firefox (Mani et al., 2019; Guo et al., 2020; Ye et al., 2020), Apache (Zhang et al., 2016b), and Gentoo (Xi et al., 2018a; Xi et al., 2018b; Xi et al., 2019), etc. Along with these few other repositories, also available was data from multiple repositories like code commit repositories that were used for cross repository analysis-based bug triaging techniques. A popular such repository is Github (da Silva et al., 2020; Sajedi-Badashian & Stroulia, 2020b; Matsoukas et al., 2020), which is used in several previous studies.

Some of the popular bug repositories with their URL's are listed below:

(i) Mozilla bug repository - https://bugzilla.mozilla.org/home
(ii) GitHub bug repository - https://github.com/orgs/github/repositories
(iii) Eclipse bug repository - https://bugs.eclipse.org/bugs/
(iv) NetBeans bug repository - https://issues.apache.org/jira/projects/NETBEANS
(v) GCC bug repository – https://gcc.gnu.org/bugzilla/
(vi) OpenOffice bug repository - https://wiki.openoffice.org/wiki/Issue_Tracker

3.2. Bug data representation

Various AI architectures for bug triaging are discussed in section 4. Each architecture has its own technology for handling the newly reported bugs and identification of suitable developers for fixing them. Bug data representation and performance evaluation of the bug reports is an important aspect of all AI-based architectures. Bug reports in the bug tracking tool contain various attributes of a software bug. The two primary key attributes of a software bug are summary (sometimes termed as title) and description, and both attributes are textual. Some of the key attributes and their corresponding meaning are presented in Table 2.

Bug report representation techniques are broadly categorized into four categories namely, tossing graphs (TG), vector, matrix, and document embedding. Vector and Matrix representation techniques were the benchmark representation techniques from Text Mining and utilized in the early studies of SBT since the early 2000s. Bug TG was introduced

Table 2
Software bug attributes and their meaning.

Attribute	Meaning
Bug-id	Unique integer number representing the bug ID
Title	One/two liner summary of the reported software bug
Description	Detailed information about the reported bug
Priority	The priority of the bug
Severity	The severity of the bug
Component	Software component
OS	The operating system on which bug is reported
State	The current state of the software bug
Opened/Submitted	Date of the first-time submission
Modified/Updated	Last updated date of the software bug
Assigned-to	Presently assigned to the developer
Reported-by	Software bug identified and reported by the person

Table 3
Characteristics of different BT techniques.

Bug Triaging Type	Characteristics
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

in 2009 (Jeong et al., 2009), and word embedding-based models are utilized in the recent decade of deep learning.

The most commonly used representations in the existing studies are vector representation (Anvik, 2006; Anvik & Murphy, 2011), and matrix representation (Zhang et al., 2016a; Zhang et al., 2016b; Hu et al., 2014). In both vector and matrix representation techniques, first, the unique terms are identified after the pre-processing of the bugs. The major pre-processing tasks include the elimination of the stop words, stemming, and lemmatization. The identified unique terms are then listed in order and then bug information is converted into the vector order or matrix format as per the occurrence of the terms in the software bugs. Vectors are created using the term frequency or different measures like TF-IDF information of the bugs. Just like the vector representation, in matrix representation terms are represented as a column, and rows represent the Bugs in the case of the "Bug-Terms" matrix and represent Developers in the case of the "Developers-Terms" matrix. In the Developer-Term matrix, rows represent the developers and columns represent the frequencies or other measures such as TF-IDF in the column for all the unique terms as shown in Eq. (1).

$$\begin{array}{cc} & \begin{array}{ccc} t_1 & \cdots & t_N \end{array} \\ \begin{array}{c} D_1 \\ \vdots \\ D_M \end{array} & \begin{bmatrix} d_{11} & \cdots & d_{1N} \\ \vdots & \ddots & \vdots \\ d_{M1} & \cdots & d_{MN} \end{bmatrix} \end{array} \tag{1}$$

Bug TG is a popular data structure proposed by Jeong et al. (2009) which was used in many studies in the field of software bug triaging. Tossing graphs are further modified by Bhattacharya et al. (2012), the modified tossing graphs are built using the additional information of bug priority and component information. A tossing path is the sequence of developers through which the bug is passed until it is fixed at the end. If the length of the tossing path is zero, then it is termed as zero tossing and it indicates that the bug is fixed by the same developer to whom it was assigned. TG representation is used in many ML-based architectures to DL-based (Xi et al., 2018a; Xi et al., 2018b; Huang & Ma, 2019) architectures. The maximum accuracy obtained from TG representation is 86.09% using ML-based architecture proposed by Bhattacharya et al. (2012). The advantage of TG-based models is that they capture the essential information for bug triaging as the TG is creating bug re-assignment information from the bug repositories. The key disadvantage of TG representation is the creation of a bug TG which is a complex procedure, where the bug reports are required to be encoded in tossing sequences (Xi et al., 2019) and graphs (Bhattacharya & Neamtiu, 2010a).

Apart from these, the representation technique also depends on the type of AI architecture used for bug triaging, for example, if social network analysis techniques are used then relationship matrices (Hu et al., 2014) (derived from relationship graphs) are adapted. For DL-based architectures, document embedding (Florea et al., 2017a; Lee et al., 2017; Xi et al., 2018a; Xi et al., 2018b) such as Word2Vec (S.-R. Lee et al., 2017; Florea et al., 2017a), Glove (Xi et al., 2018a; Xi et al., 2018b), and other compatible representation techniques are used (Zaidi et al., 2020).

4. AI architectures for bug triaging

There exist several techniques of software bug triaging using different technologies. SBT techniques are classified into six major categories based on the AI technologies used. The categorized summary will help in understanding the implementation perspective of SBT techniques along with the pros and cons of each AI technology. The six major technological categories identified are ML-based, IR-based, SNA-based, RS-based, MMO-based, and DL-based SBT techniques. The characteristics of these techniques are summarized in Table 3. The summary of works done in each category is presented in this section along with discussing the pros and cons of each technique.

4.1. ML-based software bug triaging models

Machine Learning (ML) techniques were the first choice at the beginning of automating the software BT technique. Classification technique (Jiechieu & Tsopze, 2020; Kukkar et al., 2019; Mohsin & Shi, 2020) of machine learning is most frequently used for SBT, apart from clustering (Florea et al., 2017b; J. Lee et al., 2019), and association rule (Sharma et al., 2017) mining. SBT is treated as the multi-class, single-label classification problem (Murphy & Cubranic, 2004), where a software developer is presented as a class. The summary of key ML-based bug triaging work is presented in Table 4, while ML-based studies are presented in Appendix C (Table C1). From the table, it is visible that the classification technique is used widely under machine learning-based bug triaging techniques, and performance parameters of classification tasks such as accuracy, precision, recall, and F1-measure are used for evaluating the proposed techniques. The accuracy obtained using machine learning techniques is from 44.4% to 86.09% and the maximum precision obtained is 86%. A comparative survey on ML and information retrieval techniques for bug triaging is presented by Goyal and Sardana (2017a), where they have concluded that IR techniques are best as compared to ML techniques. The advantage of the ML-based SBT is its simplicity to model, there is good Application Programming Interface (API) support. Although ML is a commonly used technology, achieving a higher performance (Ahsan et al., 2009; Wu et al., 2011; Xuan et al., 2017a) is still a concern that motivates the researchers to improve the existing technologies and explore modern AI-based techniques. The performance of ML-based techniques is less as compared to DL-based techniques but is comparable with IR-based techniques.

4.2. IR-based software bug triaging techniques

The process of SBT can also be seen as the problem of IR in which the relevant information is such that the appropriate software developer is retrieved from the set of software developers for the newly reported bugs. A summary of key research work carried out in SBT using IR is presented in Table 5, and further IR-based studies are presented in Appendix C (Table C2). IR-based techniques such as LSA (Latent Semantic Analysis) and LSI (Latent Semantic Indexing) (Kagdi et al.,

Table 4
ML-based software triaging techniques.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Bhattacharya et al., 2012)	2012	Naïve Bayes, Bayesian Networks, C4.5, and SVM	JSS[a]	Tossing Graph	Mozilla and Eclipse	Accuracy of 86.09%
(Xuan et al., 2014)	2014	Instance and Feature Selection	IEEE TKDE[b]	Vector Space	Eclipse, Mozilla	Accuracy of 81.3%
(Xia et al., 2016)	2016	Multi-label learning	IEEE Transactions on Reliability	Vector Space	OpenOffice, NetBeans, Eclipse, and Mozilla	Achieve an average F-measure score of 0.56–0.62. Precision 0.6113, recall 0.6406
(Yin et al., 2018)	2018	Feature selection and an ensemble extreme learning machine	IEEE Access	TF-IDF	Bugzilla, Eclipse, GCC, and NetBeans	The maximum achieved accuracy is 71.2%
(Xi et al., 2019)	2019	Classification	Journal of computer science and technology	Vector Space	Eclipse, Mozilla, Gentoo	Accuracy@10 is 0.799

[a] Journal of system and software.
[b] IEEE transactions on knowledge and data engineering.

Table 5
IR-based software bug triaging techniques.

Author(s)	**Year**	**Technique**	**Published**	**Representation**	**Dataset**	**Performance**
(Sajedi-Badashian & Stroulia, 2020b)	2019	Developer expertise profile	Software - Practice and Experience	TF-IDF	GitHub	mAP ranges between 50% and 71%
(Alkhazi et al., 2020)	2020	activity-based features for developer profile	Applied Soft Computing Journal	TF-IDF	BIRT, Eclipse UI, JWT, SWT	Accuracy@10 is about 95%
(Xia et al., 2017)	2017	multi-feature topic model (MTM)	IEEE Transactions on Software Engineering	Vector	GCC, OpenOffice, Mozilla, NetBeans, Eclipse	Accuracy of Top-1 and Top-5 in the range of 0.4831–0.6868, and 0.7686–0.9084
(Alazzam et al., 2020)	2020	term frequency, term correlation, and topic modeling	IEEE Transactions on Computational Social Systems	Vector	Eclipse	Accuracy 94.6%, Precision 0.925 and Recall 0.941
(Lee & Seo, 2020)	2020	Multiple LDA	Human-centric Computing and Information Sciences	Vector	Android, Mozilla	Accuracy of 98.31 is achieved

2012; Linares-Vasquez et al., 2012) are also utilized along with the other techniques. The accuracies obtained using IR-based SBT techniques are in the range of 63.2% to 96%. A recall of 95% is claimed by the work of Banerjee et al. (2017). TF-IDF (Term Frequency – Inverse Document Frequency) is the most common presentation in IR-based techniques for SBT. Few studies have proposed some modifications in TF-IDF-based approaches and some variations are proposed like Shokripour et al. (2013) suggested bug location information and term weighting in TF-IDF, Shokripour et al. (2014) presented term weighting technique and the same author (Shokripour et al., 2015) presented considering time metadata in TF-IDF presentation. SBT using both Text Mining (Alenezi et al., 2013; Alonso-Abad et al., 2019; Ardimento et al., 2020; Hindle et al., 2016; Kaur & Jindal, 2019) and Text Similarity Techniques (Chen et al., 2010, 2011; Chen et al., 2019a, Chen et al., 2019b; Hu et al., 2018; Jiang et al., 2019; Resketi et al., 2020) is explored in many earlier studies of SBT. Bug triaging technique for large repositories is presented by Banerjee et al. (2017). Topic modeling is also used in many key studies (Park et al., 2011; Lee et al., 2020; Roopa et al., 2019; Xia et al., 2017; Xie et al., 2012; Zhang et al., 2016a) in software bug triaging. Latent Dirichlet Allocation (LDA) is a popular probability-based topic modeling technique given by Blei et al. (2003) is used in many studies of SBT (Park et al., 2011; Xie et al., 2012). Few studies have presented some specialized variants of topic modeling for bug triaging like the multi-feature topic model (MTM) (Xia et al., 2017), Entropy Optimized Latent Dirichlet Allocation (Zhang et al., 2017), and Multiple LDA (Lee & Seo, 2020). The maximum accuracy obtained using the topic modeling-based triaging technique is 98.31% from the work presented by Lee and Seo (2020). Just like ML, IR has good API support, and it is also simple to model. The key challenges with IR or topic modeling-based techniques are performance improvement, generation of meaningful and relevant terms in the topics generated from topic modeling techniques. Topics generated by topic modeling techniques may have some degree of randomness and may impact the analysis (Brookes & McEnery, 2019). But again, the key challenge is performance (Matter et al., 2009; Baysal et al., 2009; Xie et al., 2012), and handling of real time SBT.

There is a lot of similarity between ML-based techniques and IR-based techniques. The primary difference between the both is, IR-based techniques consider and focus mainly on textual data available in the software bug repositories. Just like ML-based techniques, modeling is easy for IR-based techniques with the least amount of computing infrastructure requirement for implementing SBT. It also has good support for available programming languages and APIs. The drawbacks are also common with ML-based techniques like scalability and real-time SBT.

4.3. SNA-based architecture for software bug triaging

The social network in context to software bug triaging refers to the developer's network where the software developers exchange information related to the development of software. Fixing bugs requires specialized skill sets and sometimes developers seek information from various online sources like StackOverflow,[1] or GitHub[2] in terms of discussion forums and technical question answering techniques. Extracting useful information from various sources for improving bug fixing tasks is called Crowdsourcing and it is explored in a few studies (Alazzam et al., 2020; Xuan et al., 2017b ;Zhang et al., 2017) for BT. The general architecture of SNA-based SBT is presented in Fig. 7.

The information from multiple repositories can be integrated along with the software bug repository information for the identification of ex-

[1] https://stackoverflow.com/
[2] https://github.com/

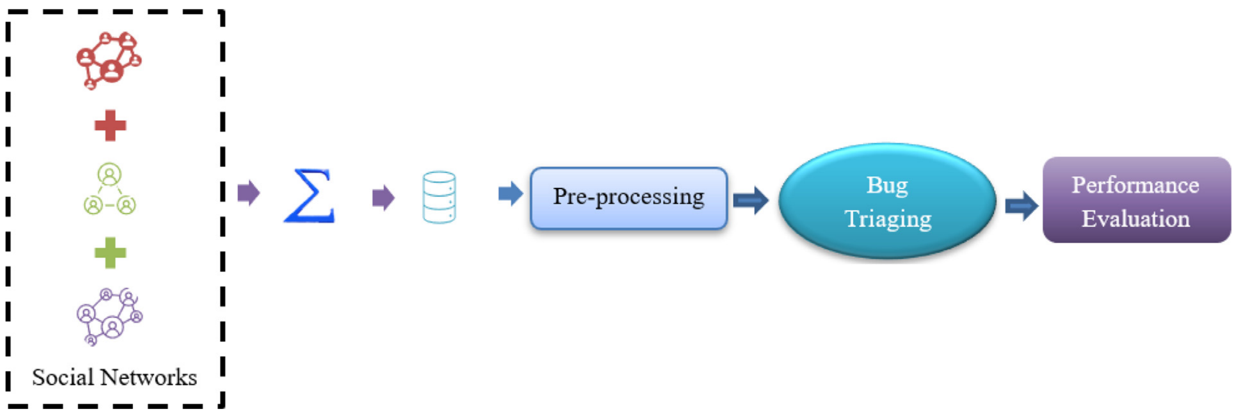

Fig. 7. Block diagram of SNA based SBT architecture.

Table 6
SNA-based models for SBT.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(W. Zhang et al., 2016a)	2016	LDA and Heterogenous Network	Chinese Journal of Electronics	Vector Space and Matrix	Eclipse JDT	Recall at Top 6 is 60.29% and the minimum of 54.76%
(W. Zhang et al., 2016b)	2016	KNN and Network	Information and Software Technology	Vector Space and Matrix	Mozilla, Eclipse, Apache Ant, and Apache Tomcat	Recall@10 0.86 and Precision@10 is about 0.6
(Sajedi-Badashian & Stroulia, 2020b)	2019	Developer expertise profile	Software - Practice and Experience	TF-IDF	GitHub	mAP ranges between 50% and 71%
(Huang & Ma, 2019)	2019	DL	Conference	Word2Vec	Eclipse	Link prediction accuracy Eclipse 0.731, Gnome 0.656, Node classification Eclipse 0.823 Gnome 0.625
(Matsoukas et al., 2020)	2020	SVM One-Vs-Rest	Conference	TF-IDF	GitHub	Accuracy in the range of 0.8 and 0.7 is achieved

pert developers. This concept is also known as cross repository analysis (Zhang et al., 2019). SBT using social network analysis and crowdsourcing is summarized in Table 6, further SNA-based studies are presented in Appendix C (Table C3). The advantage of crowd-sourcing and social network-based techniques utilizes the relationship between the developers and their knowledge for efficiently fixing the newly reported bugs more accurately. The key challenge with these techniques is, data aggregation and integration is a complex task as data is derived from multiple sources (Alazzam et al., 2020; Xuan et al., 2017b; Zhang et al., 2017) and relationship graphs are generated. SNA-based techniques are complex to model as it involves graph data structure for developer-bug relations. Due to this, the computational time is higher, but with consideration of additional information like bug reassignment, etc., there is a possibility of improving the performance, and the possibility of scalability also exists.

4.4. RS-based software bug triaging architecture

Recommender systems (Anvik & Murphy, 2011; Chamoso, Hernández, González-Briones, & García-Peñalvo, 2020; Chen et al., 2019a, Chen et al., 2019b; Florea et al., 2017b, 2017a; Xie et al., 2012; Zaidi et al., 2020) are also used for SBT, where expert developers are recommended for the newly reported software bugs. Using RS, a list of software developers and ranking of the top K developers can be generated for or assigning the newly reported software bugs. Some of the earlier studies in the field of RS for SBT are proposed by Anvik (2006), Anvik and Murphy (2011), and Xuan et al. (2012). In the prior studies (Xuan et al., 2012), a ranking of developers was performed by developer prioritization. A typical RS is presented in Fig. 8. As shown in Fig. 8, where the output of the RS is a ranked list of the software developers for fixing newly reported bugs. The typical performance evaluations for recommender systems (Shani & Gunawardana, 2011) or ranking systems are Accuracy@K (Acc@K), Precision@K (P@K), and Recall@K (R@K). K indicates the top K developers from the recommended list of developers. The summary of work done in the direction of developer ranking for fixing newly reported bugs using RS and ranking techniques is presented in Table 7, and further RS-based studies are presented in Appendix C (Table C4). Most of the studies are performed on Eclipse and Mozilla Firefox. Recall@10 is achieved up to 90%. The key advantages of RS-based techniques are a list of ranked developers generated so the triager has a choice to select the developer who is presently available and quickly fix the bugs. The key challenges in these techniques are the challenges of a general RS like the problem of cold-start (Mishra et al., 2021), and performance, and data sparseness (Najafabadi et al., 2019). The problem of cold-start and data sparseness arises due to the lack of data related to a particular item or category, which causes the degradation of the performance of the recommender system. In the case of bug triaging, the RS-based approach cannot identify the exact expert if there is not sufficient data available for that expert developer.

4.5. MMO-based software bug triaging

Various other mathematical modeling and optimization models are also explored for software bug triaging tasks. Fuzzy sets (Kumar et al., 2020a, Kumar et al., 2020b; Tamrawi et al., 2011b, 2011a), Optimization (Goyal & Sardana, 2017c; Wei et al., 2018; Zhang et al., 2017), and Knapsack programming (Kashiwa & Ohira, 2020) have also been used for bug triaging problems. A genetic algorithm-based approach for software bug triaging was proposed by Lee et al. (2019) for bug triaging. Optimization techniques like Ant Colony Optimization (ACO) are used for SBT in a few studies but the work is not covered in depth (Akila et al., 2015; Akila et al., 2016). Key works for SBT using MMO-based techniques are summarized in Table 8, and further MMO-based studies are presented in Appendix C (Table C5). Most of the works are

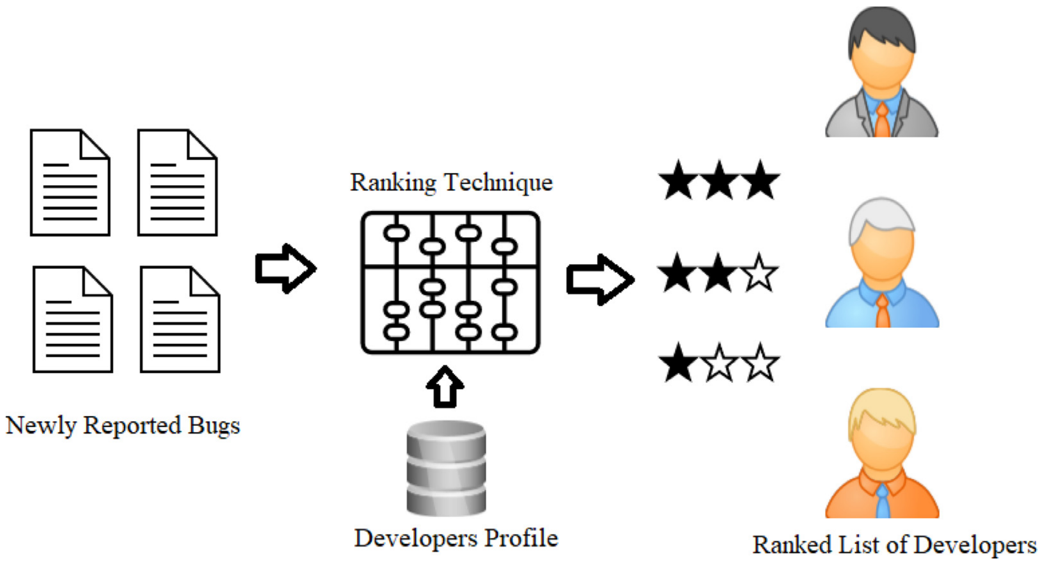

Fig. 8. A typical recommender system for software bug triaging.

Table 7
RS-based SBT models.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Kanwal & Maqbool, 2012)	2012	Classification-based ranking system	Journal of Computer Science and Technology	Vector Space	Eclipse	SVM Precision 54% and Naïve Bayes Precision is 40%
(Xia et al., 2015b)	2015	Developer Ranking	Journal of Software Evolution and Process	Vector Space	GCC, OpenOffice, Mozilla, NetBeans, and Eclipse	recall@5 0.4826 to 0.7989, recall@10 0.6063 to 0.8924
(Yadav et al., 2019)	2019	Developer Expertise Score	Information and Software Technology	Vector Space	Mozilla, Eclipse, NetBeans, Firefox, and Free desktop	Accuracy of 89.49%, precision of 89.53%, recall rate of 89.42%, and F-measure of 89.49%.
(Alkhazi et al., 2020)	2020	Feature aggregation and ranking	Applied Soft Computing	Vector Space	Eclipse UI, Bert, JDT, SWT	Accuracy@5 achieved is 80%
(da Silva et al., 2020)	2020	Developer profile-based recommender system	Conference	Vector Space	GitHub	Recall@5 is 0.81

Table 8
Optimization and fuzzy logic-based techniques.

Author(s)	**Year**	**Technique**	**Published**	**Representation**	**Dataset**	**Performance**
(Tamrawi et al., 2011a)	2011	Fuzzy set	Conference	Number	Eclipse	Accuracy@10 is 71%
(Tamrawi et al., 2011b)	2011	Fuzzy set	Conference	Number	Eclipse	Accuracy 37.81%, and Top-5 accuracy 68%
(Liu et al., 2016)	2016	Time aware optimization	IJSEKE	Series	Bugzilla	Optimization between accuracy and time cost
(Goyal & Sardana, 2017c)	2017	MCDM	Intelligent Decision Technologies	Number	Mozilla and Eclipse	Accuracy of 70.59% and 86.15% in Mozilla and Eclipse
(Kashiwa & Ohira, 2020)	2020	Knapsack Programming	IEICE TRANS. INF. & SYST.	Number	Eclipse, GCC, and Mozilla	Bugfix duration reduced by 35%–41%, as compared with manual triaging

presented in the Eclipse bug repository and the accuracy of bug triaging is achieved up to 86% using these models. Under MMO-based SBT techniques, the researcher mainly concentrates on developing mathematical formulation and objective functions to handle software bug triaging. For example, in the case of fuzzy modeling-based techniques, fuzzy representation of software bugs is required, and membership functions are developed between software bug terms and developers for triaging. In the case of optimization techniques for software bug triaging, optimization constraints are developed considering the number of bugs fixed by a particular developer in a given time. Time is an essential constraint for fixing software bugs particularly when the bug is of high priority or security bug. After framing the constraint of optimization mathematical models are developed and then data is extracted from the bug repository and the constraints are solved to get the most suitable and relevant developer for fixing up the newly reported bugs.

The optimization based SBT techniques require proper mathematical modeling of SBT problems in terms of the optimization problem. For example, in ACO (Ant Colony Optimization) based SBT system (Akila et al., 2015; Akila et al., 2016), first the bug tossing graphs are generated from the historical bug fixing information and then Ants are allowed to traverse through with these tossing graphs for detection of optimum paths in the input tossing graphs. The optimized path indicates the selection of developers for the assignment of the reported bugs. ACO modeling requires parameters like number of Ants, number of iterations, developer network, pheromone, etc. In multiple knapsack optimization based SBT techniques (Kashiwa & Ohira, 2020), the bug fixing, and developer metadata are transformed into the knapsacks. The capacity of knapsacks is represented by the time limit for fixing the reported software bugs. The items and knapsacks in the knapsack optimization are represented by the number of bugs and developers respectively. Genetic algorithm-based optimization techniques (Lee et al., 2019) for SBT, genetic fitness functions are defined considering the list of words of bugs along with other bug information like bug-id. Similarity calculations are performed for cluster centers with the fitness functions considering the developer's information. The maximum similarity indicates better belongingness in a cluster of developers for fixing the given software bugs. In greedy search-based optimization techniques (Rahman et al., 2009) for SBT, greedy search space is created by considering the information of individual developers available for a specified time for fixing the bugs. Then for bug triaging, distance functions are used to calculate the distance of all the available developers and assign the bugs to the developers with the lowest distance.

The advantages of optimization and fuzzy technique based SBT techniques are using mathematical modeling it is easy to identify the desired constraints to optimize the bug triaging work. The major challenges with these techniques are the performance of bug triaging (Goyal & Sardana, 2017c; Wei et al., 2018; Zhang et al., 2017). The applicability of such technologies in a real-time scenario is a challenge as there are chances that either the identified developers have left the company or engaged in some other work.

In fuzzy logic-based bug triaging techniques, membership functions are developed for representing relationships between the newly reported software bugs to the most appropriate software developers who can fix them (Jindal & Kaur, 2020; Panda & Nagwani, 2021; Vu et al., 2020). On arrival of the new software bugs in the bug repository, the similarity between the terms of newly reported bugs with the existing bug terms is calculated and then by applying fuzzy logic membership value are calculated for the newly reported bugs for predicting the developers. The higher membership value indicates the higher possibility of fixing the bugs by the software developers. Recently, a few studies have also been proposed where fuzzy logic using multi criteria decision making (Gupta et al., 2021) and advanced fuzzy sets like intuitionistic fuzzy sets (Panda & Nagwani, 2022) are utilized for software bug triaging.

The primary advantage of fuzzy logic is that fuzzy computation is simple and requires less computing infrastructure (Panda & Nagwani, 2021). As there may be several competent software developers, who can work and fix the newly reported software bugs, fuzzy membership will clearly indicate this possibility by computing the membership value and giving an option to triager the competency-wise list of developers for the newly reported software bugs. The disadvantage of fuzzy logic-based SBT techniques is that the development of the most relevant and logical membership function is a tricky task and requires a lot of logic-building activities to model the bug triaging task in a fuzzy environment. Another problem in fuzzy logic-based techniques is, very little work is reported in this direction, and out of these limited studies making a generalized conclusion is not possible. So, the applicability of these techniques in real-world scenarios of bug triaging is still challenging.

4.6. DL-based software bug triaging techniques

DL is becoming popular in recent times and is also applied for the task of SBT tasks (Mani et al., 2019). DL techniques like Convolutional Neural Network (CNN) (Lee et al., 2017; Florea et al., 2017a; Guo et al., 2020; Ye et al., 2020; Zaidi et al., 2020) and Recurrent Neural Network (RNN) (Florea et al., 2017a; Xi et al., 2018a; Xi et al., 2018b; Koc et al., 2019; Mani et al., 2019) are widely used in the area of software bug triaging. Both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are popular architectures in deep learning and are used in the area of software bug triaging. The primary difference between CNN and RNN is that CNN is a densely connected feed-forward network, whereas RNN is a feed-backward network that consumes the information from the previous outcome for improving the weights. Bug summary and description attributes are the key attributes considered in the bug triaging task and both attributes are textual, thus, the word embedding techniques like Word2Vec, and Glove are also emerging as key technologies to be utilized in the directions of SBT. The summary of selected studies in direction of SBT using DL-based techniques is presented in Table 9, and further DL-based studies are presented in Appendix C (Table C6). Experiments are performed on a variety of datasets and accuracies obtained are in the range of 57% to 87% using DL-based models. The Word2Vec (Lee et al., 2017; Florea et al., 2017a; Choquette-Choo et al., 2019; Koc et al., 2019; Mani et al., 2019; Huang & Ma, 2019; Guo et al., 2020; Ye et al., 2020; Zaidi et al., 2020) embedding technique is used widely as compared to the Glove (Xi et al., 2018a; Xi et al., 2018b; S. Q. Xi et al., 2019; Zaidi et al., 2020) embedding technique in the presented work of software bug triaging. The CNN architecture for SBT is shown in Fig. 9.

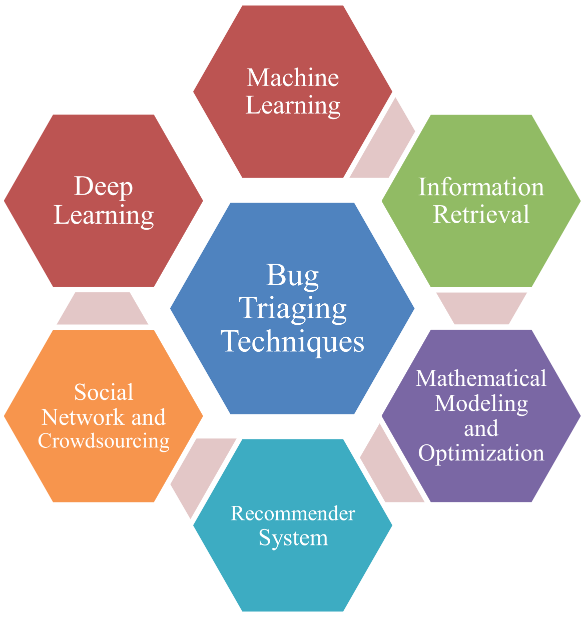

Fig. 9. CNN-based architecture for BT.

CNN architecture is shown in Fig. 9 that consists of convolutional layers and pooling layers. The purpose of these layers is feature extraction. Classification is performed at the last stage, for SBT, each software developer is represented as a category. DL provides excellent performance, scalability, and learning rate (time required for model learning) as compared to other AI-based techniques. But computational time and computing infrastructure is a key challenge in DL-based techniques. Training time is higher than the traditional machine learning or information retrieval-based techniques (Mani et al., 2019). This is primarily due to multiple layers of computing (Wani et al., 2020). Smaller and low-budget IT companies may not be able to afford to have DL technologies so other technologies with improvements can still be a choice for such IT companies.

A typical CNN multilayer architecture (Guo et al., 2020; Zaidi et al., 2020) of DL-based systems for SBT consist of word vector representation layers, convolutional layer, pooling layer, and activation functions for aggregating the output values. The convolution layer performs convolution operations for training the samples and training the input samples. The purpose of the pooling layer is to extract the task-relevant sub-sample from the feature space. Max pooling techniques are most widely used in the existing studies as they give high performance.

Table 9
DL-based techniques for software bug triaging.

Author(s)	Year	Model	Published	Embedding	Dataset	Performance
Xi *et al.*(S. Q. Xi et al., 2018)	2018	RNN	Journal of Software	Glove	Eclipse, Mozilla, and Gentoo	Top-5 accuracies for Eclipse, Mozilla, and Gentoo are 82.42%, 70.16%, and 79.91%
Xi *et al.*(S. Q. Xi et al., 2019)	2019	SVM and RNN	Journal of Computer Science and Technology	Bag of Words and Glove	Eclipse, Mozilla, Gentoo	Accuracy 0.65
Guo *et al.*(S. Guo et al., 2020)	2020	CNN	Neural Processing Letters	Word2Vec	Eclipse, Mozilla, and NetBeans	74.89% Accuracy
Ye *et al.*(Ye et al., 2020)	2020	CRNN	Journal of Computer Research and Development	Word2Vec	Eclipse and Mozilla	Eclipse Top-5 77.18 Mozilla Top-5 66.68
Zaidi *et al.*(Zaidi et al., 2020)	2020	CNN	IEEE Access	Word2Vec, Glove, and ELMo	JDT, Platform	Top-10 accuracies for JDT, Platform, and Firefox are 87.23, 74.26 and 64.70

Apart from the general deep learning architectures, customized deep learning architectures such as graph recurrent convolutional neural network (GRCNN) model (Wu et al., 2022) and reinforcement deep learning models (Liu et al., 2022) are also presented recently. The graph recurrent convolutional neural network (GRCNN) model gives F1@5 scores of 86.74% and 75.64% on the Eclipse and Mozilla projects. Deep reinforcement learning model gives 52%, 54%, 68%, and 78% top-5 accuracies for OpenOffice, NetBeans, Mozilla, and Eclipse datasets.

5. Performance evaluation

Performance is an important aspect in evaluating any bug triaging technique. Performance metrics decide the adaptability of any newer techniques in AI, and hence in bug triaging also. The performance assessment of the bug triaging technique depends on the type of machine learning and other algorithm selection. For example, in a classification-based bug triaging technique, accuracy (ACC), precision (P), recall (R), and F-measure (F1) are derived using a confusion matrix (Han et al., 2011). Precision is the ratio of relevant software developers or actual assignees of the bugs to the total developers identified by the machine learning algorithm. The recall is the ratio of relevant software developers with the total number of relevant software developers identified by the machine learning algorithm. F1-measure is the combined measure (harmonic mean) of precision and recall. Accuracy defines the ratio of correctly assigned developers to the total developers considered for software bug triaging. These measures can be calculated directly or can be calculated with the help of a confusion matrix. Precision (P), recall (R), F1-measure (F1), and accuracy (ACC) are calculated using Eq. (2), and Eq. (3) respectively. TP, TN, FP, and FN are the elements of the confusion matrix and represent the true positives, true negatives, false positives, and false negatives.

$$P = \frac{TP}{TP + FP},\ R = \frac{TP}{TP + FN},\ F1 = \frac{2 \times P \times R}{P + R} \tag{2}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Dedicated research on the evaluation of bug assignment was carried out by Sajedi-Badashian and Stroulia (2020a). They have included 74 studies on software bug triaging and listed the evaluation metrics used for evaluating the SBT process. Evaluation metrics for developer ranking top-K accuracy, Precision@K (P@K), Recall@K (R@K), mean reciprocal rank (MRR) and mean average precision (mAP) are covered in the study where K represents the number of recommended software developers. Top-K accuracy indicates the number of real bug assignees out of the K-recommended developers. It is calculated at top K recommended developers as the ratio of recommended developers divided by total developers. P@K is the ratio of relevant (real bug assignee) recommended developers to the total recommended developers by the bug triaging technique. R@K is the ratio of relevant (real bug assignee) recommended developers to the total relevant developers by the bug triaging technique. P@K and R@K are calculated using Eq. (4). F1@K is the harmonic mean of P@K and R@K. F1@K is calculated using Eq. (5). MRR is the mean reciprocal rank of highly ranked developers. Parameter mAP is the mean of average precision (AP), i.e., the average of all the precision values for all ranks of developers for the real assignment of the bugs. Average precision (AP) is calculated using Eq. (6). A working example of the calculation of performance metrics is presented in Appendix E.

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

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

$$AP = \sum (Recall@K\ -\ Recall@(K-1)) \cdot Precision@K \tag{6}$$

Apart from the above-mentioned metrics, DCG@K and NDCG@K are also widely used for evaluating recommender systems. DCG is Discounted Cumulative Gain and NDCG is Normalized Discounted Cumulative Gain. Both measures are used for rating the quality of the developer ranking during the SBT process. The expression for DCG@K and NDCG@K is shown in Eq. (7), where rel_i is the relevance of the i^{th} developer recommendation and IDCG is the Discounted Cumulative Gain in the Ideal condition.

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

Several performance parameters are discussed in this section for evaluating the performance of SBT techniques. Although Sajedi-Badashian and Stroulia (2020a) concluded that mAP is the best evaluation metric for evaluating the SBT process, the evaluation also depends on the type of AI-based technique selected for bug triaging. For example, if recommender system-based techniques are used for it, then DCG@K and NDCG@K will be the suitable metrics for evaluating the performance (Shani & Gunawardana, 2011). Similarly, for machine learning and information retrieval-based techniques, accuracy, precision, recall, and F-measure are considered to be popular metrics (Han et al., 2011).

6. Challenges and future directions

The categorized summary of a review of literature is presented in the previous section where it is shown that researchers have presented numerous ways of developing bug triaging approaches. The open challenges in bug triaging techniques along with their possible solutions are summarized in this section. For addressing the future research directions, the future scope of the relevant selected studies is also extracted and a few more related works in software repository mining and expert identification are identified for the discussion of future work and key challenges in the direction of SBT. The future directions are identified with the help of existing and related works from the various domains that can also be explored for software bug triaging works.

i *Developers Vocabulary Generation:* The generation of accurate and perfect developers' vocabulary is an essential activity in the SBT process. While developing the SBT techniques, bug information and technical skills of the software developer are considered for assigning the bugs to the right fixers, but apart from the technicality, there are other attributes also presented in the recent studies (Kalliamvakou et al., 2019; Li et al., 2019), by which a software engineer or developers can be termed as an expert, maybe included for improving developer vocabulary generation for future SBT techniques.

ii *Data Reduction Techniques:* Fan et al. (2020) has worked on filtering and eliminating invalid bugs from the bug repositories, which will reduce the bug triaging data and at the same time save time and energy for managers and triagers. They have also identified five feature groups in their work namely, the experience of the reporter, collaboration network, completeness, and text. Then, a Random Forest classifier was applied to categorize the invalid bug. They have also provided an overview and more such possible works in the direction of data reduction in the area of SBT. A similar approach was adopted by Goyal and Sardana (2019a), in which identification and handling of non-reproducible bugs are performed in a software bug repository.

iii *Bug Prioritization:* Mostafa et al. (2021) presented their work in the direction of prioritization of security bugs, as it is an essential exercise in software development. Security patches are a top critical activity in software development. They suggested the task of reducing the training data size and application of transfer learning for further improvement. Similar approaches may also be carried for developing future SBT techniques.

iv *Feature selection and aggregation:* Feature selection, aggregation, and ranking (Alkhazi et al., 2020) is a common and popular task in pre-processing of the SBT process. A few selected supporting works in this direction are discussed here in brief. Martínez-García et al. (2020) have proposed the development of an ontology for expertise location-based knowledge condensation during the coding phase of software development. The ontology then can be utilized for bug triaging at the later phases. The task of generating user summaries for recommending similar projects in software development is presented by Resketi et al. (2020), which can be combined and utilized for SBT also. Another similar work was presented by Nayebi et al. (2019), where the authors have presented a method for text summarization, predicting ticket escalation, creating the ticket's title and content, and assigning the ticket to an available developer. A similar technique can also be developed and proposed for software bug triaging. Apart from traditional feature selection techniques, feature enhancement for bug triaging is also carried out in some studies (Alazzam et al., 2020) where additional useful features and metrics are derived for bug triaging work.

v *Metrics:* The impact analysis of bug fixing operation using four popular internal quality attributes was carried out by Kumar et al. (2020a), Kumar et al. (2020b), where the quality attributes complexity, cohesion, inheritance, and coupling were covered in the analysis. In future work, we will focus on other software quality attributes like maintainability, reusability, and refactoring.

vi *Exploring Large Bug Repositories:* Most of the proposed bug triaging techniques were applied on publicly available bug repositories like Mozilla, Apache, Google, Git, etc. for evaluation of the proposed bug triaging techniques. Most of the studies like Martínez-García et al. (2020) have suggested considering the big repositories for further improving the bug triaging processes and to include modern technologies like big data analytics.

vii *Network and Graph-based approaches:* Alazzam et al. (2020) suggested incorporating semantic, multiplex, and multimode networks that can be utilized to model the bug report components relationships for developing newer bug triaging models. Kim et al. (2020), proposed the development of a DAG (Directed Acyclic Graph)-based visual analysis tool that can be used to visualize the relationship between developers and can be utilized for bug triaging. Xi et al. (2019) proposed a technique named iTriage, which generates a sequence-to-sequence model for learning the textual features and tossing sequence. The generated information can be utilized for SBT. One similar work in the GitHub repository is presented by Zhang et al. (2019), where semantically linked issues are reported in the GitHub repository. The proposed work can be utilized for the identification of similar bugs, finding critical and useful information for quick resolution of the issues. Bug dependency and dependency graphs can also be utilized for improving bug triaging (Almhana & Kessentini, 2021).

viii *Cross Repository Analysis:* Limited work is proposed in SBT using cross repository analysis. An automatic commit message generation technique for version control systems is proposed by Huang et al. (2020), the commit message was generated based on most similar past commits from a large commit repository. Another relevant supporting work is presented by Yang et al. (2019), where they proposed a personalized recommender system for open source repositories. An interest measurement model on GitHub is proposed for developers working on a project considering the common technical interest of other developers. The proposed work can be modified to link with the bug repositories for developing the newer SBT techniques.

ix *Advanced DL-based Models:* DL-based techniques are developed and proposed by various researchers for SBT as discussed in the previous section. The Convolutional Neural Network model of DL along with the Word2vec word embedding model was utilized frequently for bug triaging e.g., Guo et al. (2020) apart from the proposed technique, DL provides a vast futuristic opportunity by applying the modern and more recent models in SBT. Variations of CNN such as bioinspired piking CNN (SCNN), which is considered higher performing than the traditional. Non-spiking neural networks due to their bio-realism (Zaidi et al., 2020) can be utilized for the SBT process. Further, Chamoso, Hernández, González-Briones, & García-Peñalvo (2020) proposed technological profile recommendations using document embedding models, and Jiechieu and Tsopze (2020) proposed skills prediction based on multi-label resume classification using CNN with model predictions, it can also be utilized for SBT as a future direction work.

x *Modern Machine Learning and Big Data Approach:* Modern machine learning techniques such as ensemble learning and transfer learning along with Big Data analytics can be explored further to develop more efficient bug triaging techniques. Techniques such as label prediction in bug repositories by Alonso-Abad et al. (2019), identification of the distinguishing characteristics of great software engineers by Li et al. (2019), software fault prediction technique by Rathore and Kumar (2017), and automated issue assignment technique by Aktas and Yilmaz (2020) can be explored and utilized for developing newer techniques in SBT. Blockchain technology for bug triaging is also proposed in a recent study (Gupta & Freire, 2021).

xi *Handling class imbalancing:* Class imbalancing is a practical challenge in artificial intelligence approaches. In context to SBT, class imbalancing can occur in cases when there is a non-equal distribution of fixing the number of bugs by the developers. In other words, few developers have fixed a greater number of bugs and others have fixed only a few. It can lead to class imbalancing in the training dataset for bug triaging. Class imbalancing is a practical problem, in a few recent studies it is explored for defect prediction model (Bejjanki et al., 2020; Gong et al., 2019; Tantithamthavorn et al., 2018; Yuan et al., 2020), predicting the fixability of bugs (Goyal & Sardana, 2017b), and classifying the bugs (Chen et al., 2019a, Chen et al., 2019b) but for software bug triaging no in-depth studies are presented in this direction yet. A similar model can be extended considering the recent technologies like deep learning and soft computing as a future direction for handling the class imbalancing due to the unavailability of the software developers.

The third dimension is listing the challenges and possible future works in the direction of SBT. For analyzing the third dimension, the future scope of the selected studies is extracted, and supporting work in the direction of SBT is listed. Based on the extracted information, key challenges, and future directions in the field of SBT were summarized and presented in the previous section. The future directions are grouped into several topics and provide a comprehensive summary of possible research scope in all those identified groups.

7. Discussions

Artificial intelligence has given a new orientation in the direction of developing automated and accurate SBT systems. Looking into the covered studies and from the AI technological perspective, it can be inferred that AI technology is our trending tool for SBT and helps in improving and automating the SBT process. So, our hypothesis holds true that AI is improving and contributing effectively to automating the SBT process. So, from the literature and presented review it is proven that AI is improving and contributing effectively to automating the SBT process. However, as technologies are evolving there will always be a scope for further improvements. The answers to the three research questions are discussed below.

7.1. RQ1: can AI improve and contribute effectively to automating the SBT process?

Based on the comprehensive review carried out, a brief discussion on the assumed hypothesis and recommendations of selecting the appropriate AI-based BT technology is discussed in this section. The hypothesis was presented using the analysis of three dimensions of the BT techniques. The first dimension is trending AI research technologies for BT. The findings for the first dimension are, around a decade ago machine learning techniques were very popular for SBT, and then information retrieval-based techniques were introduced followed by the recommender systems, and in the very recent time, DL-based techniques are utilized for bug triaging. The paradigm shift of the bug triaging technologies is visible from the literature survey presented in the proposed study. DL-based techniques were explored in recent studies and are expected in more upcoming studies in the direction of software SBT.

It is observed from the study that AI technologies, from machine learning and information retrieval to DL, were explored in various studies for automating bug triaging. Automated bug triaging will not only save the time of overall software development, but at the same time, it will minimize the cost. Hence it can be concluded that AI plays a key role in software development, particularly in SBT, and the hypothesis, AI is contributing to automation and improvement of the SBT process holds true.

7.2. RQ2: identify the major performance parameters used in the existing studies?

The second dimension is the study of performance evaluation and parameters for the evaluation of AI-based SBT techniques. The finding of the second dimension's analysis is that the choice of performance parameters depends on the type of technology used for developing the SBT techniques. For example, in a classification-based bug triaging technique, accuracy (ACC), precision (P), recall (R), and F-measure (F1) are derived using a confusion matrix Evaluation metrics for developer ranking top-K accuracy, Precision@K (P@K), Recall@K (R@K), mean reciprocal rank (MRR) and mean average precision (mAP) are covered in the study where K represents the number of recommended software developers. DCG@K and NDCG@K are also widely used for evaluating recommender systems.

Although Sajedi-Badashian and Stroulia (2020a) concluded that mAP is the best evaluation metric for evaluating the SBT process, the evaluation also depends on the type of AI-based technique selected for bug triaging. For example, if recommender system-based techniques are used for it, then DCG@K and NDCG@K will be the suitable metrics for evaluating the performance. Similarly, for machine learning and information retrieval-based techniques, accuracy, precision, recall, and F-measure are considered to be popular metrics.

7.3. RQ3: list the challenges and issues and identify possible future work for further improving the SBT process?

The challenges and issues in existing SBT techniques provide the possible future scope for improvement of the SBT process. The challenges and issues and possible future scope are summarized in section 6. Eleven categories for possible future scopes are identified and summarized. These categories are, developers' vocabulary generation, data reduction techniques, bug prioritization, feature selection and aggregation, metrics, exploring large bug repositories, network and graph-based approaches, cross repository analysis, advanced DL-based models, modern machine learning and Big Data approach, and handling class imbalancing. A brief overview of all these categories of possible future directions is presented in section 6. It is concluded that in spite of the fact that although there are plenty of existing works for automating the SBT process using AI, still there is a huge scope of further improvement. SBT automation can be further improved in the real-world scenario for the IT companies using the listed possible future scope.

7.4. Recommendations for practice

Each AI-based BT architecture has merits and demerits and practically there are several factors that can be considered for selecting a particular AI technology for bug triaging discussed in section 3. It is also observed that DL-based techniques provide better results in terms of performance, and scalability as compared to other AI-based architectures. An indicative summary of the selection of appropriate AI techniques is provided in Appendix-F for reference. The choice of selecting appropriate AI technology depends on various factors and varies from organization to organization. Major factors related to IT industries are considered for selecting the appropriate AI technology. The major factor includes time constraints (for fixing the bugs), performance (performance of the AI technologies), bug prioritization, model enhancement capabilities, and capabilities of handling large repositories are considered for selection of an appropriate AI technology for automation of software bug triaging.

8. Conclusion

Software bug triaging is a key area in mining software repositories, and it got attention in the past decade and attracted many researchers to develop and propose newer models and techniques for triaging the newly reported bugs. AI technologies help in automation of software bug triaging processes. By automating the bug triaging process, the information of bug repositories can be managed in an effective way. The presented work provides a comprehensive narrative review and meta-analysis of the major contribution made in the direction of SBT. Selection of related work is done as per the PRISMA-based analysis and a categorized comprehensive literature review is presented for all the possible categories of SBT techniques. Three research questions are framed, and their answers are also discussed. A three-dimensional analysis is carried out for software bug triaging works to answer three research questions. Performance evaluation of SBT techniques along with the commonly used performance parameters for SBT work is also explained. Challenges and future directions in software bug triaging are discussed considering various aspects and possible dimensions of developing futuristic software bug triaging techniques. Pros and cons of all the categories of bug triaging architectures are also discussed, which can provide useful information and parameters to the stakeholders to decide about the selection of appropriate bug triaging architecture and its performance evaluation. AI has changed the entire paradigm of manual bug triaging with the semi-automated and automated bug triaging system, as newer technologies are emerging, more advanced bug triaging systems are expected in the near future also.

Declaration of Competing Interest

The author declares that there is no conflict of interest.

Appendix A

Table A1, B1,B2, C1, C2, C3, C4,C5,C6, F1,F2,F3,F4

Graphical summary of AI-based architectures for software bug triaging

Table A1
List of abbreviations (in alphabetic order).

Abbreviation	Description
BT	Bug Triaging
DCG	Discounted Cumulative Gain
DL	Deep Learning
IR	Information Retrieval
mAP	Mean Average Precision
ML	Machine Learning
MMO	Mathematical Modelling and Optimization
NDCG	Normalized Discounted Cumulative Gain
RS	Recommender System
SBT	Software Bug Triaging
SBTT	Software Bug Triaging Techniques
SNA	Social Network Analysis
TG	Tossing Graph
TM	Topic Modelling

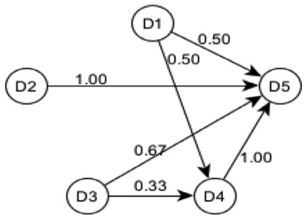

Fig. A1. Graphical summary of AI-based architectures for software bug triaging.

Appendix B

List of key journals and popular conference papers in software bug triaging

Table B1
List of reputed journals in the area of software bug triaging.

SN	Journal Name	ISSN	Publisher	IF[**]
1	IEEE Transactions on Software Engineering	0098-5589	IEEE	6.112
2	Neural Computing and Applications	0941-0643	Springer	4.774
3	ACM Transactions on Software Engineering and Methodology	1049-331X	ACM	3.857
4	Empirical Software Engineering	1382-3256	Springer	3.156
5	Information and Software Technology	0950-5849	Elsevier	2.726
6	Journal of Systems and Software	016-1212	Elsevier	2.45
7	Journal of Computer Science and Technology	1000-9000	Springer	1.506
8	E-Informatica Software Engineering Journal	1897-7979	WUST[*]	1.278
9	IET Software	1751-8806	IET	1.258
10	Journal of Software: Evolution and Process	2047-7481	Wiley	1.178
11	Int. Journal of Software Engineering and Knowledge Engineering	0218-1940	WS+	0.886

[*] Wroclaw University of Science and Technology; WS+: World Scientific; IF.
[**] : Impact Factor.

Table B2
List of important conference papers with citations in Google Scholar and Scopus.

Author	Title	Year	Google Scholar Citations	Scopus Citations
(Murphy & Cubranic, 2004)	Automatic bug triage using text categorization	2004	560	50
(Anvik et al., 2006)	Who should fix this bug?	2006	1325	717
(Anvik, 2006)	Automating Bug Report Assignment	2006	137	67
(Jeong et al., 2009)	Improving bug triage with bug tossing graphs	2009	562	289
(Matter et al., 2009)	Assigning bug reports using a vocabulary-based expertise model of developers	2009	289	154
(Ahsan et al., 2009)	Automatic Software Bug Triage System (Bts)-based on Latent Semantic Indexing and Support Vector Machine	2009	102	50
(Baysal et al., 2009)	A Bug You Like: A Framework for Automated Assignment of Bugs.	2009	87	43
(Rahman et al., 2009)	Optimized Assignment of Developers for Fixing Bugs an Initial Evaluation for Eclipse Projects	2009	60	33
(Bhattacharya & Neamtiu, 2010b)	Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging	2010	187	117
(L. Chen et al., 2010)	Improving bug assignment with bug tossing graphs and bug similarities	2010	151	5
(Xuan et al., 2017a)	Automatic bug triage using semi-supervised text classification	2017	155	53
(Anvik & Murphy, 2011)	Reducing the effort of bug report triage: Recommenders for development-oriented decisions	2011	338	162
(P. J. Guo et al., 2011)	Not my bug! and other reasons for software bug report reassignments	2011	158	87
(Tamrawi et al., 2011b)	Fuzzy set-based automatic bug triaging (NIER track)	2011	90	52
(Tamrawi et al., 2011a)	Fuzzy set and cache-based approach for bug triaging	2011	188	120
(W. Wu et al., 2011)	Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking	2011	129	68
(J. Park et al., 2011)	Costriage: A cost-aware triage algorithm for bug reporting systems	2011	108	34
(Xie et al., 2012)	Dretom: Developer recommendation-based on topic models for bug resolution	2012	107	55
(Linares-Vasquez et al., 2012)	Triaging incoming change requests: Bug or commit history or code authorship?	2012	143	72
(Xuan et al., 2012)	Developer prioritization in bug repositories	2012	212	105
(Xia et al., 2013)	Accurate developer recommendation for bug resolution	2013	152	93
(Shokripour et al., 2013)	Why So Complicated? Simple Term Filtering and Weighting for Location-based Bug Report Assignment Recommendation	2013	156	92
(H. Hu et al., 2014)	Effective Bug Triage-based on Historical Bug-Fix Information	2014	91	43
(Mao et al., 2015)	Developer Recommendation for Crowdsourced Software Development Tasks.	2015	114	61

Appendix C

Summary of Categorized BT Techniques

Table C1
ML-based SBT techniques.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Anvik, 2006)	2006	Recommender system	Conference	Vector Space	Eclipse, Firefox	The precision of 86%
(Ahsan et al., 2009)	2009	LSI, SVM	Conference	Matrix	Mozilla	Accuracy 44.4%, Precision, and Recall are 30% and 28%.
(Jeong et al., 2009)	2009	Tossing Graphs	Conference	Tossing Graph	Eclipse and Mozilla	Accuracy of 75.88% using Bayesian Network. Reduced tossing events, by up to 72%.
(Xuan et al., 2017a)	2017	Naïve Bayes, EM	Conference	Vector	Eclipse	accuracy 48.07%
(Bhattacharya & Neamtiu, 2010b)	2010	Multi-Feature Tossing Graph Classification	Conference	Tossing Graph	Mozilla and Eclipse	Accuracy of 83.62%
(L. Chen et al., 2010)	2010	Tossing Graphs with Bug Similarity	Conference	Tossing Graph and Vector Space Model	Eclipse and Mozilla	The tossing path length is reduced
(W. Wu et al., 2011)	2011	K-Nearest-Neighbor	Conference	TF-IDF	Mozilla Firefox	Recall@10 is 0.65, Precision@10 is 0.65
(L. Chen et al., 2011)	2011	Tossing Graphs with Bug Similarity	Journal of Software	Tossing Graph and Vector Space Model	Eclipse and Mozilla	MLTP (Mean Length of Tossing Path) is reduced by 76.25%
(Xia et al., 2013)	2013	KNN, LDA	Conference	Vector Space	GCC, OpenOffice, Mozilla, NetBeans, and Eclipse	Achieve recall@5 and recall@10 scores of 0.4826–0.7989, and 0.6063–0.8924, respectively
(Naguib et al., 2013)	2013	LDA-SVM	Conference	Vector Space	Open-source bug repositories	achieve an average hit ratio of 88%
(N. Goyal et al., 2015)	2015	Classification and Clustering	Advances in Intelligent Systems and Computing	Vector Space	Custom real-world data	Accuracy and Recall of about 72% and Precision of about 67% is achieved
(Cavalcanti et al., 2016)	2016	Rule-based approach, SVM	Journal of System and Software	Rule Representation	Various	Improvement on Accuracy as compared to sole machine learning-based approach

(continued on next page)

Table C1 (*continued*)

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Sharma & Singh, 2016)	2016	AR with Clustering	International Journal of Business Intelligence and Data Mining	Rule Representation	Thunderbird, Add-on SDK, and Mozilla	First clustering then association rule mining results in higher confidence rules.
(A. Goyal & Sardana, 2017a)	2017	Comparative Study	e-Informatica Software Engineering Journal	Various	Various	Information Retrieval-based techniques outperform Machine learning techniques
(Sharma et al., 2017)	2017	Association Rule Mining	International Journal of Reliability, Quality and Safety Engineering	Rule Representation	Sea Monkey, Firefox, and Bugzilla	Higher confidence value in the AR
(Sbih & Akour, 2018)	2018	Ensemble learning	Journal of Multiple-Valued Logic and Soft Computing	Vector Space	Custom dataset	Recall up to 96%
(Pahins et al., 2019)	2019	T-REC (LC25F and KNN), KNN, Random Forest, SVM, and MLP.	Conference	Vector Space	Custom real-world data	Accuracy@5 of 76.1%, Accuracy@10 of 83.6%, and Accuracy@20 of 89.7%.
(A. Goyal & Sardana, 2019b)	2019	Ensemble learning	Conference	Vector Space	Mozilla Firefox, Gnome, and OpenOffice	The highest accuracy of 82.35% is achieved
(Sarkar et al., 2019)	2019	logistic regression classifier	Conference	Vector Space	Ericsson	Highest precision and recall of 78.09% and 79.00%
(Sawarkar et al., 2019)	2019	Random Forest, SVM using linear, Polynomial, Radial, and Sigmoid, and J48	Conference	Vector Space	Kaggle	Accuracy 56.29, precision 75.83, and recall 81.25 achieved
(Matsoukas et al., 2020)	2020	SVM One-Vs-Rest	Conference	TF-IDF	GitHub	Accuracy achieved values to 0.8 and 0.7,
(Yadav & Singh, 2020)	2020	Developer profile-based recommender system	International Journal of Intelligent Systems Technologies and Applications	Vector Space	Eclipse and Mozilla	F-score of 90% and reduced bug tossing length of 11.8%.

Table C2
IR-based triaging techniques.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Matter et al., 2009)	2009	Developer Vocabulary	Conference	Term author matrix	Eclipse	Top-1 precision of 33.6% and Top-10 recall of 71.0%
(Baysal et al., 2009)	2009	Vector Space Model	Conference	Vector	Custom software projects	The precision of 64%
(Kagdi et al., 2012)	2010	LSI	Journal of Software Evolution and Process	Vector	ArgoUML, Eclipse, and KOffice	Accuracies achieved in the range of 47 and 96% for bug reports
(J. Park et al., 2011)	2011	LDA	Conference	Vector	Apache, Eclipse, Linux kernel, and Mozilla	Accuracy is 69.7% for Apache
(Xie et al., 2012)	2012	LDA	Conference	Vector	Eclipse JDT, and Mozilla Firefox	Recall of 82% and 50% with Top 5 and Top 7 recommendations achieved
(Linares-Vasquez et al., 2012)	2012	LSI Indexing	Conference	Vector	ArgoUML, JEdit, Mu Commander	Top-5 precision of 50%
(Xie et al., 2012)	2012	LDA	Conference	Vector	Eclipse JDT and Mozilla Firefox	Recall of 82% and 50% with Top 5 and Top 7 recommendations achieved.
(Shokripour et al., 2013)	2013	bug location information and term weighting	Conference	TF-IDF	Eclipse and Mozilla	accuracy of 89.41% and 67.9% for Eclipse and Mozi
(Shokripour et al., 2014)	2014	term-weighting technique	IET Software	TF-IDF	Eclipse JDT, ArgoUML, NetBeans	accuracy 67%
(Shokripour et al., 2015)	2015	time-metadata in TF-IDF	The Journal of Systems and Software	TF-IDF	Eclipse JDT, NetBeans, ArgoUML	accuracy 63.2%, mean reciprocal rank (MRR) 58.96%
(T. Zhang et al., 2016)	2016	Classification + Topic Modeling	Journal of Systems and Software	TF-IDF	GCC, OpenOffice, Eclipse, NetBeans, and Mozilla	F-measure is 80.25%, Precision and Recall is 83.72%, and 77.07%
(J. W. Park et al., 2016)	2016	Topic modeling-based and content-based recommender	Knowledge and Information System	Vector	Apache, Eclipse, Mozilla, Linux Kernel	Accuracy of 69.70% is achieved

(*continued on next page*)

Table C2 (*continued*)

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Banerjee et al., 2017)	2017	Random Forest	Information and Software Technology	Vector	Eclipse, Firefox, OpenOffice	Recall 95%
(W. Zhang et al., 2017)	2017	Entropy Optimized Latent Dirichlet Allocation	Entropy	Vector	Eclipse JDT and Mozilla Firefox	Recall up to 84% and Precision up 41%
(A. Goyal & Sardana, 2017a)	2017	Comparative Study	e-Informatica Software Engineering Journal	Various	Various	Information Retrieval-based techniques outperform Machine learning techniques
(da Silva et al., 2020)	2020	Developer profile-based recommender system	Conference	Vector Space	GitHub	Recall@5 is 0.81

Table C3
SNA-based models for triaging.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Mao et al., 2015)	2015	C4.5, Naïve Bayes, K-Nearest Neighbor	Conference	TF-IDF	TopCoder	Accuracy from 50% to 71% and Diversity from 40% to 52%
(da Silva et al., 2020)	2020	Developer profile-based recommender system	Conference	Vector Space	GitHub	Recall@5 is 0.81

Table C4
RS-based BT techniques.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Anvik, 2006)	2006	Recommender system	Conference	Vector Space	Eclipse, Firefox	The precision of 86%
(Anvik & Murphy, 2011)	2011	Recommender system	ACM Trans. Softw. Eng. Methodol	Vector Space	Eclipse, Firefox	Precision 98%, Accuracy 75%
(H. Hu et al., 2014)	2014	Recommender system	Conference	Vector Space and Network	Eclipse, Mozilla, and NetBeans	Accuracy of 61.19% is achieved using the SVM algorithm
(Yadav & Singh, 2020)	2020	Developer profile-based recommender system	Conference	Vector Space	Eclipse and Mozilla	F-measure of 90% and tossing length reduction of 11.8%.

Table C5
MMO-based BT techniques.

Author(s)	Year	Technique	Published	Representation	Dataset	Performance
(Gupta et al., 2021)	2021	Fuzzy Muti Criteria Decision Making	Journal	Number	GitHub	Harmonic mean of precision, recall, f-measure, and accuracy obtained is 92.05%, 89.21%, 85.09%, and 93.11% respectively.
(Panda & Nagwani, 2022)	2022	Intuitionistic Fuzzy Set Similarity	Journal	Number	Eclipse, Mozilla, and NetBeans	Accuracy of 0.93, 0.90, and 0.88 for the Eclipse, Mozilla, and NetBeans data sets
(Rahman et al., 2009)	2009	Greedy Search	Conference	Number	Eclipse JDT	The degree of total fitness is improved by 16%.
(Kashiwa, 2019)	2019	Knapsack Programming	Conference	Number	Eclipse, GCC, and Mozilla	Accuracy of 81.7% achieved

Table C6
DL-based BT techniques.

Author(s)	Year	Model	Published	Embedding	Dataset	Performance
(S.-R. Lee et al., 2017)	2017	CNN	Conference	Word2Vec	JDT, Platform, and Firefox	Accuracies for JDT, Platform and Firefox are 86.1%, 65.8%, and 57.1%
(Florea et al., 2017a)	2017	CNN and RNN	Conference	Word2Vec	Eclipse, Mozilla, and NetBeans	The precision of 0.9, F1 measure as 0.87, and training time of LSTM is 408 secs
(S. S. Xi et al., 2018)	2018	RNN	Conference	Glove	Eclipse, Mozilla, Gentoo	Accuracy 0.5428
(Choquette-Choo et al., 2019)	2019	Dual DNN	Conference	Word2Vec	proprietary dataset	Top-10 accuracy of 38.83%
(Koc et al., 2019)	2019	RNN, GNN, and LSTM	Conference	Word2Vec	OWASP dataset	LSTM achieve more than 98% for recall, precision, and accuracy
(Mani et al., 2019)	2019	RNN (DBRNN-A)	Conference	Word2Vec	Google Chromium, Mozilla, and Firefox	Rank-10triaging accuracy in the range of 34 – 47%.
(J. Huang & Ma, 2019)	2019	DL	Conference	Word2Vec	Eclipse	Link prediction accuracy Eclipse 0.731, Gnome 0.656, Node classification Eclipse 0.823 Gnome 0.625
(H. Wu et al., 2022)	2022	Graph RNN	Journal	Word2Vec	Eclipse and Mozilla	F1@5 scores of 86.74% and 75.64% on the Eclipse and Mozilla projects is achieved.
(Y. Liu et al., 2022)	2022	Deep Reinforcement Learning	Journal	Word2Vec	OpenOffice, NetBeans, Mozilla, and Eclipse	Maximum accuracy@5 is achieved as 78%.

Appendix D

Examples of Bug Representation Techniques

Vector and Matrix Representation

Examples of both representations are given below. Let us assume that the following four bugs are given with their titles.

B_1: Titlebar is not showing the correct information

B_2: Menubar is not working

B_3: Label display is incorrect

B_4: Titlebar information is too long

After the pre-processing task, the identified unique terms in order are T = [*correct, display, incorrect, information, label, long, menubar, show, titlebar, work*]. Total unique terms are 10 for this example. If the frequency of the terms is taken in the vector representation model, then the bugs can be represented as below, where 1 indicates the presence of a term in the bug and 0 indicates that the term is not present in the bug.

B_1 [1, 0, 0, 1, 0, 0, 0, 1, 1, 0]

B_2 [0, 0, 0, 0, 0, 0, 1, 0, 0, 1]

B_3 [0, 1, 1, 0, 1, 0, 0, 0, 0, 0]

B_4 [0, 0, 0, 1, 0, 1, 0, 0, 1, 0]

$$\begin{array}{c|cccccccccc|}
 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \\
B_1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
B_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
B_3 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
B_4 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}$$

Tossing Graphs Representation

The process of the creation of a bug tossing graph is explained with the help of an example here. Let us assume that there are 5 bug tossing paths:

$D_1 \rightarrow D_2 \rightarrow D_4 \rightarrow D_5$

$D_1 \rightarrow D_3 \rightarrow D_4$

$D_1 \rightarrow D_4$

$D_2 \rightarrow D_3 \rightarrow D_5$

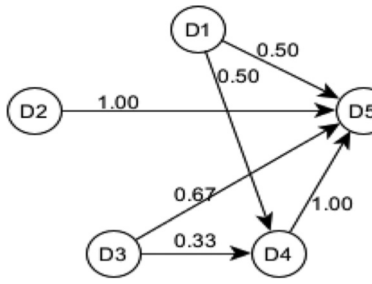

Fig. D1. Example of bug tossing graph.

$D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow D_5$

Then the actual computed paths for the developers are:

$D_1 \rightarrow D_2(2), D_1 \rightarrow D_3(2), D_1 \rightarrow D_4(3), D_1 \rightarrow D_5(2),$

$D_2 \rightarrow D_3(1), D_2 \rightarrow D_4(1), D_2 \rightarrow D_5(3),$

$D_3 \rightarrow D_4(1), D_3 \rightarrow D_5(2),$

$D_4 \rightarrow D_5(1)$

The goal-oriented paths for the developer D_1 are $D_1 \rightarrow D_2(2), D_1 \rightarrow D_5(2)$, so as per the Hidden Morkov Model (HMM) probability of fixing the bugs of D_1 with D_4 and D_5 is 50%. Similarly, the goal-oriented path for developer D_2 is $D_2 \rightarrow D_5(3)$, so the probability of fixing the bugs of D_2 with D_5 is 100%. The goal-oriented paths for developer D_3 are $D_3 \rightarrow D_4(1), D_3 \rightarrow D_5(2)$, hence the probability of fixing the bugs of D_3 with D_4 and D_5 are 33% and 67% respectively. The goal-oriented path for the developer D_4 is $D_4 \rightarrow D_5(1)$, so the probability of fixing the bugs of D_4 with D_5 is 100%. Based on this information, the tossing graph for the example tossing paths is generated and shown in Fig. D1.

Appendix E

Example Performance Parameter Calculation for BT

Let us assume that $K = 4$ and the Recommended developers are $D = \{1,\ 2,\ 3,\ 4\}$ and real assignee developers $R = \{1,\ 3\}$ then $P@1 = \frac{|\{1\}|}{|\{1\}|} = \frac{1}{1} = 1,\ P@2 = \frac{|\{1\}|}{|\{1,2\}|} = \frac{1}{2} = 0.5,\ P@3 = \frac{|\{1,3\}|}{|\{1,2,3\}|} = \frac{2}{3} =$ 0.67 and

$$P@4 = \frac{|\{1,3\}|}{|\{1,2,3,4\}|} = \frac{2}{4} = 0.5$$

Similarly,

$\mathrm{R}@1 = \frac{|\{1\}|}{|\{1,3\}|} = \frac{1}{2} = 0.5, \mathrm{R}@2 = \frac{|\{1\}|}{|\{1,3\}|} = \frac{1}{2} = 0.5, \mathrm{R}@3 = \frac{|\{1,3\}|}{|\{1,3\}|} = \frac{2}{2} =$ 1.0 and

$$\mathrm{R}@4 = \frac{|\{1,3\}|}{|\{1,3\}|} = \frac{2}{2} = 1.0$$

$$AP = \frac{1.0 + 0.67}{2} = 0.835$$

Appendix F

Strength, weakness, threats, and opportunity (SWOT) analysis of various AI architectures for software bug triaging along with a suggestive selection matrix.

Table F1
SWOT parameters.

Strength	**Weakness**
S1: Simple to model	W1: Higher computation time
S2: Less infrastructure	W2: Complex to model
S3: Availability of API	W3: Performance
Threats	**Opportunity**
T1: Availability of developers	O1: Incorporate additional information
T2: Prioritization	O2: Possibility of performance improvement and scalability
T3: Invalid and nonfiltered bugs	O3: Real-time triaging

SWOT Analysis of SBT Techniques

Each cluster of software bug triaging techniques has its own advantages and disadvantages. A basic SWOT (Strength, Weakness, Opportunities, and Threats) characteristic is listed by identifying the parameters of SWOT from the analysis of selected studies in the review work. Three main contributing parameters are identified from selected studies, and these parameters are generalized for doing a comparison of different types of software bug triaging techniques. The generalized parameters are tabulated in Table F1. After tabulating the generalized SWOT parameters, a mapping between the various categories of SBT techniques with these parameters is performed at Table F2.

Table F4
Description of software bug triaging architecture symbols.

BT[e] Type	Description
ML	Machine learning-based techniques
IR	Information retrieval-based techniques
SNA	Crowdsourcing and social networking-based models
RS	Recommender systems and ranking of developers
MMO	Mathematical modeling and optimization-based techniques
DL	Deep learning-based techniques

[e] Bug triaging techniques.

Identification of SWOT Parameters

The strength parameters of SWOT analysis are identified by generalizing the characteristics from the clustered studies included in each SBT technique category. The identified parameters are namely, simple to model, computing infrastructure requirements, and availability of programming languages for the implementation of the techniques. From the SWOT parameters mapping Table F2, it is presented that for machine learning and information retrieval based SBT techniques all the three strengths are applicable as for both techniques modeling is simple, both requires less computing infrastructure and good programming support, and API is available for implementation (Alazzam et al., 2020; Alkhazi et al., 2020; Alonso-Abad et al., 2019; Ardimento et al., 2020; Banerjee et al., 2017; Florea et al., 2017b; Hindle et al., 2016; D. Hu et al., 2018; Jiang et al., 2019; Kagdi et al., 2012; Kukkar et al., 2019; J. Lee et al., 2019; Linares-Vasquez et al., 2012; Mohsin & Shi, 2020; Murphy & Cubranic, 2004; Resketi et al., 2020; Sajedi-Badashian & Stroulia, 2020b; Sharma et al., 2017; Shokripour et al., 2013, 2014, 2015; S. Q. Xi et al., 2019; Xia et al., 2016; Xuan et al., 2014; Yin et al., 2018). For SNA-based (Alazzam et al., 2020; Matsoukas et al., 2020; Sajedi-Badashian & Stroulia, 2020b; Zhang et al., 2017; W. Zhang et al., 2016a, 2016b) and MMO-based techniques (A. Goyal & Sardana, 2017c; Kashiwa & Ohira, 2020; Kumar et al., 2020a, Kumar et al., 2020b; J. Liu et al., 2016; Tamrawi et al., 2011a, 2011b; Wei et al., 2018; W. Zhang et al., 2017) computing infrastructure requirements are less so it is common strength to both techniques, but modeling and API support are totally logic dependent so that is the major problem in both techniques. For RS-based technique (Anvik & Murphy, 2011; Chamoso, Hernández, González-Briones, & García-Peñalvo, 2020; Chen et al., 2019; Florea et al., 2017b, 2017a; Jindal & Kaur, 2020; Kanwal & Maqbool, 2012; Panda & Nagwani, 2021; Vu et al., 2020;

Table F2
SWOT parameter mapping with bug triaging techniques.

Techniques	Strength			Weakness			Threats			Opportunity		
	S1	**S2**	**S3**	**W1**	**W2**	**W3**	**T1**	**T2**	**T3**	**O1**	**O2**	**O3**
ML	√	√	√	✗	✗	√	√	✗	✗	√	√	√
IR	√	√	√	✗	✗	√	√	√	✗	√	√	√
SNA	✗	√	✗	√	√	√	√	√	√	√	√	✗
RS	✗	√	√	√	√	✗	✗	√	✗	√	√	✗
MMO	✗	√	✗	✗	√	√	√	√	√	√	√	✗
DL	✗	✗	√	√	√	✗	√	√	√	√	√	√

Table F3
An indicative/suggestive matrix for selecting a bug triaging architecture.

Case	ML	IR	SNA	RS	MMO	DL
Time constraints	√	√	✗	✗	√	✗
Performance	✗	✗	✗	√	✗	√
Prioritization	√	√	✗	√	√	✗
Model enhancement	√	√	√	√	√	√
Large repositories	√	√	√	√	√	√

Xia et al., 2015a; Xie et al., 2012; Yadav et al., 2019; Zaidi et al., 2020) computing infrastructure requirement is less and API support is good, but modeling is a key challenge for effective implementation of software bug triaging. For DL-based SBT techniques (Florea et al., 2017a; S. Guo et al., 2020; Koc et al., 2019; S.-R. Lee et al., 2017; Mani et al., 2019; S. S. Xi et al., 2018; S. Q. S. Xi et al., 2018; Ye et al., 2020; Zaidi et al., 2020), there is a good support of programming languages from an implementation point of view, for example, Python provides a good set of libraries for implementation, but DL-based techniques need high computing infrastructures and modeling the SBT problem is a complex task as it involves the multi-layer dense network for processing of logic and optimizing the weights for better performance.

Weakness parameters are derived from the poor-quality component of a particular SBT technique. Three main components listed under weakness are higher computation time, complexity to model, and performance of the software bug triaging techniques. ML-based and IR-based techniques are faster and simple to model for SBT tasks they only have the limitation of performance, although performance is acceptable still there is a scope for further improvements on it (Florea et al., 2017b; Mohsin & Shi, 2020; Sajedi-Badashian & Stroulia, 2020b). SNA-based techniques suffer from all the three weaknesses identified as the modeling as SNA consumes the relationship between the software bugs and developers network information for modeling of SBT problem, it also has higher computation time as graph data structures are used for discovering the knowledge that creates the complexity for these SBT techniques (Alazzam et al., 2020; Sajedi-Badashian & Stroulia, 2020b; Zhang et al., 2017). Both RS-based (Anvik & Murphy, 2011; Jindal & Kaur, 2020; Kanwal & Maqbool, 2012; Panda & Nagwani, 2021; Vu et al., 2020; Xia et al., 2015a; Yadav et al., 2019) and DL-based techniques (Florea et al., 2017a; S.-R. Lee et al., 2017; S. S. Xi et al., 2018; S. Q. S. Xi et al., 2018; Ye et al., 2020) give a comitative performance as compared to the other techniques, but the model building is complex in both techniques. MMO-based techniques(Kashiwa & Ohira, 2020; Kumar et al., 2020a, Kumar et al., 2020b; Wei et al., 2018) are driven by mathematical modeling so complexity in model building, and performance is the key challenge in these techniques, however, the computational time is acceptable in the available MMO-based studies of software bug triaging.

Opportunity parameters in SWOT analysis include incorporating the additional information, performance improvement and scalability, and scope of real-time triaging of the existing works. Incorporating other information means deriving additional information (generating additional features) from the existing software bug datasets which can be utilized for improving the existing techniques of software bug triaging. Both ML-based and IR-based techniques (Alazzam et al., 2020; Alkhazi et al., 2020; Alonso-Abad et al., 2019; Ardimento et al., 2020; Banerjee et al., 2017; Hindle et al., 2016; D. Hu et al., 2018; Jiang et al., 2019; Linares-Vasquez et al., 2012; Resketi et al., 2020; Shokripour et al., 2013; Xia et al., 2016; Xuan et al., 2014; Yin et al., 2018) have all three opportunities where the performance of the existing techniques can be improved the existing techniques can be transformed to scalable and can accommodate large bug repositories for real-world projects. Rest all four SBT-techniques have the common opportunity characteristics, i.e., rest all can consider generating and utilizing the additional features for improving the performance, but the scope of performance is limited due to the technology limitation and modeling building complexity. DL-based techniques (Florea et al., 2017a; Lee et al., 2017; S. Xi et al., 2018; S. Xi et al., 2018; Ye et al., 2020; Zaidi et al., 2020) are already giving a better performance as compared to other SBT-techniques so there is less opportunity for further drastic improvement of the existing techniques.

The threats parameters consist of considering the availability of software developers, handling capabilities of priority bugs (bug prioritization), and handling of invalid and nonfiltered software bugs. Most of the ML-based studies have focused primarily on identifying expert developers but very few studies are presented where the availability and workload of the identified developers are considered for generating the output for software bug triaging (Sawarkar et al., 2019). Only a few studies are available in ML-based techniques which consider the bug prioritization or filtering of invalid bugs (Banerjee et al., 2017; A. Goyal & Sardana, 2017a; N. Goyal et al., 2015; Jiechieu & Tsopze, 2020; Kagdi et al., 2012; J. Lee et al., 2019; Linares-Vasquez et al., 2012; Mostafa et al., 2021; Murphy & Cubranic, 2004; Sharma et al., 2017; Shokripour et al., 2013, 2014, 2015; Uddin et al., 2017; W. Wu et al., 2011). A few studies focus on filtering invalid bugs in IR-based SBT techniques but very few studies are presented where the availability of existing developers and bug prioritization is considered (N. Goyal et al., 2015; Kanwal & Maqbool, 2012; Mostafa et al., 2021; Uddin et al., 2017). Although a few studies under RS-based techniques have considered the availability of expert developers and handling the invalid bugs, but very few works are reported in the bug prioritization for software bug triaging. Technically, computing the availability of developers is comparatively easy in RS-based techniques as these techniques generate the ranked list of developer's and inactive developers can be filtered easily from the ranked list. SNA-based, MMO-based, and DL-based techniques are suffering from all the three threats identified as it is observed from the existing studies that very limited studies are presented from these categories of SBT where researchers have considered the scenarios of availability of developers for fixing the newly reported bugs, bug prioritization, and handling of invalid bugs coming as input from bug repositories (Banerjee et al., 2017; A. Goyal & Sardana, 2017a; N. Goyal et al., 2015; Jiechieu & Tsopze, 2020; Kagdi et al., 2012; J. Lee et al., 2019; Linares-Vasquez et al., 2012; Mostafa et al., 2021; Murphy & Cubranic, 2004; Sharma et al., 2017; Shokripour et al., 2013, 2014, 2015; Uddin et al., 2017; W. Wu et al., 2011).

Mapping of SBT Technique with SWOT Parameters

Machine learning techniques, being the most popular techniques in the area of software bug triaging techniques, have several advantages and disadvantages. ML-based techniques have all three strengths filtered in SWOT criteria, that is, it is simple to model, needs moderate computing infrastructure, and is supported by many programming languages with good collections of APIs (Application Programming Interface). The performance of ML-based techniques is generally less as compared to the DL-based techniques, so the weakness component performance is mapped (Florea et al., 2017b; Kukkar et al., 2019; Mohsin & Shi, 2020; Xi et al., 2019). Most of the studies included generally do not consider the availability of the developers so it is mapped in the treats section. ML-based techniques have the capabilities of further improvements so by considering other relevant additional information performance can be improved and at the same time, ML-based techniques are scalable (Jiechieu & Tsopze, 2020; Lee et al., 2019; Murphy & Cubranic, 2004; Sharma et al., 2017), so it handles the large bug repositories so in the opportunities section ML-based techniques are fit under all the three criteria as mentioned in the Table F2.

IR-based techniques have a similar set of strengths, weaknesses, opportunities, and threats with ML-based techniques (Jiechieu & Tsopze, 2020; J. Lee et al., 2019; Murphy & Cubranic, 2004; Sharma et al., 2017). The only difference observed from the SWOT analysis is that for IR-based techniques less work is presented in the earlier studies focusing on the availability of developers and bug prioritization. Just like ML-based techniques, the strength of IR-based techniques is it is simple to model, take less computing infrastructure, and there is a good API availability for implementation of IR-based techniques (Alazzam et al., 2020; Alkhazi et al., 2020; Alonso-Abad et al., 2019; Ardimento et al., 2020; Hindle et al., 2016; D. Hu et al., 2018; Jiang et al., 2019; Kagdi et al., 2012; Linares-Vasquez et al., 2012; Shokripour et al., 2015; Xia et al., 2017). Performance is the main weakness and there is the same can be improved as shown in the opportunity column by considering the additional information and model rearrangements (Alazzam et al., 2020; Alkhazi et al., 2020; Kukkar et al., 2019). So, performance improvement and scalability are possible in IR-based techniques and there is a scope of achieving the real-time triaging using these techniques.

From the studies included in the review, it is observed that the strength of SNA-based techniques (Alazzam et al., 2020; Matsoukas et al., 2020; Sajedi-Badashian & Stroulia, 2020b; Xuan et al., 2017b; Zhang et al., 2017; Ye et al., 2020; Zhang et al., 2016a) is the implementation of these techniques requires less computing infrastructure. Most of the studies are presented in general computing infrastructure. All the studies suffer from the three weakness criteria, that are, computation time is higher, model building is complicated, and performance achieved is less as compared to ML-based, IR-based, or DL-based SBT techniques. Major threats include the consideration of the availability of developers, bug prioritization, and handling of invalid and nonfiltered bugs. Opportunity for MMO-based SBT techniques includes incorporating additional information for further improvements on performance improvement and scalability. Real-time triaging is still a challenge with these techniques as applying the relationship using complex data structures at the run time might not be feasible.

The primary strength of RS-based software bug triaging techniques (Jindal & Kaur, 2020; Panda & Nagwani, 2021; Vu et al., 2020; Xia et al., 2015a; Yadav et al., 2019) is, it takes less computing infrastructure and is backed up by good support of programming languages and API availability for implementing recommender systems for software bug triaging. Weakness includes computational time is higher as compared to ML-based and IR-based SBT techniques. The main threat in RS-based techniques is that less work considering the bug prioritization is presented in the previous studies. RS-based techniques are developed for the real-world scenario of assigning the bugs to the developers so there is a chance of performance improvements by considering the additional information.

MMO-based techniques (Kashiwa & Ohira, 2020; Kumar et al., 2020a, Kumar et al., 2020b; J. Liu et al., 2016; Wei et al., 2018; W. Zhang et al., 2017) need less computing infrastructure that is the key strength of these techniques. The main weakness of MMO-based techniques is model building complexity and performance presented in the earlier studies. MMO-based SBT techniques need a lot of attention in mathematical formulation and setting up models for addressing the bug triaging tasks, if anything goes wrong during modeling then it can badly impact the efficacy of the bug triaging and can further lead to a negative impact on the software development cost. MMO-based SBT techniques have all three threats identified, i.e., consideration of the availability of developers, handling priority bugs, and handling of invalid and nonfiltered bugs during the software bug triaging process. The opportunities in MMO-based techniques are there is the scope of performance modeling and making these techniques scalable by considering the additional information and features during mathematical modeling of the bug triaging techniques.

DL-based techniques (S. Guo et al., 2020; Koc et al., 2019; Mani et al., 2019; S. S. Xi et al., 2018; S. Q. S. Xi et al., 2018; Ye et al., 2020; Zaidi et al., 2020) have the advantages of programming support and availability of APIs (Application Programming Interfaces), but the model building is complicated and computing infrastructure requirements are higher for these techniques. So, the key strength of these techniques is the availability of API and higher performance, but the weakness of these techniques is complicated model building and higher computational time. The threats under DL-based techniques are consideration of the availability of developers, bug prioritization, and handling of invalid bugs. The opportunity in DL-based techniques is considering additional information and the further chance of performance improvement. Another key opportunity is the development of intelligent techniques for real-time triaging using DL-based techniques, as deep learning techniques have the capability of achieving maximum performance.

After SWOT analysis, an indicative/suggestive matrix for selecting a suitable bug triaging technique is also derived.

Indicator Matrix for Selecting a SBT Technique

The selection criteria of a SBT technique are derived from the characteristics, advantages, and disadvantages of all the individual software bug triaging technique clusters. Five main important criteria for the selection of suitable bug triaging techniques are derived, namely, time constraints, performance, prioritization, model enhancement capabilities, and handling of large bug repositories.

Time constraints indicate that if the end-user (bug triager) has time constraints in resolving the bugs with moderate computation resources, then the best suitable choice of techniques can be ML-based, IR-based, or MMO-based bug triaging techniques. It is derived from the computing infrastructure and complexities listed in the categorized selected studies in software bug triaging. SNA-based, RS-based, and DL-based techniques take a long time in software bug triaging as compared with the ML-based, IR-based, and MMO-based software bug triaging techniques.

Performance is derived from the values of the performance parameters discussed in the studies included in the review. As discussed earlier, most of the studies (Han et al., 2011; Sajedi-Badashian & Stroulia, 2020a) have used the performance parameters namely, accuracy, precision, recall, and MAP (Mean Average Precision) or their ranked version i.e. accuracy@K, precision@K, and recall@K. So, from the maximum performance point of view, DL-based techniques are most suitable as compared to any other SBT techniques, and after the DL-based technique, RS-based techniques give better performance in general. A few studies in ML-based and IR-based techniques also mentioned better performance but in a generalized manner and from the entire category comparison point of view DL-based techniques and RS-based techniques are most suitable in the case when maximum desirable performance is expected.

The next important criteria listed from the existing studies are capabilities of handling bug prioritization (N. Goyal et al., 2015; Kanwal & Maqbool, 2012; Mostafa et al., 2021; Uddin et al., 2017). Bug prioritization refers to the scenario where if any security bug or high priority is reported, then its resolution (fixing) must be prioritized as it requires quick attention from the entire software development team to avoid critical damage to the software or reputation of the IT (Information Technology) firm. From the analysis of the studies included in the review, it is observed that more work is reported on ML-based, IR-based, RS-based, and MMO-based bug triaging techniques for the handling of bug prioritization and fewer studies are presented on SNA-based and DL-based triaging techniques, although all these techniques can handle it.

The last two parameters in the selection criteria are identified from the conclusion and future scope section of the studies included in the review. These two criteria are model enhancement capabilities and handling of large bug repositories. Both criteria are discussed in almost all the studies since in the Big Data era and from adoption of these techniques in the real-world scenario, these techniques should be scalable enough and there must be the scope for further improvement on performance for live projects. So, from the analysis of all the SBT techniques, it is observed that almost all the existing techniques (Alazzam et al., 2020; Alkhazi et al., 2020; Ardimento et al., 2020; Jiang et al., 2019; Jiechieu & Tsopze, 2020; J. Lee et al., 2019; Murphy & Cubranic, 2004; Sharma et al., 2017; Xia et al., 2016; Xuan et al., 2014; Zhang et al., 2017; Yin et al., 2018) have suggested good guidelines in the future scope of the work about enhancement of their work along with accommodating the large bug repositories. Both criteria are documented in the opportunities and future scope of the software bug triaging.

References

Ahsan, S. N., Ferzund, J., & Wotawa, F. (2009). Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *2009 fourth international conference on software engineering advances* (pp. 216–221).

Akgün, A. E. (2020). Team wisdom in software development projects and its impact on project performance. *International Journal of Information Management, 50*, 228–243. 10.1016/j.ijinfomgt.2019.05.019.

Akila, V., Govindasamy, V., & Sharmila, S. (2016). Bug Triage based on ant system with evaporation factor tuning. *International Journal of Control Theory and Applications, 9*(2), 859–863.

Akila, V., Zayaraz, G., & Govindasamy, V. (2015). Bug triaging based on ant systems. *International Journal of Bio-Inspired Computation, 7*(4), 263–268. 10.1504/IJBIC.2015.071078.

Aktas, E. U., & Yilmaz, C. (2020). Automated issue assignment: Results and insights from an industrial case. *Empirical Software Engineering, 25*(5), 3544–3589. 10.1007/s10664-020-09846-3.

Alazzam, I., Aleroud, A., Al Latifah, Z., & Karabatis, G. (2020). Automatic bug triage in software systems using graph neighborhood relations for feature augmentation. *IEEE Transactions on Computational Social Systems, 7*(5), 1288–1303. 10.1109/TCSS.2020.3017501.

Alenezi, M., Magel, K., & Banitaan, S. (2013). Efficient bug triaging using text mining. *Journal of Software, 8*(9), 2185–2190. 10.4304/jsw.8.9.2185-2190.

Alkhazi, B., DiStasi, A., Aljedaani, W., Alrubaye, H., Ye, X., & Mkaouer, M. W. (2020). Learning to rank developers for bug report assignment. *Applied Soft Computing Journal, 95*. 10.1016/j.asoc.2020.106667.

Almhana, R., & Kessentini, M. (2021). Considering dependencies between bug reports to improve bugs triage. *Automated Software Engineering, 28*(1), 1–26.

Alonso-Abad, J. M., López-Nozal, C., Maudes-Raedo, J. M., & Marticorena-Sánchez, R. (2019). Label prediction on issue tracking systems using text mining. *Progress in Artificial Intelligence, 8*(3), 325–342. 10.1007/s13748-019-00182-2.

Antunes, A. L., Cardoso, E., & Barateiro, J. (2022). Incorporation of ontologies in data warehouse/business intelligence systems—a systematic literature review. *International Journal of Information Management Data Insights, 2*(2), Article 100131. 10.1016/j.jjimei.2022.100131.

Anvik, J. (2006). Automating bug report assignment. In *Proceedings of the 28th international conference on software engineering* (pp. 937–940).

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th international conference on software engineering* (pp. 361–370).

Anvik, J., & Murphy, G. C. (2011). Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology, 20*(3). 10.1145/2000791.2000794.

Ardimento, P., Boffoli, N., & Mele, C. (2020). *A Text-Based Regression Approach to Predict Bug-Fix Time* (Vol. 880).

Ashok, M., Madan, R., Joha, A., & Sivarajah, U. (2022). Ethical framework for artificial intelligence and digital technologies. *International Journal of Information Management, 62*, Article 102433.

Banerjee, S., Syed, Z., Helmick, J., Culp, M., Ryan, K., & Cukic, B. (2017). Automated triaging of very large bug repositories. *Information and Software Technology, 89*, 1–13. 10.1016/j.infsof.2016.09.006.

Baysal, O., Godfrey, M. W., & Cohen, R. (2009). A bug you like: A framework for automated assignment of bugs. In *2009 IEEE 17th international conference on program comprehension* (pp. 297–298).

Bejjanki, K. K., Gyani, J., & Gugulothu, N. (2020). Class imbalance reduction (CIR): A novel approach to software defect prediction in the presence of class imbalance. *Symmetry, 12*(3), 407.

Bhattacharya, P., & Neamtiu, I. (2010a). Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. *IEEE international conference on software maintenance, ICSM*. 10.1109/ICSM.2010.5609736.

Bhattacharya, P., & Neamtiu, I. (2010b). Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. *IEEE international conference on software maintenance, ICSM*. 10.1109/ICSM.2010.5609736.

Bhattacharya, P., Neamtiu, I., & Shelton, C. R. (2012). Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software, 85*(10), 2275–2292. 10.1016/j.jss.2012.04.053.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research, 3*, 993–1022.

Boehmke, B., Hazen, B., Boone, C. A., & Robinson, J. L. (2020). A data science and open source software approach to analytics for strategic sourcing. *International Journal of Information Management, 54*, Article 102167.

Borges, A. F. S., Laurindo, F. J. B., Spínola, M. M., Gonçalves, R. F., & Mattos, C. A. (2021). The strategic use of artificial intelligence in the digital era: Systematic literature review and future research directions. *International Journal of Information Management, 57*, Article 102225. 10.1016/j.ijinfomgt.2020.102225.

Brookes, G., & McEnery, T. (2019). The utility of topic modelling for discourse studies: A critical evaluation. *Discourse Studies, 21*(1), 3–21.

Campos, E. C., Maia, M., & de, A. (2019). Discovering common bug-fix patterns: A large-scale observational study. *Journal of Software: Evolution and Process, 31*(7), e2173.

Cavalcanti, Y. C., MacHado, I. D. C., Neto, P. A. D. M. S., & Almeida, E. S. D. (2016). Towards semi-automated assignment of software change requests. *Journal of Systems and Software, 115*, 82–101. 10.1016/j.jss.2016.01.038.

Chamoso, P., Hernández, G., González-Briones, A., & García-Peñalvo, F. J. (2020). Recommendation of technological profiles to collaborate in software projects using document embeddings. *Neural Computing and Applications, 34*, 8423–8430. 10.1007/s00521-020-05522-1.

Chen, L., Wang, X., & Liu, C. (2010). Improving bug assignment with bug tossing graphs and bug similarities. *2010 international conference on biomedical engineering and computer science, ICBECS 2010*. 10.1109/ICBECS.2010.5462287.

Chen, L., Wang, X., & Liu, C. (2011). An approach to improving bug assignment with bug tossing graphs and bug similarities. *Journal of Software, 6*(3), 421–427. 10.4304/jsw.6.3.421-427.

Chen, M., Hu, D., Wang, T., Long, J., Yin, G., Yu, Y., & Zhang, Y. (2019). *Using document embedding techniques for similar bug reports recommendation. 2018-November*, 811–814. https://doi.org/10.1109/ICSESS.2018.8663849

Chen, R., Guo, S.-. K., Wang, X.-. Z., & Zhang, T.-. L. (2019b). Fusion of multi-RSMOTE with fuzzy integral to classify bug reports with an imbalanced distribution. *IEEE Transactions on Fuzzy Systems, 27*(12), 2406–2420.

Choquette-Choo, C. A., Sheldon, D., Proppe, J., Alphonso-Gibbs, J., & Gupta, H. (2019). A multi-label, dual-output deep neural network for automated bug triaging. In *IEEE international conference on machine learning and applications* (pp. 937–944). 10.1109/ICMLA.2019.00161.

Collins, C., Dennehy, D., Conboy, K., & Mikalef, P. (2021). Artificial intelligence in information systems research: A systematic literature review and research agenda. *International Journal of Information Management, 60*, Article 102383.

Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., & Larrucea, X. (2018). A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management, 40*, 186–189. 10.1016/j.ijinfomgt.2017.11.005.

Cui, Y., Jia, M., Lin, T.-. Y., Song, Y., & Belongie, S. (2019). Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 9268–9277).

da Silva, M. C., Cizotto, A. A. J., & Paraiso, E. C. (2020). A developer recommendation method based on code quality. In *IEEE 2020 international joint conference on neural networks (IJCNN* (pp. 1–8). 1728169267.

Deng, T., & Robinson, W. N. (2021). Changes in emergent software development routines: The moderation effects of routine diversity. *International Journal of Information Management, 58*, Article 102306. 10.1016/j.ijinfomgt.2020.102306.

Fan, Y., Xia, X., Lo, D., & Hassan, A. E. (2020). Chaff from the wheat: Characterizing and determining valid bug reports. *IEEE Transactions on Software Engineering, 46*(5), 495–525. 10.1109/TSE.2018.2864217.

Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., & Zhang, M. (2021). COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. *Information and Software Technology, 129*, Article 106432.

Florea, A. C., Anvik, J., & Andonie, R. (2017a). Parallel implementation of a bug report assignment recommender using deep learning. In *International conference on artificial neural networks* (pp. 64–71). Vol. 10614 LNCS.

Florea, A. C., Anvik, J., & Andonie, R. (2017b). Spark-based cluster implementation of a bug report assignment recommender system. In *International conference on artificial intelligence and soft computing* (pp. 31–42). Vol. 10246 LNAI.

Gong, L., Jiang, S., & Jiang, L. (2019). Tackling class imbalance problem in software defect prediction through cluster-based over-sampling with filtering. *IEEE Access : Practical Innovations, Open Solutions, 7*, 145725–145737.

Goyal, A., & Sardana, N. (2017a). Machine learning or information retrieval techniques for bug triaging: Which is better? *E-Informatica Software Engineering Journal, 11*(1), 117–141. 10.5277/e-Inf170106.

Goyal, A., & Sardana, N. (2017b). NRFixer: Sentiment based model for predicting the fixability of non-reproducible bugs. *E-Informatica Software Engineering Journal, 11*(1) Art. 1.. 10.5277/e-Inf170105.

Goyal, A., & Sardana, N. (2017c). Optimizing bug report assignment using multi criteria decision making technique. *Intelligent Decision Technologies, 11*(3) Art. 3. 10.3233/IDT-170297.

Goyal, A., & Sardana, N. (2019a). An empirical study of non-reproducible bugs. *International Journal of System Assurance Engineering and Management, 10*(5), 1186–1220. 10.1007/s13198-019-00850-5.

Goyal, A., & Sardana, N. (2019b). Empirical analysis of ensemble machine learning techniques for bug triaging. In *2019 twelfth international conference on contemporary computing (IC3)* (pp. 1–6).

Goyal, N., Aggarwal, N., & Dutta, M. (2015). A novel way of assigning software bug priority using supervised classification on clustered bugs data. In *Advances in intelligent informatics* (pp. 493–501). Springer.

Guo, P.J., Zimmermann, T., Nagappan, N., & Murphy, B. (2011). *Not my bug! And other reasons for software bug report reassignments*. 395–404. https://doi.org/10.1145/1958824.1958887

Goyal, S. (2022). Handling class-imbalance with KNN (Neighbourhood) under-sampling for software defect prediction. *Artificial Intelligence Review, 55*, 2023–2064.

Guo, S., Zhang, X., Yang, X., Chen, R., Guo, C., Li, H., & Li, T. (2020). Developer activity motivated bug triaging: Via convolutional neural network. *Neural Processing Letters, 51*(3), 2589–2606. 10.1007/s11063-020-10213-y.

Gupta, C., & Freire, M. M. (2021). A decentralized blockchain oriented framework for automated bug assignment. *Information and Software Technology, 134* Scopus. 10.1016/j.infsof.2021.106540.

Gupta, C., Inácio, P. R. M., & Freire, M. M. (2021). Improving software maintenance with improved bug triaging. *Journal of King Saud University - Computer and Information Sciences* Scopus. 10.1016/j.jksuci.2021.10.011.

Han, J., Kamber, M., & Pei, J. (2011). Data mining concepts and techniques third edition. *The Morgan Kaufmann Series in Data Management Systems, 5*(4), 83–124.

Herath, H. M. K. K. M. B., & Mittal, M. (2022). Adoption of artificial intelligence in smart cities: A comprehensive review. *International Journal of Information Management Data Insights, 2*(1), Article 100076. 10.1016/j.jjimei.2022.100076.

Higgins, J. P., Altman, D. G., Gøtzsche, P. C., Jüni, P., Moher, D., Oxman, A. D., Savović, J., Schulz, K. F., Weeks, L., & Sterne,, J. A. (2011). The cochrane collaboration's tool for assessing risk of bias in randomised trials. *BMJ (Clinical Research ed.), 343*, d5928.

Hindle, A., Alipour, A., & Stroulia, E. (2016). A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering, 21*(2), 368–410. 10.1007/s10664-015-9387-3.

Hu, D., Chen, M., Wang, T., Chang, J., Yin, G., Yu, Y., & Zhang, Y. (2018). *Recommending similar bug reports: A novel approach using document embedding model. 2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 725–726. 10.1109/APSEC.2018.00108.

Hu, H., Zhang, H., Xuan, J., & Sun, W. (2014). Effective bug triage based on historical bug-fix information. In *2014 IEEE 25th international symposium on software reliability engineering* (pp. 122–132).

Huang, J., & Ma, Y. (2019). Predicting the fixer of software bugs via a collaborative multiplex network: Two case studies. In *International conference on collaborative computing: networking, applications and worksharing* (pp. 469–488).

Huang, Y., Jia, N., Zhou, H.-. J., Chen, X.-. P., Zheng, Z.-. B., & Tang, M.-. D. (2020). Learning human-written commit messages to document code changes. *Journal of Computer Science and Technology, 35*(6), 1258–1277. 10.1007/s11390-020-0496-0.

Jena, B., Saxena, S., Nayak, G. K., Saba, L., Sharma, N., & Suri, J. S. (2021). Artificial intelligence-based hybrid deep learning models for image classification: The first narrative review. *Computers in Biology and Medicine, 137*, 104803.

Jeong, G., Kim, S., & Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering* (pp. 111–120). 10.1145/1595696.1595715.

Jiang, J., Lo, D., Zheng, J., Xia, X., Yang, Y., & Zhang, L. (2019). Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction. *Journal of Systems Software, 154*, 196–210.

Jiechieu, K. F. F., & Tsopze, N. (2020). Skills prediction based on multi-label resume classification using CNN with model predictions explanation. *Neural Computing and Applications*. 10.1007/s00521-020-05302-x.

Jindal, S. G., & Kaur, A. (2020). Automatic keyword and sentence-based text summarization for software bug reports. *IEEE Access : Practical Innovations, Open Solutions, 8*, 65352–65370.

Kagdi, H., Collard, M. L., & Maletic, J. I. (2007). A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice, 19*(2), 77–131.

Kagdi, H., Gethers, M., Poshyvanyk, D., & Hammad, M. (2012). Assigning change requests to software developers. *Journal of Software: Evolution and Process, 24*(1), 3–33.

Kalliamvakou, E., Bird, C., Zimmermann, T., Begel, A., DeLine, R., & German, D. M. (2019). What MAKES A GREAT MANAGER OF SOFTWARE Engineers? *IEEE Transactions on Software Engineering, 45*(1), 87–106. 10.1109/tse.2017.2768368.

Kanwal, J., & Maqbool, O. (2012). Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology, 27*(2), 397–412.

Kar, A. K., Choudhary, S. K., & Singh, V. K. (2022). How can artificial intelligence impact sustainability: A systematic literature review. *Journal of Cleaner Production*, Article 134120.

Kar, A. K., & Kushwaha, A. K. (2021). Facilitators and barriers of artificial intelligence adoption in business–insights from opinions using big data analytics. *Information Systems Frontiers, 2021*, 1–24.

Kashiwa, Y. (2019). RAPTOR: Release-aware and prioritized bug-fixing task assignment optimization. In *2019 IEEE international conference on software maintenance and evolution (ICSME)* (pp. 629–633).

Kashiwa, Y., & Ohira, M. (2020). A release-aware bug triaging method considering developers' bug-fixing loads. *IEICE Transactions on Information and Systems, E103D*(2), 348–362. 10.1587/transinf.2019EDP7152.

Kaur, A., & Jindal, S. G. (2019). Text analytics based severity prediction of software bugs for apache projects. *International Journal of Systems Assurance Engineering and Management, 10*(4), 765–782. 10.1007/s13198-019-00807-8.

Ketler, K., & Turban, E. (1992). Productivity improvements in software maintenance. *International Journal of Information Management, 12*(1), 70–82. 10.1016/0268-4012(92)90053-S.

Kim, Y., Kim, J., Jeon, H., Kim, Y. H., Song, H., Kim, B., & Seo, J. (2020). Githru: Visual analytics for understanding software development history through git metadata analysis. *IEEE Transactions on Visualization and Computer Graphics*. 10.1109/TVCG.2020.3030414.

Koc, U., Wei, S., Foster, J.S., Carpuat, M., & Porter, A.A. (2019). *An empirical assessment of machine learning approaches for triaging reports of a Java static analysis tool*. 288–299. https://doi.org/10.1109/ICST.2019.00036

Kukkar, A., Mohana, R., Nayyar, A., Kim, J., Kang, B. G., & Chilamkurti, N. (2019). A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting. *Sensors (Switzerland), 19*(13), 1–22. 10.3390/s19132964.

Kumar, L., Tummalapalli, S., & Murthy, L. B. (2020a). An empirical framework to investigate the impact of bug fixing on internal quality attributes. *Arabian Journal for Science and Engineering*. 10.1007/s13369-020-05095-0.

Kumar, R., Khan, A. I., Abushark, Y. B., Alam, M. M., Agrawal, A., & Khan, R. A. (2020b). A knowledge-based integrated system of hesitant fuzzy set, AHP and TOPSIS for evaluating security-durability of web applications. *IEEE Access: Practical Innovations, Open Solutions, 8*, 48870–48885. 10.1109/access.2020.2978038.

Kumar, S., Kar, A. K., & Ilavarasan, P. V. (2021). Applications of text mining in services management: A systematic literature review. *International Journal of Information Management Data Insights, 1*(1), Article 100008. 10.1016/j.jjimei.2021.100008.

Kushwaha, A. K., Kar, A. K., & Dwivedi, Y. K. (2021). Applications of big data in emerging management disciplines: A literature review using text mining. *International Journal of Information Management Data Insights, 1*(2), Article 100017. 10.1016/j.jjimei.2021.100017.

Lee, D. G., & Seo, Y. S. (2020). Improving bug report triage performance using artificial intelligence based document generation model. *Human-Centric Computing and Information Sciences, 10*(1). 10.1186/s13673-020-00229-7.

Lee, J., Kim, D., & Jung, W. (2019). Cost-aware clustering of bug reports by using a genetic algorithm. *Journal of Information Science and Engineering, 35*(1), 175–200. 10.6688/JISE.201901_35(1).0010.

Lee, J.-. B., Lee, T., & In, H. P. (2020). Topic modeling based warning prioritization from change sets of software repository. *Journal of Computer Science and Technology, 35*(6), 1461–1479. 10.1007/s11390-020-0047-8.

Lee, S.-. R., Heo, M.-. J., Lee, C.-. G., Kim, M., & Jeong, G. (2017). Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering* (pp. 926–931).

Li, P. L., Ko, A. J., & Begel, A. (2019). What distinguishes great software engineers? *Empirical Software Engineering, 25*(1), 322–352. 10.1007/s10664-019-09773-y.

Linares-Vasquez, M., Hossen, K., Dang, H., Kagdi, H., Gethers, M., & Poshyvanyk, D. (2012). *Triaging incoming change requests: Bug or commit history, or code authorship?* 451–460. https://doi.org/10.1109/ICSM.2012.6405306

Liu, J., Tian, Y., Yu, X., Yang, Z., Jia, X., Ma, C., & Xu, Z. (2016). A multi-source approach for bug triage. *International Journal of Software Engineering and Knowledge Engineering, 26*(9–10), 1593–1604. 10.1142/S0218194016710030.

Liu, Y., Qi, X., Zhang, J., Li, H., Ge, X., & Ai, J. (2022). Automatic bug triaging via deep reinforcement learning. *Applied Sciences (Switzerland), 12*(7) Scopus. 10.3390/app12073565.

Mahdi, S. S., Battineni, G., Khawaja, M., Allana, R., Siddiqui, M. K., & Agha, D. (2023). How does artificial intelligence impact digital healthcare initiatives? A review of AI applications in dental healthcare. *International Journal of Information Management Data Insights, 3*(1), Article 100144. 10.1016/j.jjimei.2022.100144.

Mani, S., Sankaran, A., & Aralikatte, R. (2019). Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In *Proceedings of the ACM India joint international conference on data science and management of data* (pp. 171–179). 10.1145/3297001.3297023.

Mao, K., Yang, Y., Wang, Q., Jia, Y., & Harman, M. (2015). Developer recommendation for crowdsourced software development tasks. In *2015 IEEE symposium on service-oriented system engineering* (pp. 347–356).

Martínez-García, J. R., Castillo-Barrera, F.-. E., Palacio, R. R., Borrego, G., & Cuevas-Tello, J. C. (2020). Ontology for knowledge condensation to support expertise location in the code phase during software development process. *IET Software, 14*(3), 234–241. 10.1049/iet-sen.2019.0272.

Matsoukas, V., Diamantopoulos, T., Papamichail, M. D., & Symeonidis, A. L. (2020). Towards analyzing contributions from software repositories to optimize issue assignment. In *2020 IEEE 20th international conference on software quality, reliability and security (QRS)* (pp. 243–253).

Matter, D., Kuhn, A., & Nierstrasz, O. (2009). Assigning bug reports using a vocabulary-based expertise model of developers. In *2009 6th IEEE international working conference on mining software repositories* (pp. 131–140). 10.1109/MSR.2009.5069491.

Mishra, N., Chaturvedi, S., Vij, A., & Tripathi, S. (2021). Research problems in recommender systems. *Journal of Physics: Conference Series, 1717*(1), Article 012002.

Moher, D., Liberati, A., Tetzlaff, J., Altman, D. G., & Group, P. (2009). Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. *PLoS Medicine, 6*(7), Article e1000097.

Mohsin, H., & Shi, C. (2020). SPBC: A self-paced learning model for bug classification from historical repositories of open-source software. *Expert Systems with Applications*. 10.1016/j.eswa.2020.113808.

Mostafa, S., Findley, B., Meng, N., & Wang, X. (2021). Sais: Self-adaptive identification of security bug reports. *IEEE Transactions on Dependable and Secure Computing, 18*(4), 1779–1792 Scopus. 10.1109/TDSC.2019.2939132.

Murphy, G., & Cubranic, D. (2004). Automatic bug triage using text categorization. In *Proceedings of the sixteenth international conference on software engineering & knowledge engineering* (pp. 1–6).

Naguib, H., Narayan, N., Brügge, B., & Helal, D. (2013). Bug report assignee recommendation using activity profiles. In *2013 10th working conference on mining software repositories (MSR)* (pp. 22–30).

Najafabadi, M. K., Mohamed, A. H., & Mahrin, M. N. (2019). A survey on data mining techniques in recommender systems. *Soft Computing, 23*(2), 627–654.

Nasir, J. A., Khan, O. S., & Varlamis, I. (2021). Fake news detection: A hybrid CNN-RNN based deep learning approach. *International Journal of Information Management Data Insights, 1*(1), Article 100007. 10.1016/j.jjimei.2020.100007.

Nayebi, M., Dicke, L., Ittyipe, R., Carlson, C., & Ruhe, G. (2019). ESSMArT way to manage customer requests. *Empirical Software Engineering, 24*(6), 3755–3789. 10.1007/s10664-019-09721-w.

Pahins, C. A. D. L., D'Morison, F., Rocha, T. M., Almeida, L. M., Batista, A. F., & Souza, D. F. (2019). T-REC: Towards accurate bug triage for technical groups. In *2019 18th IEEE international conference on machine learning and applications (ICMLA)* (pp. 889–895).

Panda, R. R., & Nagwani, N. K. (2021). Multi-label software bug categorisation based on fuzzy similarity. *International Journal of Computational Science and Engineering, 24*(3), 244–258.

Panda, R. R., & Nagwani, N. K. (2022). Classification and intuitionistic fuzzy set based software bug triaging techniques. *Journal of King Saud University-Computer and Information Sciences*.

Park, J., Lee, M., Kim, J., Hwang, S., & Kim, S. (2011). Costriage: A cost-aware triage algorithm for bug reporting systems. In *Proceedings of the national conference on artificial intelligence* (p. 139).

Park, J. W., Lee, M. W., Kim, J., Hwang, S. W., & Kim, S. (2016). Cost-aware triage ranking algorithms for bug reporting systems. *Knowledge and Information Systems, 48*(3), 679–705. 10.1007/s10115-015-0893-9.

Rahman, M. M., Ruhe, G., & Zimmermann, T. (2009). Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects. In *2009 3rd international symposium on empirical software engineering and measurement* (pp. 439–442).

Rashid, M., Clarke, P. M., & O'Connor, R. V (2019). A systematic examination of knowledge loss in open source software projects. *International Journal of Information Management, 46*, 104–123. 10.1016/j.ijinfomgt.2018.11.015.

Rathore, S. S., & Kumar, S. (2017). A study on software fault prediction techniques. *Artificial Intelligence Review, 51*(2), 255–327. 10.1007/s10462-017-9563-5.

Ray, A., Kolekar, M. H., Balasubramanian, R., & Hafiane, A. (2023). Transfer learning enhanced vision-based human activity recognition: A decade-long analysis. *International Journal of Information Management Data Insights, 3*(1), Article 100142. 10.1016/j.jjimei.2022.100142.

Resketi, M. R., Motameni, H., Nematzadeh, H., & Akbari, E. (2020). Automatic summarising of user stories in order to be reused in future similar projects. *IET Software, 14*(6), 711–723. 10.1049/iet-sen.2019.0182.

Roopa, T. S., Purna, Y., & Krish, C. (2019). A novel approach for bug triaging with specialized topic model. *International Journal of Innovative Technology and Exploring Engineering, 8*(7), 1032–1038.

Sajedi-Badashian, A., & Stroulia, E. (2020a). Guidelines for evaluating bug-assignment research. *Journal of Software: Evolution and Process, 32*(9). 10.1002/smr.2250.

Sajedi-Badashian, A., & Stroulia, E. (2020b). Vocabulary and time based bug-assignment: A recommender system for open-source projects. *Software - Practice and Experience, 50*(8), 1539–1564. 10.1002/spe.2830.

Sarkar, A., Rigby, P. C., & Bartalos, B. (2019). Improving bug triaging with high confidence predictions at Ericsson. In *2019 IEEE international conference on software maintenance and evolution (ICSME)* (pp. 81–91).

Sawarkar, R., Nagwani, N. K., & Kumar, S. (2019). Predicting available expert developer for newly reported bugs using machine learning algorithms. In *2019 IEEE 5th international conference for convergence in technology (I2CT)* (pp. 1–4). 10.1109/I2CT45611.2019.9033915.

Sbih, A., & Akour, M. (2018). Towards efficient ensemble method for bug triaging. *Journal of Multiple-Valued Logic and Soft Computing, 31*(5–6), 567–590.

Schmitt, M. (2023). Deep learning in business analytics: A clash of expectations and reality. *International Journal of Information Management Data Insights, 3*(1), Article 100146. 10.1016/j.jjimei.2022.100146.

Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In *Recommender systems handbook* (pp. 257–297). Springer.

Sharma, M., & Singh, V. (2016). Clustering-based association rule mining for bug assignee prediction. *International Journal of Business Intelligence and Data Mining, 11*(2), 130–150.

Sharma, M., Tandon, A., Kumari, M., & Singh, V. B. (2017). Reduction of redundant rules in association rule mining-based bug assignment. *International Journal of Reliability, Quality and Safety Engineering, 24*(6). 10.1142/S0218539317400058.

Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2013). Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation. In *2013 10th working conference on mining software repositories (MSR)* (pp. 2–11).

Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2014). Improving automatic bug assignment using time-metadata in term-weighting. *IET Software, 8*(6), 269–278. 10.1049/iet-sen.2013.0150.

Shokripour, R., Anvik, J., Kasirun, Z. M., & Zamani, S. (2015). A time-based approach to automatic bug report assignment. *Journal of Systems and Software, 102*, 109–122. 10.1016/j.jss.2014.12.049.

Singh, V., Chen, S.-. S., Singhania, M., Nanavati, B., kar, A. k., & Gupta, A (2022). How are reinforcement learning and deep learning algorithms used for big data based decision making in financial industries–a review and research agenda. *International Journal of Information Management Data Insights, 2*(2), Article 100094. 10.1016/j.jjimei.2022.100094.

Suri, J., Agarwal, S., Gupta, S. K., Puvvula, A., Viskovic, K., Suri, N., … Fatemi, M (2021). Systematic review of artificial intelligence in acute respiratory distress syndrome for COVID-19 lung patients: A biomedical imaging perspective. *IEEE Journal of Biomedical and Health Informatics, 25*(11), 4128–4139. 10.1109/JBHI.2021.3103839.

Tamrawi, A., Nguyen, T. T., Al-Kofahi, J. M., & Nguyen, T. N. (2011a). Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 365–375). 10.1145/2025113.2025163.

Tamrawi, A., Nguyen, T. T., Al-Kofahi, J., & Nguyen, T. N. (2011b). Fuzzy set-based automatic bug triaging (NIER track). In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 884–887). 10.1145/1985793.1985934.

Tantithamthavorn, C., Hassan, A. E., & Matsumoto, K. (2018). The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. *IEEE Transactions on Software Engineering, 46*(11), 1200–1219.

Uddin, J., Ghazali, R., Deris, M. M., Naseem, R., & Shah, H. (2017). A survey on bug prioritization. *Artificial Intelligence Review, 47*(2), 145–180.

Uren, V., & Edwards, J. S. (2023). Technology readiness and the organizational journey towards AI adoption: An empirical study. *International Journal of Information Management, 68*, Article 102588.

Votto, A. M., Valecha, R., Najafirad, P., & Rao, H. R. (2021). Artificial intelligence in tactical human resource management: A systematic literature review. *International Journal of Information Management Data Insights, 1*(2), Article 100047. 10.1016/j.jjimei.2021.100047.

Vu, P. M., Nguyen, T. T., & Nguyen, T. T. (2020). Fuzzy multi-intent classifier for user generated software documents. In *Proceedings of the 2020 ACM southeast conference* (pp. 292–295).

Wani, M. A., Bhat, F. A., Afzal, S., & Khan, A. I. (2020). *Advances in deep learning*. Springer.

Wei, M., Guo, S., Chen, R., & Gao, J. (2018). *Enhancing bug report assignment with an optimized reduction of training set: Vol. 11062 LNAI*.

Wu, H., Ma, Y., Xiang, Z., Yang, C., & He, K. (2022). A spatial–temporal graph neural network framework for automated software bug triaging. *Knowledge-Based Systems, 241* Scopus. 10.1016/j.knosys.2022.108308.

Wu, W., Zhang, W., Yang, Y., & Wang, Q. (2011). Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-pacific software engineering conference* (pp. 389–396). 1457721996.

Xi, S. Q., Yao, Y., Xiao, X. S., Xu, F., & Lv, J. (2019). Bug triaging based on tossing sequence modeling. *Journal of Computer Science and Technology, 34*(5), 942–956. 10.1007/s11390-019-1953-5.

Xi, S. Q., Yao, Y., Xu, F., & Lü, J. (2018a). Bug triaging approach based on recurrent neural networks. *Ruan Jian Xue Bao/Journal of Software, 29*(8), 2322–2335. 10.13328/j.cnki.jos.005532.

Xi, S., Yao, Y., Xiao, X., Xu, F., & Lu, J. (2018). An effective approach for routing the bug reports to the right fixers. 1–10.

Xia, X., Lo, D., Ding, Y., Al-Kofahi, J. M., Nguyen, T. N., & Wang, X. (2017). Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering, 43*(3), 272–297. 10.1109/TSE.2016.2576454.

Xia, X., Lo, D., Shihab, E., & Wang, X. (2016). Automated bug report field reassignment and refinement prediction. *IEEE Transactions on Reliability, 65*(3), 1094–1113. 10.1109/TR.2015.2484074.

Xia, X., Lo, D., Shihab, E., Wang, X., & Zhou, B. (2015a). Automatic, high accuracy prediction of reopened bugs. *Automated Software Engineering, 22*(1), 75–109.

Xia, X., Lo, D., Wang, X., & Zhou, B. (2013). Accurate developer recommendation for bug resolution. In *2013 20th working conference on reverse engineering (WCRE)* (pp. 72–81).

Xia, X., Lo, D., Wang, X., & Zhou, B. (2015b). Dual analysis for recommending developers to resolve bugs. *Journal of Software: Evolution and Process, 27*(3), 195–220.

Xie, X., Zhang, W., Yang, Y., & Wang, Q. (2012). Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering* (pp. 19–28).

Xuan, J., Jiang, H., Hu, Y., Ren, Z., Zou, W., Luo, Z., & Wu, X. (2014). Towards effective bug triage with software data reduction techniques. *IEEE Transactions on Knowledge and Data Engineering, 27*(1), 264–280.

Xuan, J., Jiang, H., Ren, Z., Yan, J., & Luo, Z. (2017). Automatic bug triage using semi-supervised text classification. *ArXiv Preprint ArXiv:1704.04769*.

Xuan, J., Jiang, H., Ren, Z., & Zou, W. (2012). Developer prioritization in bug repositories. In *2012 34th international conference on software engineering (ICSE)* (pp. 25–35).

Xuan, J., Jiang, H., Zhang, H., & Ren, Z. (2017b). Developer recommendation on bug commenting: A ranking approach for the developer crowd. *Science China Information Sciences, 60*(7), Article 072105 1674-733X.

Yadav, A., & Singh, S. K. (2020). A novel and improved developer rank algorithm for bug assignment. *International Journal of Intelligent Systems Technologies and Applications, 19*(1), 78–101. 10.1504/IJISTA.2020.105178.

Yadav, A., Singh, S. K., & Suri, J. S. (2019). Ranking of software developers based on expertise score for bug triaging. *Information and Software Technology, 112*, 1–17. 10.1016/j.infsof.2019.03.014.

Yang, C., Fan, Q., Wang, T., Yin, G., Zhang, X., Yu, Y., & Wang, H. (2019). RepoLike: Amulti-feature-based personalized recommendation approach for open-source repositories. *Frontiers of Information Technology & Electronic Engineering, 20*(2), 222–237. 10.1631/fitee.1700196.

Ye, L., Jinxiao, H., & Yutao, M. (2020). An automatic method using hybrid neural networks and attention mechanism for software bug triaging. *Journal of Computer Research Development, 57*(3), 461.

Yin, Y., Dong, X., & Xu, T. (2018). Rapid and efficient bug assignment using ELM for IOT software. *IEEE Access: Practical Innovations, Open Solutions, 6*, 52713–52724. 10.1109/ACCESS.2018.2869306.

Yuan, Z., Chen, X., Cui, Z., & Mu, Y. (2020). ALTRA: Cross-project software defect prediction via active learning and tradaboost. *IEEE Access: Practical Innovations, Open Solutions, 8*, 30037–30049.

Zaidi, S. F. A., Awan, F. M., Lee, M., Woo, H., & Lee, C.-. G. (2020). Applying convolutional neural networks with different word representation techniques to recommend bug fixers. *IEEE Access: Practical Innovations, Open Solutions*, 2169–3536.

Zhang, T., Chen, J., Yang, G., Lee, B., & Luo, X. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software, 117*, 0164–1212 166-184.

Zhang, W., Cui, Y., & Yoshida, T. (2017). En-LDA: An novel approach to automatic bug report assignment with entropy optimized Latent Dirichlet Allocation. *Entropy, 19*(5) Art. 5. 10.3390/e19050173.

Zhang, W., Wang, S., & Wang, Q. (2016a). BAHA: A novel approach to automatic bug report assignment with topic modeling and heterogeneous network analysis. *Chinese Journal of Electronics, 25*(6), 1011–1018. 10.1049/cje.2016.08.012.

Zhang, W., Wang, S., & Wang, Q. (2016b). KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity. *Information and Software Technology, 70*, 68–84. 10.1016/j.infsof.2015.10.004.

Zhang, Y., Wu, Y., Wang, T., & Wang, H. (2019). A novel approach for recommending semantically linkable issues in GitHub projects. *Science China Information Sciences, 62*(9). 10.1007/s11432-018-9822-1.

Nagwani, Naresh Kumar, PhD, Senior Member IEEE, is an Associate Professor at the National Institute of Technology Raipur (NIT Raipur) India. He has more than 50 research papers and more than 10 book chapters to his credit. He is also having industrial experience of more than 3 years with Persistent Systems Limited, where he was part of the software development team and developed data analytics software such as SPSS and Transaction Processing System. He is also a Sun Certified Java programmer and Sun certified web component developer. He has teaching and research experience of 14 years. He has more than **900** citations and has an **H-index~20.**

Suri, Jasjit S., PhD, MBA, is an innovator, visionary, scientist, and an internationally known world leader in Biomedical Engineering and its Management. Dr. Suri received the **(i)** *Director General's Gold medal* in 1980 and is a Fellow of **(ii)** *Institute of Electrical and Electronics Engineering (FIEEE),* **(iii)** *American Institute of Medical and Biological Engineering*, awarded by National Academy of Sciences, Washington DC (2004), **(iv)** *American Institute of Ultrasound in Medicine* (2019), **(v)** *Asia Pacific Vascular Society* (2020) and **(vi)** recipient of *Life Time Achievement Award* from Marquis (2018). He is currently Chairman of AtheroPoint, Roseville, CA, USA, dedicated to imaging technologies for cardiovascular and stroke. He has nearly **~20,000** citations, co-authored **50** books, and has an **H-index~70.**