PLOS ONE

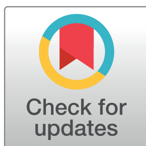
RESEARCH ARTICLE

Prioritizing tasks in software development: A systematic literature review

Yegor Bugayenko[1], **Ayomide Bakare**[2], **Arina Cheverda**[2], **Mirko Farina**[3]*, **Artem Kruglov**[2]*, **Yaroslav Plaksin**[2], **Witold Pedrycz**[4], **Giancarlo Succi**[5]

1 Huawei, Moscow, Russia, **2** Institute of Software Development and Engineering, Innopolis University, Innopolis, Russia, **3** Institute of Human and Social Sciences, Innopolis University, Innopolis, Russia, **4** Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada, **5** Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

* m.farina@innopolis.ru (MF); a.kruglov@innopolis.ru (AK)



OPEN ACCESS

Citation: Bugayenko Y, Bakare A, Cheverda A, Farina M, Kruglov A, Plaksin Y, et al. (2023) Prioritizing tasks in software development: A systematic literature review. PLoS ONE 18(4): e0283838. https://doi.org/10.1371/journal.pone.0283838

Editor: Bilal Alatas, Firat Universitesi, TURKEY

Received: January 4, 2023

Accepted: March 17, 2023

Published: April 6, 2023

Copyright: © 2023 Bugayenko et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: Data are all contained within the paper and/or Supporting information files.

Funding: This research was supported by Huawei Technologies. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Abstract

Task prioritization is one of the most researched areas in software development. Given the huge number of papers written on the topic, it might be challenging for IT practitioners–software developers, and IT project managers–to find the most appropriate tools or methods developed to date to deal with this important issue. The main goal of this work is therefore to review the current state of research and practice on task prioritization in the Software Engineering domain and to individuate the most effective ranking tools and techniques used in the industry. For this purpose, we conducted a systematic literature review guided and inspired by the Preferred Reporting Items for Systematic Reviews and Meta-Analyses, otherwise known as the PRISMA statement. Based on our analysis, we can make a number of important observations for the field. Firstly, we found that most of the task prioritization approaches developed to date involve a specific type of prioritization strategy—*bug prioritization*. Secondly, the most recent works we review investigate task prioritization in terms of "pull request prioritization" and "issue prioritization," (and we speculate that the number of such works will significantly increase due to the explosion of version control and issue management software systems). Thirdly, we remark that the most frequently used metrics for measuring the quality of a prioritization model are *f-score*, *precision*, *recall*, and *accuracy*.

Introduction

In software development, the vast majority of tasks do not have mandatory dependencies and it is up to the project manager to decide which task should be completed first. The proper continuous prioritization of tasks (known as *backlog refinement* in agile terminology) becomes a critical success factor for any software development project, as it guarantees that the company's crucial goals are in focus and can be met [1].

What is a task, though? The term "task" in software engineering refers to the smallest unit of work subject to management accountability that needs to be completed as part of a software development project [2]. So, in the context of software development, the term task is an umbrella term that encompasses concepts, such as "pull request" and "issue," commonly

found in GitHub/GitLab integration (so development areas) [3], or to ideas, such as "bug," "feature," "improvement," commonly used in task management. Although these concepts and ideas are considered conceptually independent, they often overlap in practice.

In an attempt to optimize the process and practice of task prioritization, researchers approached the problem from a bug-fixing perspective; that is, in terms of selecting the most appropriate developer for the given task [4]. Cubranic and Murphy were among the first to analyze the problem of task prioritization in terms of Machine Learning (ML); namely as a classification problem [5]. The datasets provided in their research, Eclipse (see https://bugs.eclipse.org/bugs/) and Mozilla (see http://www.mozilla.org/projects/bugzilla), have become "de facto" the standard for training and testing ML models for this problem domain.

However, it is worth noting, that other researchers developed alternative methods and approaches to improve the process of prioritizing and assigning bug fixes. For example, Zimmermann et al. [6] provided a series of recommendations for formulating and better classifying bug reports, while Anvik et al. [7] proposed an effective strategy for developers selection. Panjer [8] formulated a method capable of predicting bugs' lifetime and Wang et al. [9] suggested a new technique for identifying bug duplicates.

Menzies and Marcus [10] adopted another conceptual framework for dealing with the problem of task prioritization and proposed a solution based on the prediction of the severity of bug reports. Their work formed the conceptual palette necessary for the development of further research on bug priorities prediction, such as the works by Sharma et al. [11] and Tian et al. [12].

The importance, urgency, and significance of this problem for the Software Engineering community is also attested by the recent publication of several surveys, such as [13–16]. Among them, the work of Gousios et al. showed that the issue of task prioritization is particularly sensitive for development teams that follow a pull-based development model [16–18].

The considerations made above clearly demonstrate that task prioritization has become an active research topic in software engineering. On the one hand, its growth signals a positive trend: the more people get involved in the discussion of these issues, the more ideas are generated and accumulated in the scientific community. On the other hand, though, wide participation poses potentially insurmountable challenges for researchers and developers in terms of understanding the current state and capabilities of the field. Therefore, we believe that a comprehensive systematic literature review (SLR) carried out on this topic is going to be highly beneficial for researchers, project managers, developers, scrum masters, and other industry practitioners.

Research problem and objectives

Taking this important observation as a starting point this work reviews how the IT industry addresses the problem of task prioritization and attempts to produce a state-of-the-art summary of tools and techniques used for this purpose. Although we do not limit our work to specific methods, we expect to mostly gather Machine Learning (ML)-based approaches. This is because of the recent successes of ML in software engineering and computer science [19, 20].

The objectives of the SLR are therefore to:

- present our readership (mostly IT practitioners) with newly-developed techniques for ranking tasks that they can reliably use in their work,
- develop new strategies in ranking and prioritizing tasks, thus filling current gaps in the relevant literature and
- identify possible directions for future research.

The scientific contribution of this paper includes structured information on task prioritization, a survey of existing tools and approaches, methods, and metrics, as well as some estimates about their effectiveness and reliability.

Structure of the paper

This paper is organized as follows. Section Related Works provides an overview of current research on task prioritization and a helpful comparison between such research and the focus and scope of our work. Section SLR Protocol Development describes the protocol used in this systematic literature review. Section Results presents the results of our work, while section Discussion contextualizes our findings and section Critical Review of our Research Question provides their critical interpretation. Section Limitations, Threats to Validity, and Review Assessment evaluates the limitations and various other shortcomings potentially affecting our study, while section Conclusion summarizes what we achieved and points out future research directions.

Related works

There exist a number of studies devoted to requirements prioritization techniques. For example, Achimugu et al. [21] found that the most cited techniques for requirements prioritization include Analytical Hierarchy Process (AHP), Pairwise Comparison, Cost-Value Prioritization, and Cumulative Voting. More recent trends in prioritizing requirements include ML techniques (such as Case Base Ranking and Fuzzy AHP). Bukhsh et al. [22] also identified a trend toward fuzzy logic and machine learning methods. Somohano-Murrieta et al. [23] investigated the most documented techniques with regard to scalability and time consumption problems. Rashdan [24] found evidence of a shift towards computed-assisted/algorithmic methods, while Sufian et al. [25] analyzed factors that influence prioritization and identified commonly used techniques and tools aimed at improving the process. These studies underline the importance and evolution of requirements prioritization techniques, and -at the same time- emphasize the need for real-world evaluations and scalability solutions.

There are also studies aimed at analyzing other aspects of software engineering, which are typically connected with prioritization issues (such as analysis of non-functional requirements, code smells, technical debt, and software bugs). For example, Kaur et al. [26] identified existing techniques for code smell prioritization and introduced different tools for prioritizing code smells (such as Fusion, ConQAT, SpIRIT, JSpIRIT, PMD, Fica, JCodeOdor, and DT-SOA). Alfayez et al. [27] investigated technical debt prioritization and identified a number of important techniques used, which include: Cost-Benefit Analysis, Ranking, Predictive Analytics, Real Options Analysis, Analytic Hierarchy Process, Modern Portfolio Theory, Weighted Sum Model, Business Process Management, Reinforcement Learning, and Software Quality Assessment Based on Lifecycle Expectations (SQALE). However, the researchers concluded that more research is needed to develop technical debt prioritization approaches capable of effectively considering costs, values, and resource constraints. Ijaz et al. [28] looked at non-functional requirements prioritization techniques and found that AI techniques can potentially handle uncertainties in requirements while contributing to overcome the most common limitations characterizing standard approaches (such as AHP). Pasikanti and Kawaf [29] studied the latest trends in software bug prioritization and identified a series of ML techniques (such as Naive Bayes, Support Vector Machines, Random Forest, and Multinational Naive Bayes) that are most commonly used for prioritizing software bugs.

Several SLRs were also conducted to identify the most commonly used techniques for test case selection and prioritization in software testing. For example, Pan et al. [30] found that

Supervised Learning, Unsupervised Learning, Reinforcement Learning, and NLP-based methods have been applied to test case prioritization; yet, due to a lack of standard evaluation procedures, the authors couldn't draw reliable conclusions on their effective performance. Bajaj and Sangwan [31] observed that genetic algorithms bear great potential for solving test case prioritization problems, while nevertheless noting that the design of parameter settings, type of operators, and fitness function significantly affects the quality of the solutions obtained.

Another important area of research focuses on aspects of task assignment and allocation in software development projects. Filho et al. [32] reviewed works on multicriteria models for task assignment in distributed software development projects with a special focus on qualitative decision-making methods. TAMRI emerged as the most efficient and widely used approach, while McDSDS, Global Studio Project, and 24-Hour Development Model received lower scores. Fatima et al. [33] studied the models used for task assignment and scheduling in software projects. The review found that static models are the most widely used for task scheduling, while the Support Vector Machine algorithm is the most widely used for task assignment. Both these papers demonstrated the importance of considering, as crucial for the practice of software management, specific factors (such as personal aspects, team skills, labor cost, geographic issues, and task granularity).

However, the contribution of our SLR is unique and different from that of the above-mentioned studies because: (Table 1):

- Unlike other SLRs, which have focused -as we have seen above- on prioritization techniques for requirements, test cases, bugs, and/or other artifacts of software development; our own review provides a comprehensive coverage of the problem at stake. Crucially, it does so by describing the broad category of "task", without focusing on a specific type of prioritized item.
- In addition, our research differs from prior studies on task allocation/assignment in several aspects. Firstly, the problem of assignment/allocation involves distributing tasks based on

Table 1. Summary of existing related literature reviews.

Reference	Covered years	Number of studies	Domain	Focus
Achimugu et al. [21]	1996–2013	73	Software requirements	Requirements prioritization techniques
Sufian et al. [25]	2009–2017	33	Software requirements	Requirements prioritization techniques and tools
Bukhsh et al. [22]	2007–2019	102	Software requirements	Requirements prioritization methods and their empirical evaluation
Somohano-Murrieta et al. [23]	2010–2019	35	Software requirements	Requirements prioritization techniques
Rashdan A. [24]	2014–2020	53	Software requirements	Taxonomy and trends in requirements prioritization techniques
Bajaj and Sangwan. [31]	1999–2018	20	Regression testing	Use of genetic algorithms in test case prioritization
Pan et al. [30]	2006–2020	29	Regression testing	Use of ML techniques for test case selection and prioritization
Kaur et al. [26]	till 2020	23	Code smells	Code smell prioritization techniques and tools
Alfayez et al. [27]	1992–2018	23	Approaches and techniques for technical debt	technical debt prioritization
Ijaz et al. [28]	2008–2019	30	Non-functional requirements	Non-functional requirements prioritization methods, including AI, and their validation
Pasikanti and Kawaf [29]	2015–2022	34	Software defects and bugs	Techniques, algorithms and methods of defects or bugs prioritization
Filho et al. [32]	till 2016	21	Task assignment	Qualitative decision-making models for assigning tasks in distributed software development projects
Fatima et al. [33]	2012–2019	23	Task assignment	Techniques and ML algorithms for software project scheduling
This study	2006–2022	83	Task prioritization	ML methods and metrics for task prioritization

https://doi.org/10.1371/journal.pone.0283838.t001

various factors (such as skills, availability, workload, etc), whereas the problem of prioritization focuses on determining which tasks should be completed first. Secondly, prior research has predominantly relied on qualitative analyses of algorithms, methods, and tools for task allocation/assignment, without conducting detailed quantitative analyses of their effectiveness. Our research aims to bridge these important gaps in the literature by conducting a comprehensive quantitative analysis of task prioritization techniques, which are used to determine their effectiveness in different contexts.

SLR protocol development

SLRs offer a comprehensive analysis of the research conducted in the field while also providing critical, original insights [34]. They are of paramount importance for scientific progress and, for this reason, represent one of the preferred methods used by researchers to investigate the state of the art of a particular research topic [35].

The quality of SLRs can vary greatly and it is important to ensure that an SLR is conducted in a rigorous and systematic manner [36, 37]. Thus, to ensure the comprehensiveness and soundness of our work we followed the PRISMA Statement [38], which is essentially a checklist, conventionally adopted by researchers worldwide, to guide, orient, and inform the development of any SLR. The PRISMA 2020 checklist adopted for this study is included as S1 Table.

Since the Prisma checklist abovementioned is not -strictly speaking- a methodological framework; rather a series of suggestions or -better- recommendations to be implemented for the sound development of any SLR (even beyond computer science), we decided to integrate it and complement it with a more specific methodological framework; the one recently developed by Kitchenham and Charters [39]. This framework was chosen due to its focus on software engineering and because its effectiveness has been amply demonstrated in previous studies [40–42]. We believe that complementing the general indications or recommendations outlined in the PRISMA checklist (which are valid for any field) with a framework specifically designed for research on software engineering is highly beneficial for this study, as it guarantees better accuracy. In addition, since the checklist and the framework partially overlap (despite being also complementary), one can use them to mutually strengthen each other. The stages of the methodological framework adopted in this SLR, are:

1. Specification of the research questions.
2. Development of the review protocol.
3. Formulation of the literature log.
4. Performance of quality assessment.
5. Extraction of Data.
6. Data synthesis.
7. Formulation of the main report.
8. Evaluation of the review and of the report.

Research questions

The first step in any SLR involves the formulation of a series of research questions that can guide and inform its development.

To formulate the most appropriate research questions, we adopted the Goal Question Metric (GQM) model developed by Basili et al. [43]. This model requires specifying up front the purpose of analysis, the objects and the issues to analyze, as well as the standpoints from which the analysis is performed. The Goal Question Metric model for this work is the following:

Purpose Systematic literature review.

Object Peer review publications in computer science and software engineering.

Issue Approaches for ranking tasks in software development.

Viewpoint Software engineers and industry practitioners.

With the GQM model in place, we then formulated the Research Questions (RQs) that characterized this work:

RQ1 What are the existing approaches for automatic task ranking in software development?

RQ2 Which methods are used in automatic task ranking models and approaches and how is their effectiveness assessed?

RQ3 What are the most effective and versatile models for automatic task ranking developed so far?

The motivation for RQ1 is to gain a clear understanding of existing research on the topic. Then, moving from general to more specific tasks, we formulate RQ2 with the intent of finding out which methods for task ranking are currently the most popular in the software development industry and how their effectiveness can be assessed. Further research along these lines leads to RQ3, through which we try to rank such methods in terms of effectiveness, accuracy, fidelity, and reliability. This could help developing new ranking strategies and remedial approaches for the field.

Literature search process

Following the best practices in the field [42], we selected the following databases for our searches: Google Scholar, Microsoft Academic, ScienceDirect, IEEE Xplore, and ACM digital library.

We then extracted a set of basic keywords, which describe our research questions. The keywords are: *a)* manage, *b)* backlog, *c)* priority, *d)* task, *e)* job, *f)* commit, *g)* bug, *h)* pull request, *i)* issue, *j)* feature, *k)* software, *l)* rank, *m)* distributed software development, *n)* machine learning.

Searches via keywords yielded a very large number of papers. Thus, to screen out irrelevant documents and add focus and precision to our work, we formulated a set of search queries by using Boolean operators, as common in the literature. Upon conducting an initial screening of papers, we discovered that there were more papers focused on prioritizing bug reports than those focused on prioritizing pull requests and issues. Because of this, we decided to model a series of search queries around these themes for better coverage. The list of queries we used for our searches is reported below:

1. ((pull OR merge) AND request) OR Github issue) AND (prioritization or priority OR rank OR order OR ranking OR ordering)
2. (task or bug or defect or feature) AND (prioritization or priority OR rank OR order OR ranking OR ordering)
3. bug severity AND priority AND (machine learning OR neural network)

Table 2. Results of the search queries in a number of scientific databases. Acronyms used: GS—Google Scholar, IEEE—IEEE Xplore, MA—Microsoft Academic, SD—ScienceDirect.

Query	GS	MA	SD	IEEE	ACM
1	52400	9106	143702	0	582019
2	20500	10619	1813	2	431329
3	22000	13177	430	10	66381
Total	358900	36842	15118	256	1108517

https://doi.org/10.1371/journal.pone.0283838.t002

We performed our searches by using these queries on the selected databases. Table 2 displays the results we obtained.

Inclusion and exclusion criteria

Next, we specified inclusion (IC) and exclusion (EC) criteria as recommended by Patino and Ferreira [44]. IC and EC help the authors decide which articles found through seminal searches deserve to be considered for further analysis. In this study we used the following IC and EC:

IC1 The paper is written in English.

IC2 The paper is peer-reviewed and published by a reputable publisher.

IC3 The paper was published as early as 2006*.

IC4 The paper uses ML techniques to deal with backlog systems or tasks/todos.

IC5 The paper compares different ML models or compares ML models with other learning models.

EC1 The paper does not satisfy at least one of the ICs.

EC2 The paper is a duplicate or contains duplicate information.

EC3 The paper is an editorial, an opinion piece, or an introduction. In general, the paper is excluded if it does not contain any original insight.

EC4 The paper does not present any type of experimentation or comparison or results.

*Shoham et al. [45] noted that around 2006, there was a pick of interest in ML in the software engineering community. We thus selected this year as the starting point for our systematic review.

Search results by sources

In this subsection, we offer to our readers a detailed description of the process that led to the inclusion of preliminary selected papers in our final reading log (Table 3).

We note that we only considered the first 100 results displayed in the relevant databases for each of the four queries we formulated. This is justified by the fact that the databases we used normally sort out results by significance and credibility (e.g., h-index, number of citations, impact factor, etc.) and by the observation that usually no relevant paper is found after the first 100 results.

The PRISMA flow chart diagram shown in Fig 1 represents the process of inclusion/exclusion visually for the reader.

Table 3. Papers selection. The table shows the procedure through which potentially relevant papers were screened out through the adoption of IC and EC criteria. The number of papers included in the final reading log is shown in the column "Selected papers".

Source	Initial selection	Potentially relevant	Removed papers									Selected papers
			IC1	IC2	IC3	IC4	IC5	EC1	EC2	EC3	EC4	
Google Scholar	400	117	1	5	5	1	-	-	42	4	-	59
Microsoft Academic	400	0	-	-	-	-	-	-	-	-	-	-
ScienceDirect	400	41	-	2	-	6	-	-	24	-	-	9
IEEE Xplore	256	41	-	-	-	-	-	-	29	-	-	12
ACM	400	22	2	-	-	-	-	-	13	4	-	3

https://doi.org/10.1371/journal.pone.0283838.t003

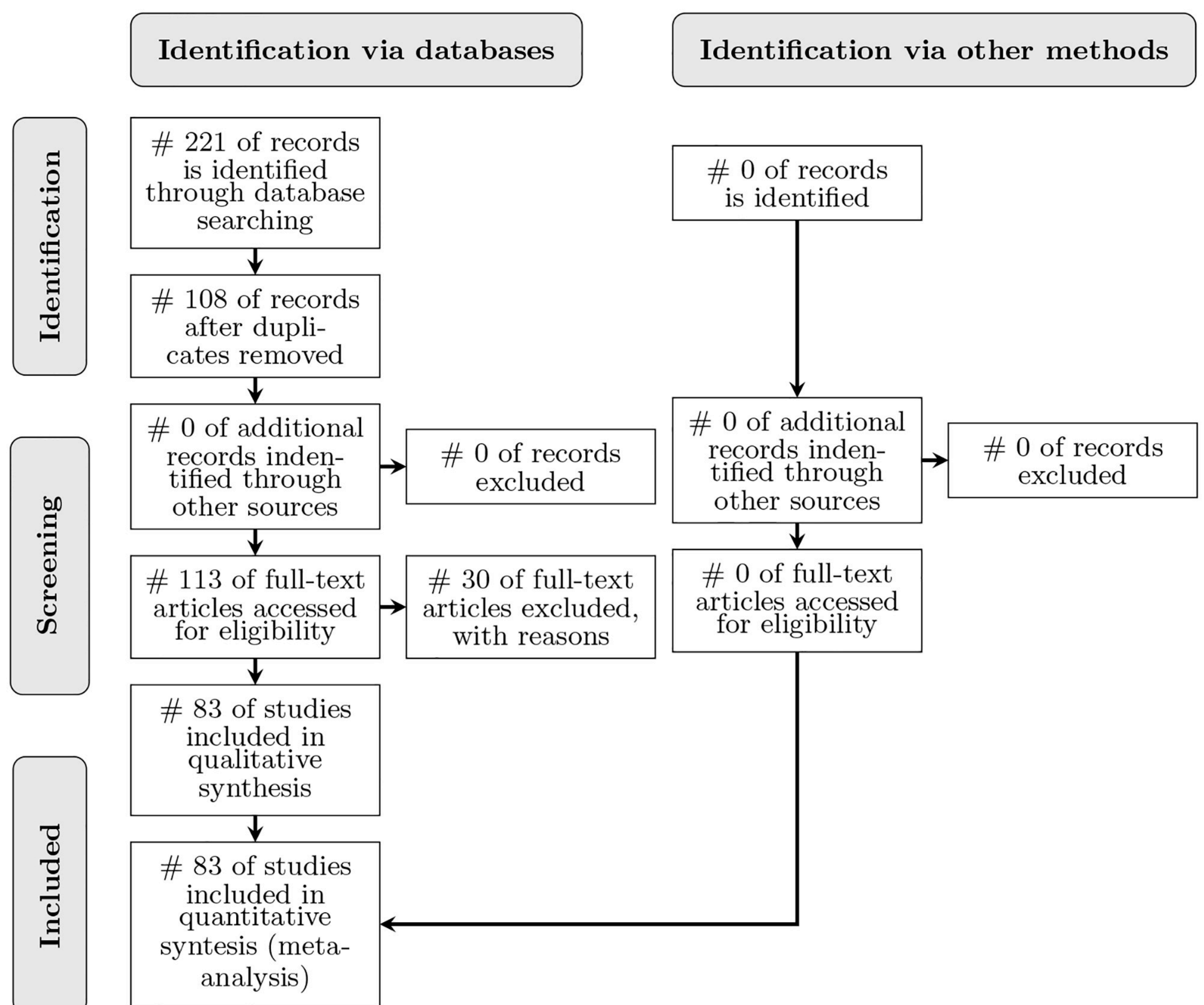


Fig 1. PRISMA flow diagram. It shows the stages of the search process as a flowchart diagram [38].

https://doi.org/10.1371/journal.pone.0283838.g001

Quality assessment

To assess the quality of the manuscripts, we defined a set of criteria and applied them to all the papers selected for inclusion in our reading log:

QA1 Were the objectives and the research questions clearly specified?

QA2 Were the results evaluated critically and comprehensively?

QA3 Was the research process transparent and reproducible?

QA4 Are there comparisons with alternatives?

We then determined whether the papers we selected matched the criteria and—in case—the extent to which they did so. So, we assigned 1 if a paper fully matched the criterion, 0.5 if it partially matched the criterion, and 0 otherwise.

The criteria used for QA1 are:

Fully matched The objectives and research questions were explicitly stated.

Partially matched The goals of the paper and its research questions were sufficiently clear but could be improved.

Not matched No objectives were stated if the research questions were hard to determine, or if they didn't relate to the research being carried out.

The criteria used for QA2 are:

Fully matched The authors of the paper provided a critical, balanced, and fair analysis of their results.

Partially matched The results were only partly (sufficiently) scrutinized and a comprehensive critical analysis was missing.

Not matched The authors did not evaluate their results.

The criteria used for QA3 are:

Fully matched The paper specified the methodology and the technologies used as well as the data gathered.

Partially matched Minor details were lacking (for example, a dataset is not readily available).

Not matched It was impossible to restore the sequence of actions or if other critical details (such as an algorithm or technologies used) were missing.

The criteria used for QA4 are:

Fully matched A comparison with other solutions offered; advantages and limitations clearly stated.

Partially matched The comparison was offered, but it was not comprehensively discussed.

Not matched No comparison was provided.

The resulting scores are shown in S2 Table, and their distribution can be found in Fig 2.

There were many high-quality papers among those we selected for inclusion in our final log, which is demonstrated by the scores reported in Fig 2. The average quality score was 2.9 out of 4. This confirms the reliability of the findings on which we based our SLR.

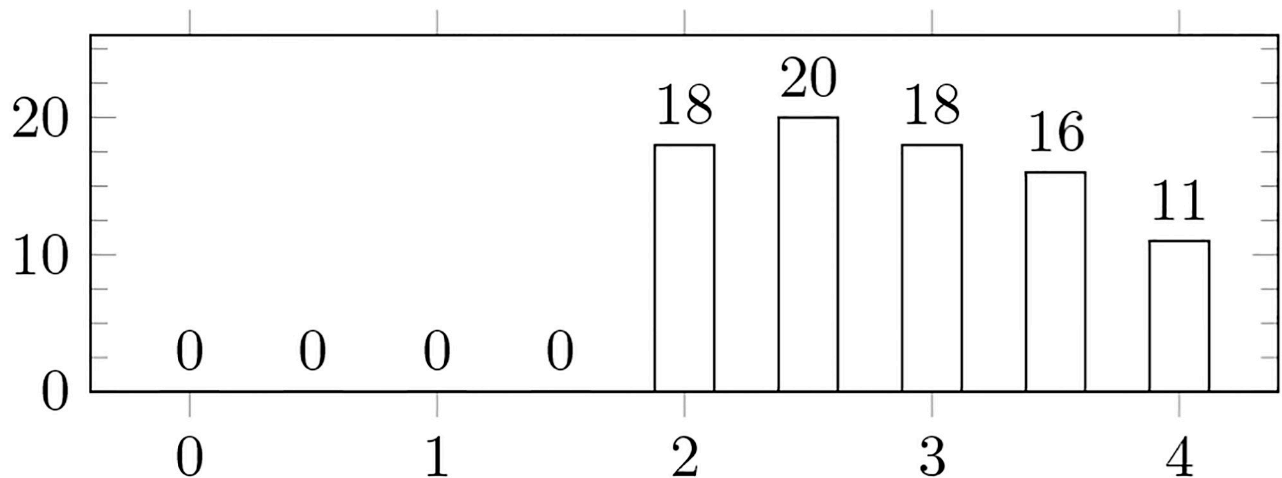


Fig 2. Papers distribution by quality score. Each paper was evaluated on a scale from 0 to 1 as per QA1-QA4. The bars display the number of papers with their respective quality score.

https://doi.org/10.1371/journal.pone.0283838.g002

Results

This section presents the findings gathered from the papers we included in our final reading log. More specifically, in this section, we use a series of statistical tools to cluster and organize the papers we selected in meaningful ways. Such clustering is beneficial for our readers because it provides some background for the conclusions we will draw in subsequent sections.

Preliminary clustering

We start this process of clustering by summarizing the potential advantages and disadvantages of the databases we used to perform our searches.

Microsoft Academic has the advantage of extensive coverage of scientific research, including patents. Its limitation is that some of the papers it lists are not peer-reviewed.

IEEE Xplore provides peer-reviewed publications, generally of high quality. Its limitation is that its full functionality requires a subscription, which is pricy.

ScienceDirect offers comprehensive coverage with tools for statistical analysis. However, it is beyond a paywall and has limitations for query building.

ACM provides comprehensive coverage with a particular emphasis on IT. Its major limitation is that it requires a subscription.

Google Scholar is one of the best database aggregators. It provides comprehensive coverage and tools for statistical analysis. However, it includes grey literature and non-peer-reviewed publications.

While not of crucial importance for the development of this work, we notice that such–complementary–information can be useful to ensure the academic integrity and scientific soundness of our approach.

The distribution of the papers included in the final reading log by databases is presented in Table 3. Fig 3 re-elaborates the information contained in Table 3 in the form of a pie chart, which is probably more appealing for the reader. For convenience, we attributed papers to

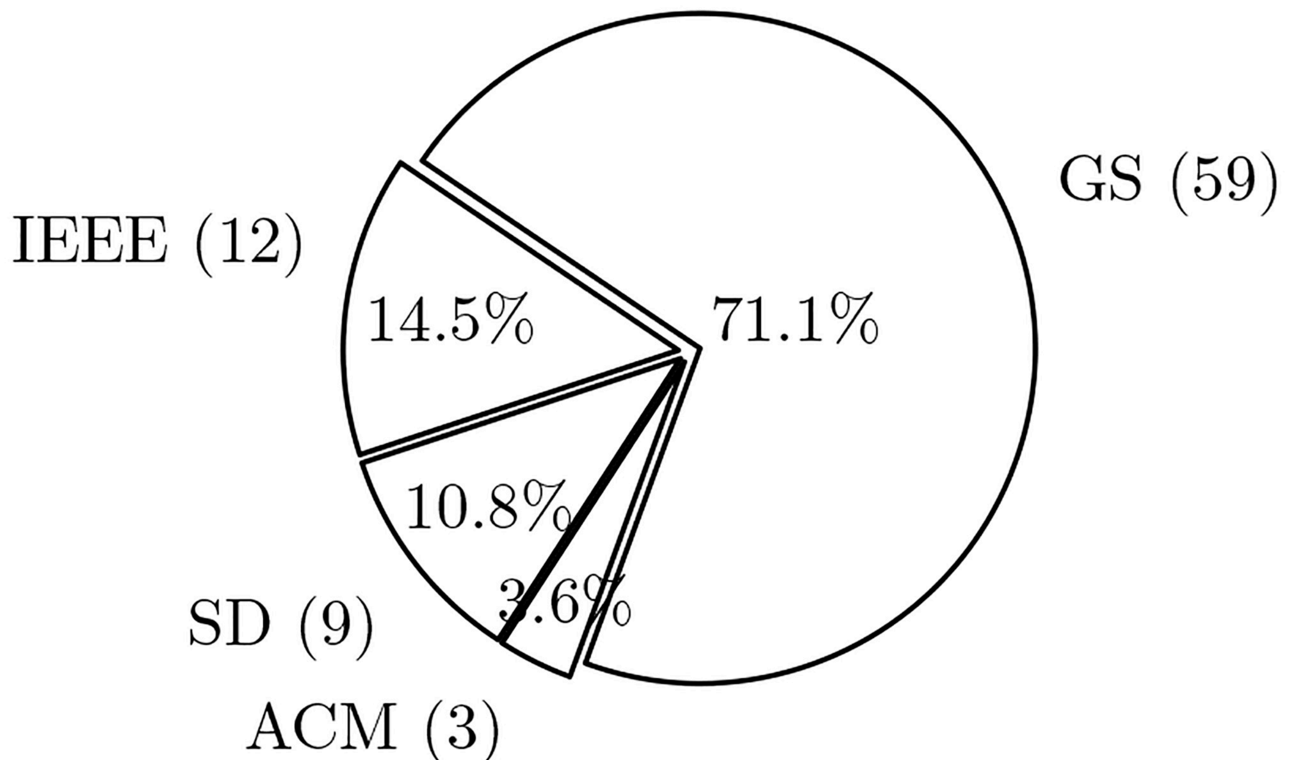


Fig 3. Papers distribution by databases. The pie chart shows the percentage of papers found in the databases we considered in this study. Acronyms used: GS—Google Scholar, IEEE—IEEE Xplore, SD—ScienceDirect. The number of papers is given in brackets.

https://doi.org/10.1371/journal.pone.0283838.g003

single repositories (even though some papers could be found across different databases). The attribution was subjective in character and determined by the chronological order of the searches we performed.

To give the reader a fuller picture of our results, we added information about the distribution of papers by publisher. This information can be found in Fig 4. We note that the following journals and publishers fall under the label "others," which accounts for about 14% of selected studies: ASTL (SERSC), CES (hikari), EISEJ, IJACSA, IJARCS, IJCNIS, IJCSE, IJOSSP, JATIT, Sensors (MDPI), and TIIS (KSII).

Studies classification

In this subsection, we present a series of statistical data that can be used to cluster our findings. Firstly, we identified 2 major topics characterizing the studies we included in our reading log: "Bug prioritization", "Bug severity prediction", and 2 minor topics "Issue prioritization", and "Pull Request prioritization". It is worth noting that even though bug severity [46] and bug priority [47] are two different theoretical entities (often treated as such even by project managers), a few works [12, 48, 49] demonstrated that severity can sometimes help predict priority. This is why, in this study, we consider not only papers concerned with bug priority but also those related to bug severity. Table 4 shows the distribution of publications across these topics.

Secondly, we clustered the distribution of topics by year of publication (Figs 5 and 6). The dynamics of growth for the key topics underlying this study are roughly the same. This suggests that the scientific community is equally interested in both topics. As we noted above, this demonstrates their close interrelation.

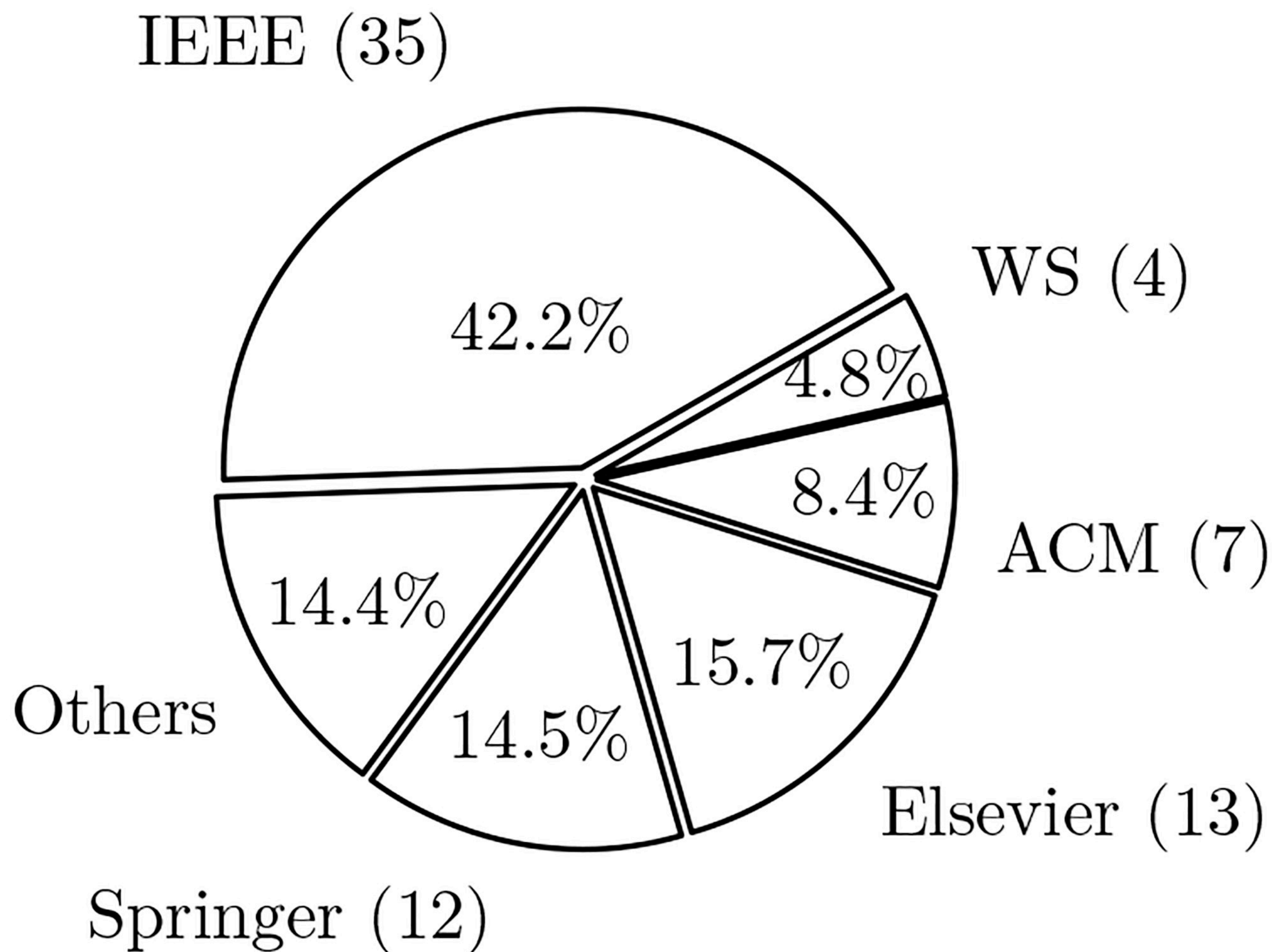


Fig 4. Papers distribution by publishers. The pie chart shows the distribution (in percentages) of the papers we considered in our reading log by publishers. Acronyms used: IEEE—IEEE Xplore, WS—World Scientific.

https://doi.org/10.1371/journal.pone.0283838.g004

Thirdly, building and expanding on this classification, we clustered the papers we selected by the year of publication. Fig 7 shows our results. The same information is presented in Table 5, where it is aggregated and visualized over a 4-years period.

Data from Table 5 suggests that the topics of our SLR are becoming the focus of many researchers worldwide (about 50% of the papers included in our reading log were produced in the last four years). We can also observe that the number of papers on these topics has grown at least two times over the last four years. This can be (presumably) explained by the

Table 4. Papers distribution by key topics. The table shows the number (column "Quantity") of papers devoted to a particular key topic (column "Topic"). Note: 2 papers have content for both topic 1 and 2 distribution.

Topic	Earliest publication date	Latest publication date	Quantity
Bug prioritization	2010	2022	31
Bug severity prediction	2012	2022	43
Issue priority	2014	2020	4
Pull Request priority	2014	2021	6

https://doi.org/10.1371/journal.pone.0283838.t004

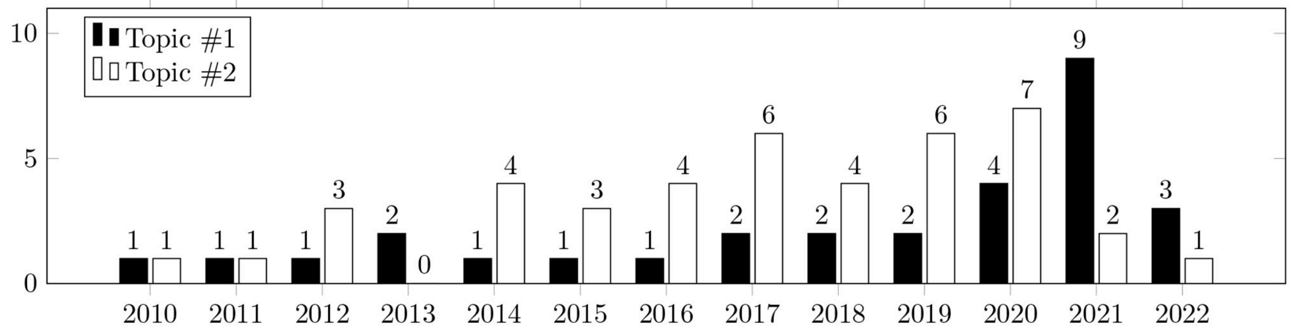


Fig 5. Topics distribution by year of publication. The bars show the number of papers related to the key topic published in a particular year. Black bars show the number of papers related to "bug prioritization". White bars show the number of papers related to "bug severity and prediction".

https://doi.org/10.1371/journal.pone.0283838.g005

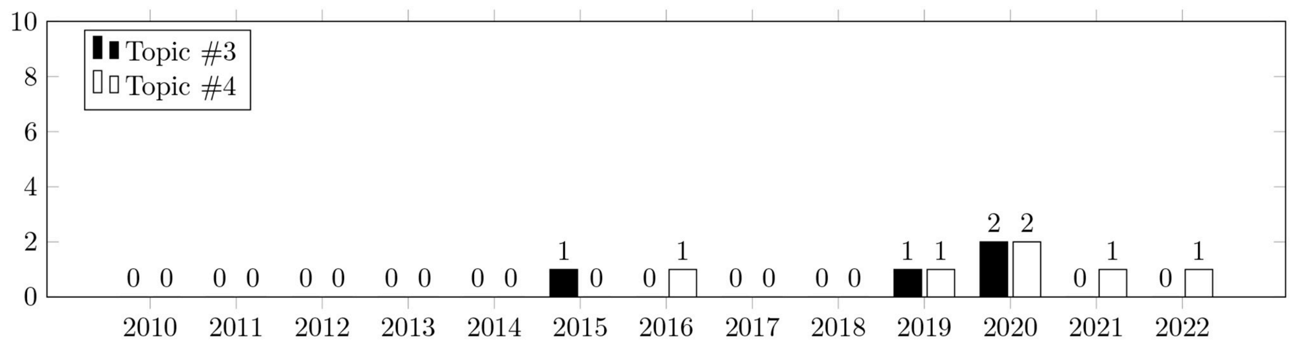


Fig 6. Topics distribution by year of publication. The bars show the number of papers related to the key topic published in a particular year. Black bars show the number of papers related to "issue prioritization". White bars show the number of papers related to "pull request prioritization".

https://doi.org/10.1371/journal.pone.0283838.g006

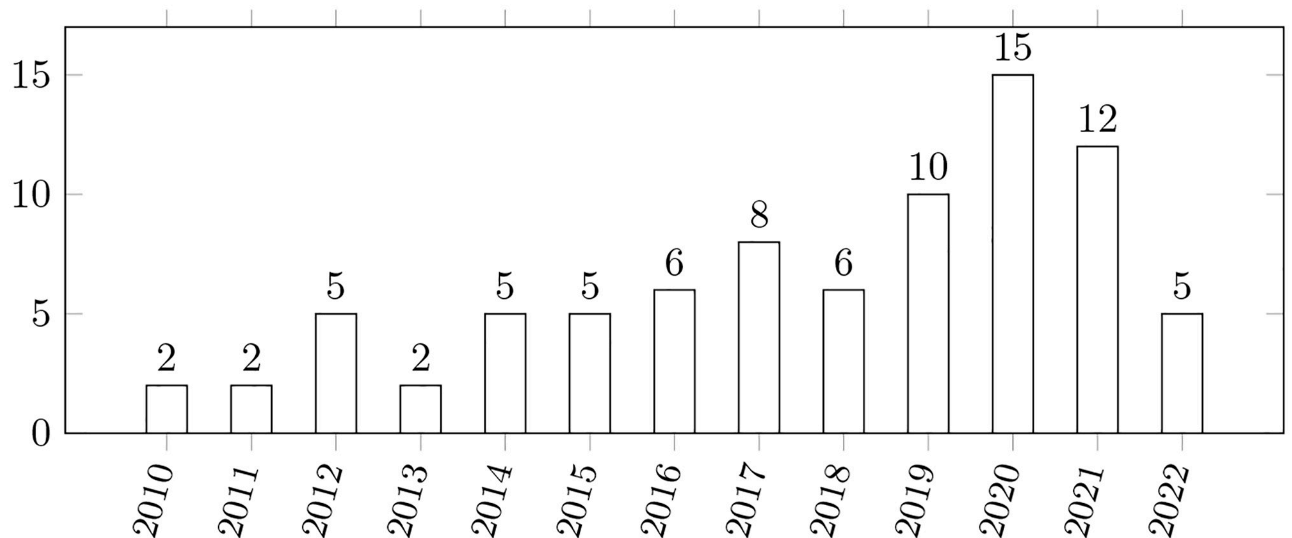


Fig 7. Papers distribution by years. The bars show the number of papers published on the topic between 2010 and 2021. No papers we found for the period 2006–2009. 2006 was the starting year for our SLR IC3.

https://doi.org/10.1371/journal.pone.0283838.g007

Table 5. Papers distribution over a 4-years period. The table shows the number (column "Quantity") and percentage (column "Percentage") of papers for the specified period (column "Years").

Years	Quantity	Percentage
2010–2014	16	19.3
2015–2018	25	30.1
2019–2022	42	50.6

https://doi.org/10.1371/journal.pone.0283838.t005

widespread adoption of new techniques in ML, which was probably determined by an increased interest in Artificial Intelligence (AI). To verify this hypothesis, we found the correlation between these two topics by calculating the relevant Pearson coefficient [50], which is given by:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}}, \quad (1)$$

where x_i is the number of papers published on the keyword "artificial intelligence" counted using the Scopus query (artificial AND intelligence), and y_i is the number of papers relevant for this SLR, for the i-th year in the period 2010–2022.

The Pearson correlation coefficient is 0.9 with a p-value of $6.4e - 06$. This confirms our assumption that there is a significant synergy between the growth in the number of ML tools and their application to our problem domain.

Fourthly, we also collected some statistics related to the tags used in the papers we included in the final reading log. Information about this point is presented in Fig 8.

We note that the information presented in Fig 8 can be used to:

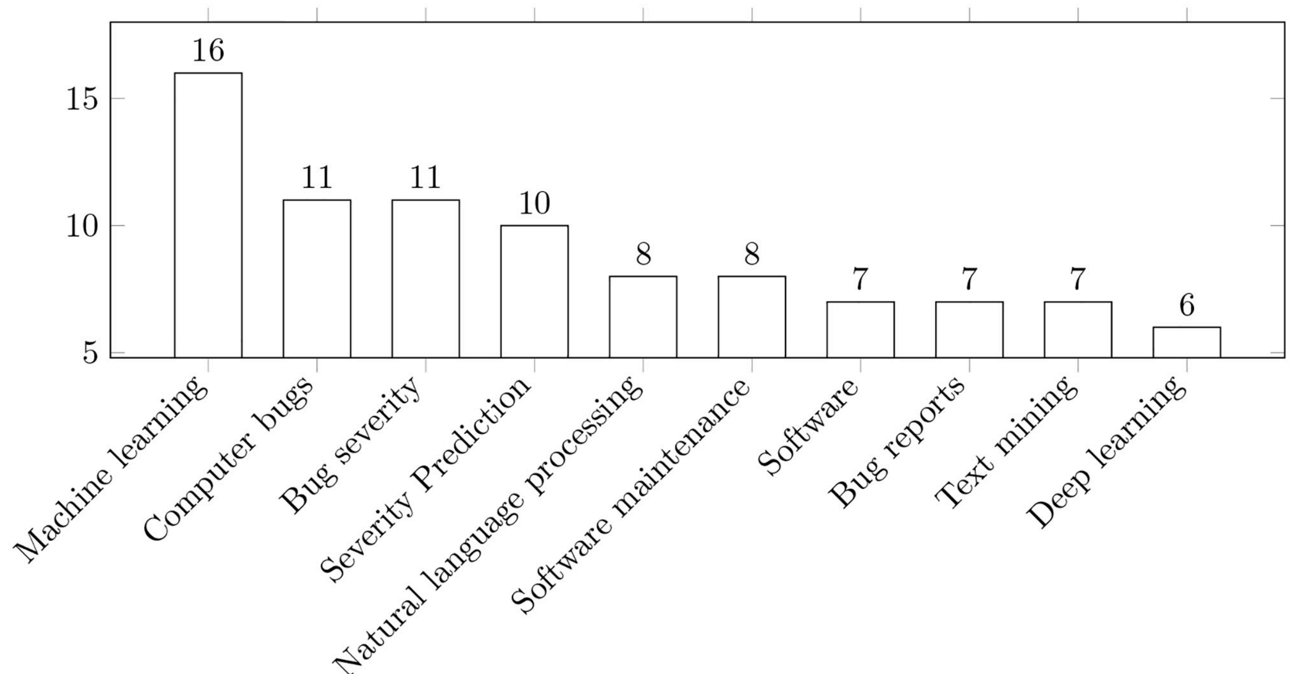


Fig 8. Tags distribution among the papers included in the final reading log. The bars show the number of papers related to a specific tag. Note: several tags can be assigned to one paper.

https://doi.org/10.1371/journal.pone.0283838.g008

- add more substance to the conclusions related to algorithm distribution we made in Further Clustering;
- validate the relevance and the significance of our selection (the papers included in the final reading log);
- characterize the most popular "subtopics" investigated by researchers worldwide in the selected domain.

Further clustering

We next proceed to further cluster our results and we do so along three dimensions: *a)* algorithms, *b)* datasets, and *c)* metrics. Fig 9 shows the algorithms used for training the models. Naive Bayes [51] is the most popular method among the models observed in the papers we reviewed.

In addition, we also clustered the datasets used in the papers included in the final reading log. The most often used datasets are presented in Fig 10.

Fig 10 shows the datasets most frequently used, which account for 48.7% of all datasets. The remaining datasets, accounting for 51.3% of the total, have been used only once, for example, bug repository of hdfs, etc. It is also worth noting that a single dataset can be found in many articles. The total number of dataset occurrences is calculated based on this important observation.

Finally, we collected statistics about the metrics used in the papers we included in our reading log (Fig 11). We did not plot the metrics reported once. Nevertheless, we believe that such metrics are important because they might be used to create a comprehensive overview of their usage, which can be instrumental in evaluating the effectiveness of task prioritization models.

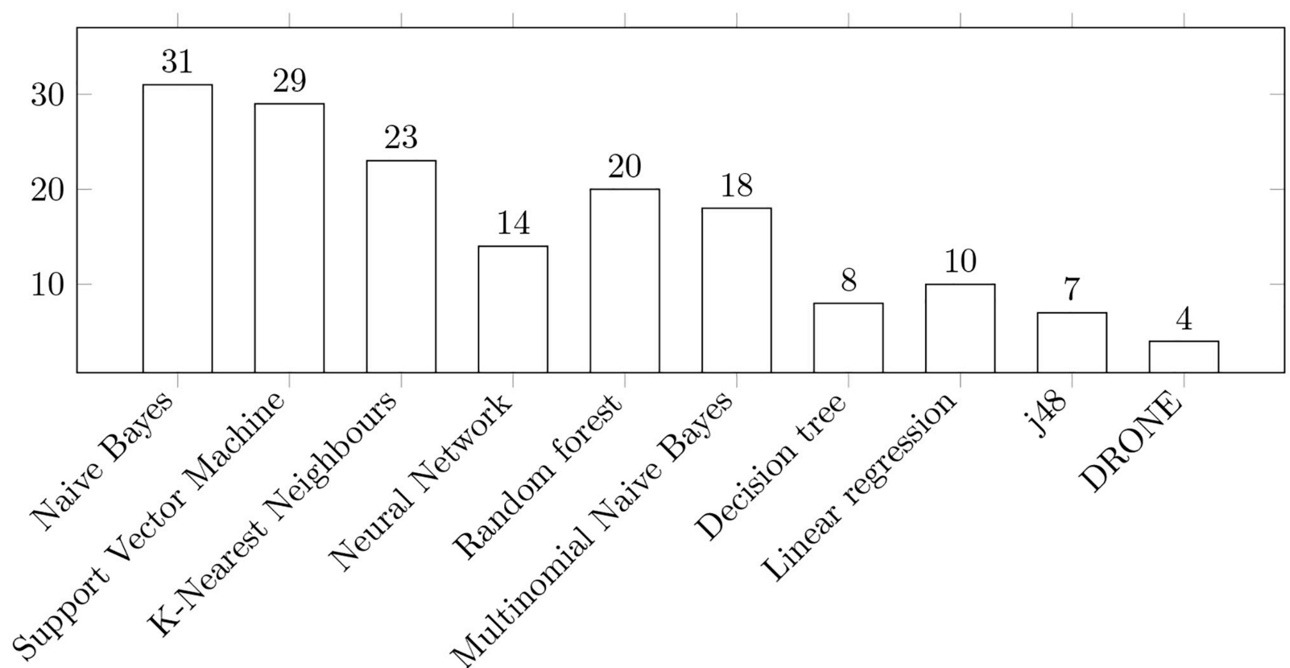


Fig 9. Algorithms used in the papers included in the final reading log. The bars show the number of papers in which the specified algorithms were considered. Note: several algorithms can be considered in one paper.

https://doi.org/10.1371/journal.pone.0283838.g009

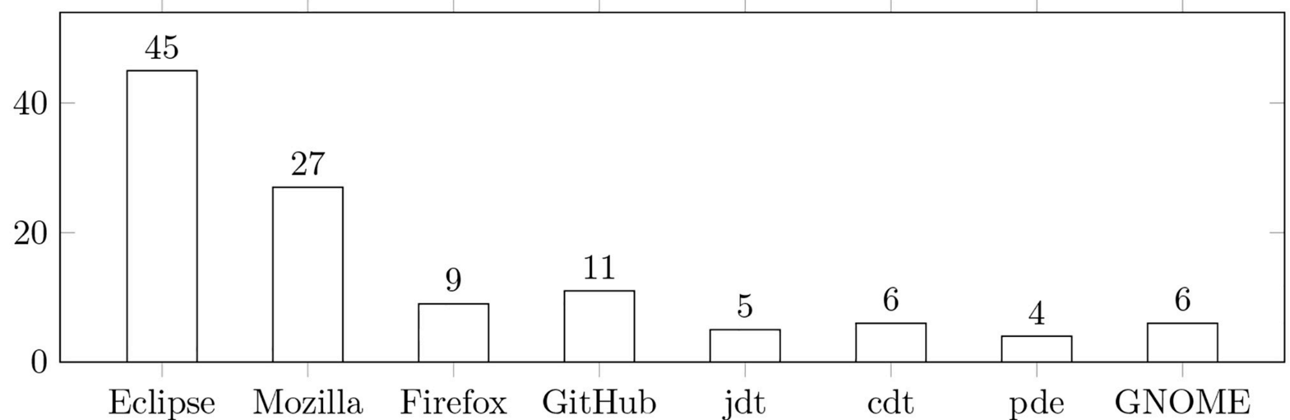


Fig 10. Distribution of datasets. The bars show the number of papers in which the specified algorithms were considered. Note: several datasets could be considered in one paper.

https://doi.org/10.1371/journal.pone.0283838.g010

These metrics include: average percentage of faults detected (APFD), normalized discounted cumulative gain (NDCG), mean squared error (MSE), Cohen's kappa coefficient, nearest false negatives, nearest false positives, adjusted r squared, prediction time, training time, and robustness.

We note that some papers may contain multiple metrics, which might be jointly used to assess and more comprehensively evaluate the quality of a model. The f-score, as shown in Fig 11, is the most commonly used metric in the papers we reviewed.

Discussion

In this section, we contextualize and critically discuss the data presented in Results, while also highlighting their significance and relevance for the field.

RQ1. What are the existing approaches for automatic task ranking in software development?

As discussed in the Introduction, task prioritization can be divided into 3 subtopics: issues prioritization, PRs prioritization, and bugs prioritization. In this study, we treat these topics as

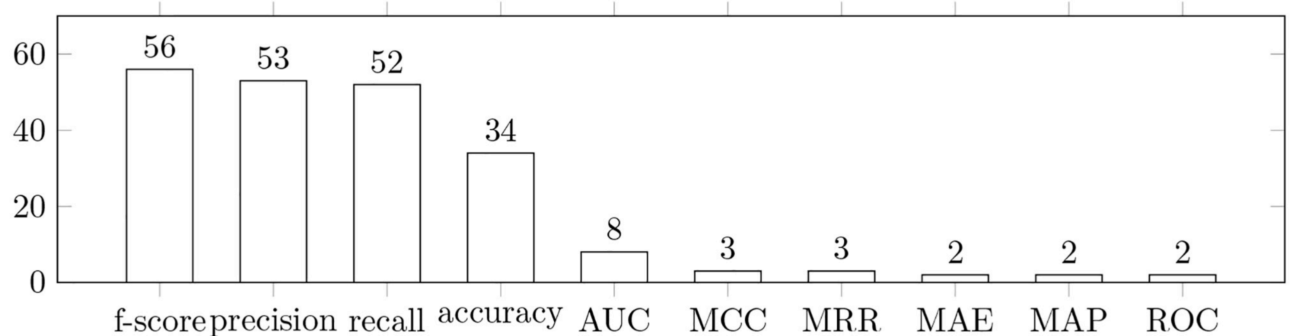


Fig 11. Metrics used in the papers included in the final reading log. The bars show the number of papers in which the specified metrics were used. Note: several metrics could be used in one paper. Acronyms used: AUC—area under the ROC curve, MCC—Matthews Correlation Coefficient, MRR—mean reciprocal rank, MAE—mean absolute error, MAP—mean average precision, ROC—receiver operating characteristic curve.

https://doi.org/10.1371/journal.pone.0283838.g011

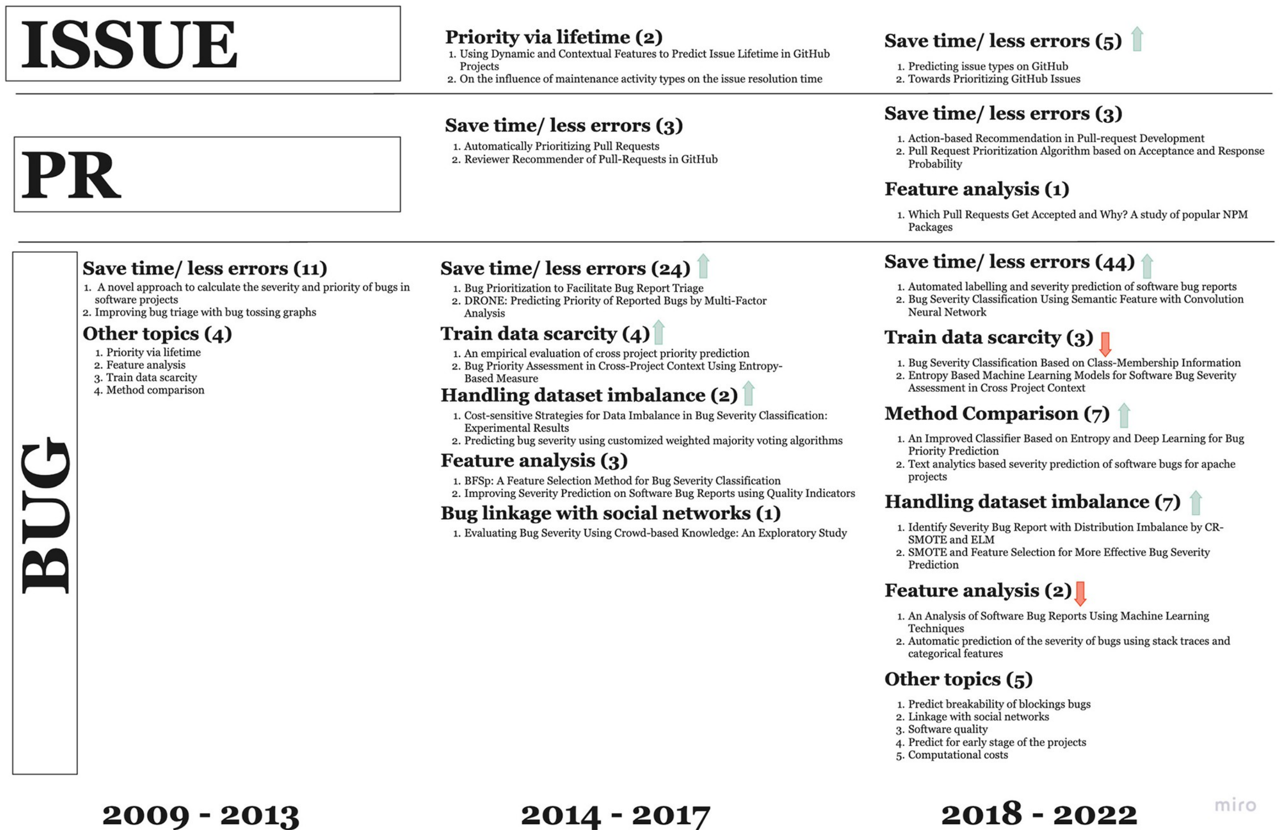


Fig 12. Dynamics of task prioritization research over time. The graph illustrates the chronological development of approaches to task prioritization from 2009 to 2022, specifically with respect to bugs, issues, and pull requests (PR). Each subtopic is denoted by bold text, followed by the total number of associated works. The figure is divided into three vertical fragments, each representing one of the subtopics. Green and red arrows indicate an increase or decrease in the number of publications, respectively.

https://doi.org/10.1371/journal.pone.0283838.g012

independent issues and discuss them below in order of appearance in the literature. Bugs prioritization originated first, and it involves prioritizing the bugs based on their severity and impact on the software system [52]. Issue prioritization is the process of selecting and ranking issues based on factors such as their importance and urgency [53]. Finally, Pull Request prioritization is the process of selecting and ranking pull requests based on their impact on the software system and their relationship with other pull requests [54]. Overall trends in the task prioritization field are shown in Fig 12.

Bug prioritization

In this work, we understand bugs as reports by users and developers about program components that do not function properly. The collection of attributes used to describe the reports is usually determined by the platform on which the report was created. The platform used to detect bugs in most cases is Bugzilla (see https://www.bugzilla.org/). In 2004, one of the first studies by Cubranic and Murphy [5] on bug prioritization was conducted. Although this paper has been highly influential in the literature, we did not include this study in our main log because it failed to satisfy one of our inclusion criteria (namely, the third criterion).

However, since 2004, which is the year in which this paper was published, the field of bug prioritization boomed, giving raise to many profitable investigations on: prioritization on

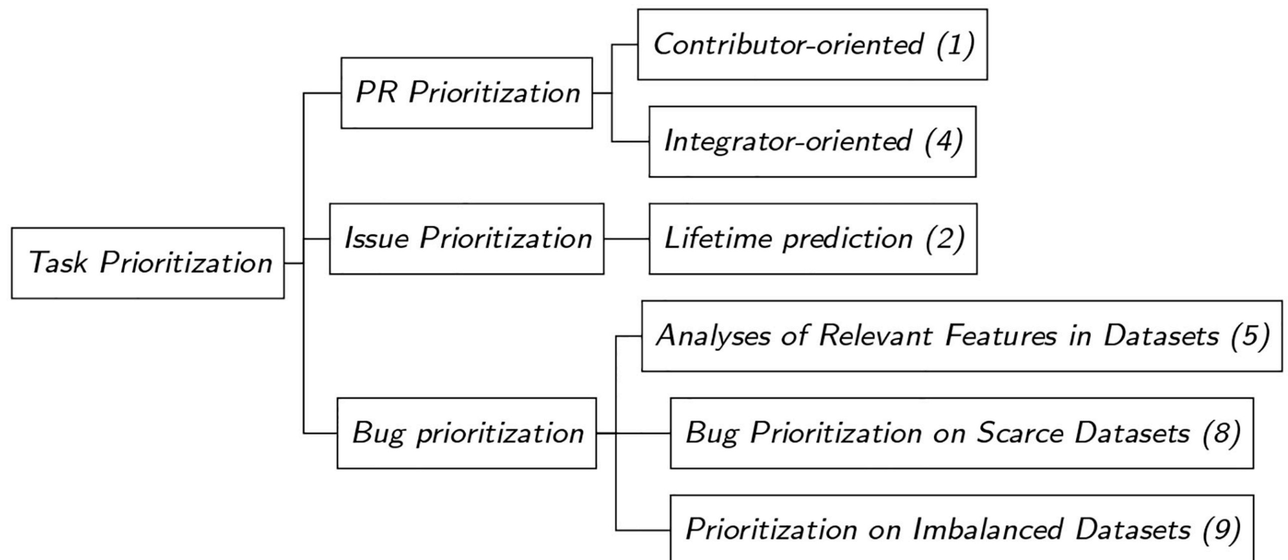


Fig 13. Branches of task prioritization. The numbers in brackets indicate the number of publications found for each branch.

https://doi.org/10.1371/journal.pone.0283838.g013

imbalanced datasets, prioritization in case of scarce datasets, and analyses concerning relevant features in datasets. Each of these areas deals with separate problems inherent to the field of bug prioritization.

Since we were unable to find precise causal relationships in the development of each of these directions, we describe them below on the basis of their popularity. The popularity of each direction is hereby determined by the number of scientific articles we found to be related to that specific direction. Fig 13 provides information about the popularity of each direction.

Prioritization on imbalance datasets. According to Fig 13, the most popular subtopic within bug prioritization is prioritization on imbalanced datasets. We believe that the popularity of this area is determined by the problem it subtends.

In brief, machine learning systems trained on imbalanced data will only perform well on samples with a dominant label. A dataset with a high number of low-priority bugs, for example, is more likely to classify subsequent bugs, even those with a high priority, as low priority. One of the researchers who also noted the importance of balancing the dataset is Thabtah [55]. Having explained the possible reasons for the popularity of this topic, we next move on to analyze its general structure as well as some of the most representative works we found that relate to it.

In light of the data we gathered, we can divide bug prioritization on imbalance datasets into two categories, based on the specific (machine learning) techniques used: those that use one predictor and those that use several predictors (what is known as the ensemble approach).

An example of work belonging to the former category is the work of Singha and Rossi [56]. The authors of this work used a modified version of Support Vector Machine (SVM) to weight classes based on the inverse occurrence of class frequencies. The results suggest that the model provides better prediction quality than standard SVM. Another example of the approach predicated by the former category is the work of Guo et al. [57]. In this study, the authors used Extreme Learning Machine (ELM) as a predictor. Several oversampling strategies were also tested.

The findings of this study are interesting for the purpose of this study because they indicate that the suggested approach can effectively balance an imbalanced dataset, which can contribute to increase the accuracy of bug prioritization.

Methods using a single predictor to deal with bug prioritization on imbalance datasets, as previously indicated, are not the only ones available. An example of work using multiple predictors (hence belonging to the latter category above-mentioned) is the work of Awad et al. [58].

The authors of this work proposed using the so-called ensemble method, in which each category of bug has its own predictor plus an additional general predictor for any type of bug. The topic of ensemble methods was described in more detail by Optiz and Maclin [59]. The peculiarity of the method is that any machine learning technique can be used as a predictor; the authors of the paper tested several techniques (such as Nave Bayes Multinomial (NBM), Random Forest (RF), and SVM). They also evaluated their proposed approach, which used both textual-based and non-textual datasets. Results showed that the proposed method can be successfully used to improve classical methods; however, this could be done only in the presence of a textual dataset.

Bug prioritization on scarce datasets. A second subtopic we found within bug prioritization is prioritization in case of scarce data. Research in this area typically attempts to formulate methods capable of showing consistent and accurate results despite the availability of a relatively small amount of training data. The first work we found for bug prioritization in case of scarce data is the work of Sharma et al. [11], which was published in 2012. Several machine learning techniques, like SVM, Naive Bayes (NB), Neural Networks (NN), K-Nearest Neighbours (KNN), were tested to ascertain the best suitable and most accurate among them all. The authors showed that overall SVM and NN produce better results.

It is nevertheless worth noting that M. Sharma is the primary contributor to the topic, having published 5 of the 8 papers we found. We note that in this research group repeatedly utilized the same set of machine learning techniques (such as SVM, NB, NN, and KNN) [11, 60, 61]. We, therefore, acknowledge that this may lead to biased conclusions.

We also note that the techniques listed above are not all the techniques that are currently applied, used, or tested in the literature. For example, Zhang et al. [62] proposed using ELM, while Hernández-González et al. [63] used the Expectation Maximization (EM) Algorithm.

Analyses of relevant features in datasets. This brings us to the discussion of the last subtopic within bug prioritization: analyses of relevant features in datasets. The goal of researchers in this field is to identify a set of features within a dataset that will yield the highest accuracy for a model trained on such data.

Although the topic is well-researched, there is still no consensus on the optimal set of attributes to be used. For example, the first publication in the domain by Alenezi and Banitaan [64] indicates that meta-data attributes are more relevant than textual description features. Sharmin et al. [65] also investigate the significance of features; however, they only compare two fields (text description and text conclusion).

Another perspective on the relevance and significance of features/attributes is offered by Sabor et al. [66]. The authors of this article proposed using stack traces as well as attributes, that were discussed by Alenezi and Banitaan [64]. More recently, a few works explored new ways for supplementing datasets with social-media information, for example, the work of Zhang et al. [67].

In light of the evidence reviewed above, we believe that the wide range of techniques and opinions developed in the literature thus far makes the task of identifying optimal qualities considerably challenging. Hence, we fear we are not in a position to make any specific recommendation with respect to this subtopic.

Issue prioritization

An issue (see https://docs.github.com/en/issues) is an object that describes the work and the prerequisites for completing it. Any member of the open-source community can create an

issue in order to enhance any given product. Issues in software development are typically found in platforms like GitHub, GitLab, or Bitbucket. Because of the novelty of these platforms, the subject has received little attention from the research community.

The main approach for dealing with issue prioritization has been that of predicting the lifetime of the issue itself. This approach was initially discussed by Murgia et al. [53]. The authors of this paper also investigate the impact of different types of developers' activities (such as maintenance type, adding a new feature, refactoring, etc) on issue resolution time (or issue lifetime). These activities are often represented with labels. The results of this study show that fixing defects and implementing/improving new features is more effective and typically less time-consuming than other activities (such as testing or documenting).

The idea of using labels to represent activities also inspired other authors, such as Kikas et al. [68]. Subsequent work by Kallis et al. [69] confirmed the potential of this research direction and analyzed the relationship between static/dynamic features and issues' lifetime.

Static features are those that remain consistent over time (for example, the number of issues created by the issue submitter in the three months before opening the issue). Dynamic features on the contrary are those features that change depending on when an observation is made (for example, if we look at the number of comments on an issue, we can see how it changes over time).

Another work that attempts to resolve issues prioritization by utilizing the concept of issue-lifetime prediction is the work of Dhasade et al. [70]. The authors of this work continued to use both static and dynamic attributes. They also expanded the previously developed approach by including in the model (and subsequently testing within it) various other hyperparameters (such as time and hotness). The changes implemented in the model by these researchers made the model more flexible and therefore capable of being adjusted to the needs of different teams.

As previously stated, the majority of articles predict the priority based on the expected issue lifetime. A slightly different strategy is however demonstrated by Kallis et al. [69], where labels are used by the authors to assist developers in the organization of their work (hence prioritizing their tasks). The method developed in this study can correctly and reliably anticipate one of three labels: bug, enhancement, or question.

PR prioritization

We cannot fully understand task prioritization if we do not discuss the third category that falls within it; namely, PR Prioritization. Research on PR Prioritization may be divided into two sub-topics, as shown in Fig 13: integrator-oriented research and contributor-oriented research. An integrator is someone who is in charge of reviewing PRs, whereas a contributor is someone who creates PRs.

In the former category (integrator-oriented research), we may include the works of [54, 71–73]. Van der Veen [71] offered a tool for prioritizing PRs based on static and dynamic attributes. This type of approach is quite similar to the work of Dhasade et al. [70]. In fact, Dhasade et al. [70] were inspired by this work and used it, as we have seen above, as a conceptual palette for their investigation.

A study by Yu et al. [54] proposes another approach to improving PR prioritization. The approach revolves around the idea of recommending appropriate reviewers to PRs. A description of the PR and a comment-network are two of the most crucial features used in this model. The comment network is a graph that is constructed based on the developers' shared interests. Results from this study show that the method is capable of obtaining a 71 percent precision in predicting the appropriate reviewer.

Another method, that has similar goals, is discussed by Yu et al. [72]. The researchers developed an approach that is intended to aid in the prioritization of PRs, by forecasting their latency (i.e., evaluation time). To make such a prediction, the researchers took into consideration numerous socio-technical parameters (such as project age, team size, and total CI). The findings demonstrated that the length of the dialogue (the number of comments under the PR) had a substantial impact on its latency.

With respect to the latter category we introduced above (contributor-oriented research), we only found one relevant study by Azeem et al. [74]. In this study, the authors not only investigated the impact of each individual variable on the probability of a PR being merged, but they also formulated and developed a model capable of automatically estimating such a probability.

To obtain these results, the researchers used the XGBoost algorithm and over 50 different attributes. The mean average precision of their model for the first five recommended PRs was 95.3%, hovered at 89.6% for the first ten PRs, and eventually decreased to 79.6% for the first twenty PRs. The results show that the technique outperformed the baseline model presented by Gousios et al. [75] at all levels (for the first five, the first ten, and the first twenty PRs).

RQ2. Which methods are used in automatic task ranking models and approaches and how is their effectiveness assessed?

As we noted in the previous section, giving an exact definition of a task can be quite challenging, at least in software development. In our research, we found that the same approaches are usually employed to solve prioritization tasks for each of those different attributions. In other words, the observations we made for one meaning of the term invariably apply to the others. We speculate that the reason for this might be the presence of some sort of common fields or attributes between all these different meanings.

On these grounds and in light of the data presented in Fig 9, we can conclude that Naive Bayes is the most frequently used technique for solving the problem of predicting bug severity and priority.

In this context, it is essential to note that, despite the growing popularity of neural network approaches in other areas of computer science, we did not entirely observe the same popularity in the studies we selected. We believe that the relative unpopularity of neural networks and deep neural networks [76] might be caused by the relatively small size of the dataset. Even though neural networks are very powerful tools, they require a lot of training to process data properly, which was not always possible, for different reasons, in the papers we reviewed.

As we pointed out in Fig 11, the most used metrics for assessing the effectiveness of models are: f-score, precision, recall, and accuracy. This may indicate that the skewness of the datasets was relatively low [77]; however, closer scrutiny [60, 64, 78] reveals that this is not the case. Only nine studies employ the f-score as the only assessment metric. We can infer that in most cases additional metrics such as precision and recall are used in combination with the f-score to provide a complete and fair characterization of the model's quality.

Based on the results shown in Further Clustering we can make another important observation. Among the most popular metrics observed in our review, we noticed that there were not any of those typically used for recommender systems [79, 80]. Even though the question of task priority has a recommender nature, i.e., we want to know "what are the next tasks/features/bugs and how should be solved"; recommender ML techniques are not so popular. We believe that one of the reasons for that is that recommender system approaches [81] have only recently captured researchers' attention. This speculation is reinforced by the observation concerning the number of papers published under the search query "recommender system" for the period 2010–2022 (data gathered from Scopus, Fig 14). The development of recommender

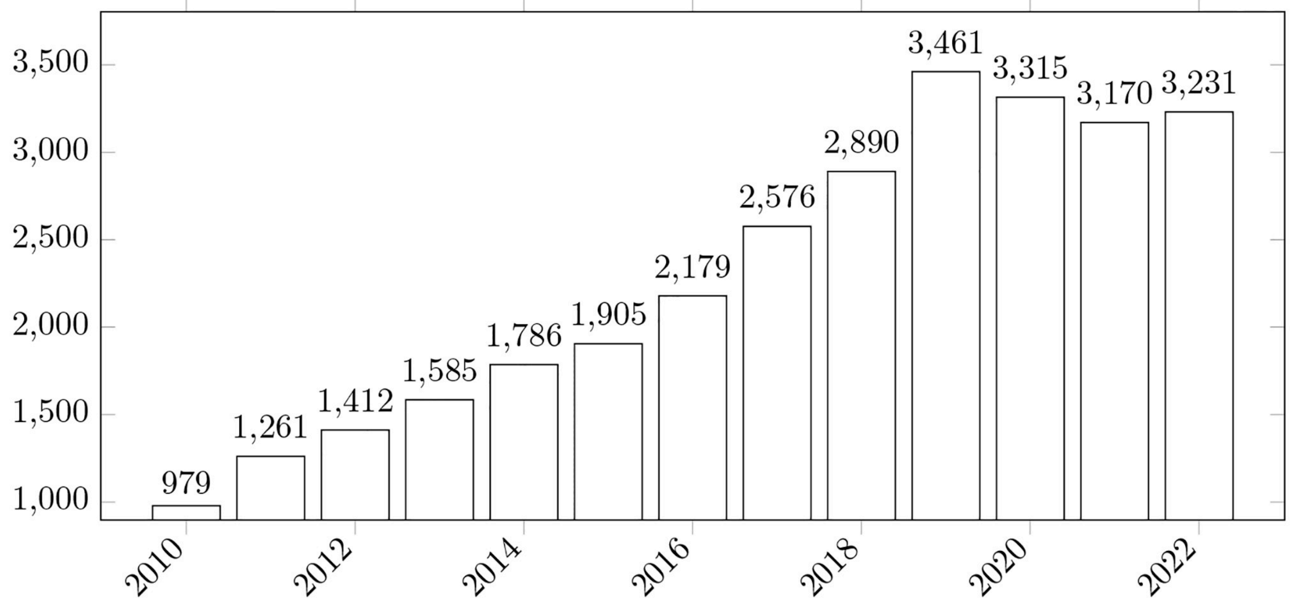


Fig 14. Papers distribution according to the query "recommender system" in scopus. The bars show the number of papers on the topic published between 2010 and 2022.

https://doi.org/10.1371/journal.pone.0283838.g014

systems can be a profitable way to guide existing works and the orientation of recommender systems can be used to gauge and support effective decision-making. This means that a model trained in this fashion cares about the ordering of output variables. So, if we have multiple entities and need only a small subset of the best of them, we most probably need to look at the solutions offered by the recommender system.

RQ3. What are the most effective and versatile models for automatic task ranking developed so far?

Giving a proper answer to the research questions proved to be more challenging due to the high variability of datasets and metrics used, and it may also depend upon the output variables selected (e.g. datasets). If we want to obtain reliable results, we should therefore compare results obtained on the same dataset [82]. Eclipse was the dataset of our choice. We chose Eclipse because it's the most popular dataset, according to the findings we presented in our results section. We must also note that if the authors made a comparison and/or had multiple models in one of their works, we decided to consider only one model, the one with better performance and the highest results. The result of our comparisons is given in Tables 6 and 7.

Table 6. Comparison of methods predicting priority. Priority levels are from Bugzilla [47]. The performance is described as precision/recall/f-score with the best results highlighted in bold. All data are shown as percentages.

Priority levels	Fang et al. [83]	Tian et al. [12]	Kanwal and Maqbool [84]	Pushpalatha et al. [85]	Choudhary and Singh [86]
p1	17 / 35 / 22	0 / 0 / 0	29 /43 / -	**100** / **97** / **98**	74 / 49 / 59
p2	19 / 32 / 24	0 / 0 / 0	29 / 84 / -	**86** / **85** / **85**	74 / 64 / 68
p3	89 / 67 / 77	88 / **100** / 94	**97** / 53 / -	75 / 87 / 80	94 / 98 / **96**
p4	11 / **30** / 16	0 / 0 / 0	37 / 24 / -	48 / 28 / **35**	**52** / 25 / 34
p5	15 / 38 / 21	0 / 0 / 0	16 / 46 / -	**67** / 67 / 67	60 / **85** / **70**

https://doi.org/10.1371/journal.pone.0283838.t006

Table 7. Comparison of methods predicting severity. Severity levels are from Bugzilla [46]. The performance is described as precision/recall/f-score with the best results highlighted in bold. All data are shown as percentages.

Severity levels	Zhang et al. [87]	Tian et al. [88]	Hamdy and El-Laithy [89]	Sharmin et al. [65]	Pundir et al. [90]	Zhang et al. [91]	Kukkar et al. [92]
blocker	64/78/78	25/27/26	28/26/27	-/-/15	**95**/**83**/**89**	31/25/27	80/75/78
critical	**86**/**91**/**89**	28/30/29	29/26/28	-/-/25	79/69/73	28/30/29	82/78/80
major	87/**96**/**91**	58/58/58	46/62/53	-/-/53	**95**/86/90	60/48/53	80/76/79
minor	97/**96**/97	42/38/40	39/30/34	-/-/31	**100**/95/**97**	43/45/44	80/77/79
trivial	78/88/**82**	28/25/27	42/15/22	-/-/29	68/**100**/81	19/42/26	**83**/79/81

https://doi.org/10.1371/journal.pone.0283838.t007

Table 6 shows a comparison between models with respect to their capacity for predicting bugs' priority. We note that even though the number of papers using the same dataset is much higher, the comparison table has only five elements. That is because some papers either used different levels of priorities and metrics or gave only graphical information, so no accurate value for the metric could be gauged.

As Table 6 shows, the work by Pushpalatha et al. [85] has the highest amount of highlighted cells. This makes it the best approach concerning the Eclipse dataset.

Table 7 shows a severity-wise comparison among all the approaches reviewed.

We note that the highest score is attributed to the approach proposed by [87], while the second-highest result is achieved by the model proposed by [90]. This paper, based on the Naive Bayes ML algorithm, is interesting because it formulated a method capable of ensuring the most accurate result for the blocker bug, which is one of the most severe types of bugs typically found. We also note that works by [87] demonstrated better outcomes for critical bugs.

Critical review of our research question

Because the analysis we conducted above showed a limited number of research articles related to task prioritization in software development, this prompted us to make several assumptions and partially expand the scope of our study. Since the word "task" can refer to multiple concepts, we decided to consider this word in its most general meaning. This allowed us to gather more articles for our analysis. However, the fact that we only gathered a relatively limited number of scientific articles related to prioritization may indicate a significant gap in the research field. On the one hand, the presence of this gap may be taken as a sign of the relevance and novelty of this study for the research field. On the other hand, our findings raise several significant and pressing concerns. For example, why has so little research been conducted in this area? What are the pitfalls in task prioritization research? Why there was no demand for such a system? While we do not have a clear answer to all these questions, we can nevertheless assert that this SLR highlighted the need for more research in this area (task prioritization in software development) while also forming a solid basis for future progress in the field.

Analysis of RQ1: What are the existing approaches for automatic task ranking in software development?

We can make several important observations about the results we obtained. Firstly, earlier work dealt mainly with the problem of "bug" prioritization, which, albeit useful, is neither exhaustive nor comprehensive. That is so because we are interested in a broader understanding of task prioritization.

Secondly, only recently (especially in the last 5 years, as shown in Fig 12) researchers began to pay attention to the concept of "pull requests" prioritization and "issues" prioritization,

which substantially expanded on original research conducted in the prioritization of Software Development metrics. As we discussed earlier on in this paper, this may well result in substantial growth of the literature in the near future, as it was the case with “bug prioritization” in the past.

Analysis of RQ2: Which methods are used in automatic task ranking models and approaches and how is their effectiveness assessed?

The number of methods described in this SLR for task prioritization is rather limited. The most popular method we observed is Naive Bayes. This method is important because it provides the most accurate result for the blocker bug. We also analyzed several different metrics found in the papers we reviewed, the most popular of which are: *a)* f-score, *b)* precision, *c)* recall, and *d)* accuracy.

Across the whole set of metrics, we found in the papers we reviewed for this SLR, CPU-costs related metrics are the rarest, which means that the question of computational costs has not been a priority. This may signal a new potential future research direction for task prioritization.

Based on the results presented above, we can argue that the lack of metrics commonly used in recommender systems represents an interesting research gap in the field, which is also shown in [93]. Our explanation for the existence of such a gap lies in the observation that there are still very few studies on recommender systems [81]. This is due, presumably, to the fact that recommender systems only recently attracted researchers’ attention.

Analysis of RQ3: What are the most effective and versatile models for automatic task ranking developed so far?

As we have shown above, the quality of task prioritization in software development has improved over the years as new and more accurate estimation methods have been deployed [87]. However, when it comes to prioritizing “pull requests” and “issues,” there are a lot of conflating strategies and ideas about what to prioritize [50]. Unfortunately, we have to admit that there isn’t a standardized or universally agreed approach for prioritizing such issues. Because of that, the only way to compare the approaches currently available is to fully reproduce and compare them on the same dataset and on the same set of metrics. This, however, is an extremely complicated task. Nevertheless, developing new works along this research direction may open up new vistas of vital importance for further progress in the field.

A synoptic summary

The brief synoptic summary of our results with respect to each of the research questions tackled in this study is the following:

RQ1 The number of articles that consider the problem of automatic task prioritization is still fairly small.

RQ2 Only a few articles, among those reviewed, assessed models in terms of time or CPU costs. Also, there is a sensitive lack of metrics used to analyze recommender systems.

RQ3 TSo far, there is no standardized approach or universal agreement on defining prioritization strategies for “pull requests” and “issues.” Presumably, this is because datasets are not marked up, meaning that there are no labels on data samples (see https://www.ibm.com/cloud/learn/data-labeling).

Limitations, threats to validity, and review assessment

Limitations

We start this subsection by briefly reviewing some of the obstacles that may have prevented an objective review.

In this study, we used five different databases, namely, *Google Scholar, Microsoft Academic, ScienceDirect, IEEE Xplore, and ACM*. A skeptical reader may point out that we should have used more databases (such as *Springer Link, Web of Science, Scopus, ProQuest, Academic OneFile*). We certainly are aware of the existence of many more databases (besides those we selected) and could even agree that a larger set of databases might have broadened the scope of our work and increased the diversity of our searches. However, we picked those databases most commonly used by software engineers worldwide and that are known to aggregate the largest possible variety of papers. So, even though the list of databases used to perform our searches could have been ameliorated, we are pretty confident that our searches were scientifically sound.

Also, one may see as perhaps problematic the fact that we did not use any kind of grey literature in this work. Although there is a tendency to advocate for the usage of multivocal approaches (such as grey literature) in software engineering, we believe that such practice (given the dubious nature of such literature) should be limited to cases where there is a sensible lack of secondary sources, which was not our case.

Finally, one may rightly claim that in picking only works written in English, we somehow constrained and severely limited the scope and breadth of this research. Cross-cultural issues are emerging as vitally important in ensuring universalism in science. We agree with the importance of adding multicultural perspectives and even under-represented works in any study; however, most of the literature in the field is in English, and all the best journals only accept submissions in English. We, therefore, deem that the requirement we adopted in this SLR concerning language is pretty standard for the field and relatively unproblematic. Nevertheless, we note that our team of researchers is culturally very diverse, as it includes people from four different continents.

Threats to validity

In this subsection, we discuss a series of biases that might have affected the development and production of our review.

- **Bias towards Primary Sources**—SLRs are usually performed on secondary sources. This is done to maximize objectivity. In this work, we uniquely relied on secondary sources; hence we avoided this potential bias.
- **Selection Bias**—A major risk involved in any SLR is what we may call "selective outcome reporting" or "selection bias." This typically occurs when the authors present only a selection of outcomes and/or results based on their statistical significance. We note that our reading log consists of only peer-reviewed, high-quality papers. The papers, as noted above, were published by world-leading publishers (such as Springer, Elsevier, ACM, and IEEE Xplore). In addition, we selected papers (methodologies, datasets, and metrics) from several journals as well as from reputable conference proceedings. This ensured a variety of levels of analysis and experimental protocols.
- **Bias in Synthesis**—To avoid this type of bias, which can be considered as an extension of the Selection Bias above-mentioned, we carefully assessed our methodological protocol and -by extension- our findings. Thus, all the researchers involved in this study actively and

consistently participated in monitoring each other's activity to maximize objectivity and minimize mistakes (such as this bias).

Review assessment

Finally, we want to reflect on the overall quality of our work. To do so, we formulated—inspired by Kitchenham [94], a set of questions, which we critically applied to our results and findings. The questions we formulated and the answers we gave to them follow below.

- **Are the inclusion / exclusion criteria objective and reasonable?** Following the best norms in our discipline, we formulated—before conducting our searches—a set of inclusion and exclusion criteria, which we subsequently applied to finalize the reading log. The criteria we formulated are congruent with those generally used in the field and are obviously relevant to the topic of our work.
- **Has there been a quality review?** We developed a metric to assess the papers' quality (Quality Assessment). We proved that the quality of the papers we included in the log was relatively high. Thus, we can confidently assert that the results that informed our work were scientifically sound and academically grounded.
- **Were the basic data / studies adequately described?** We build a comprehensive literature log. The log consisted of all the relevant information we extracted from the papers we analyzed. This allowed us to process our data transparently and comprehensively. It also ensured the replicability of our findings, which is another key trait of any SLR.

Conclusion

This SLR investigated the problem of task prioritization in software development and focused on: *a)* identifying existing approaches for automatic task prioritization (RQ1), *b)* further investigating methods and metrics for task prioritization as developed in the literature (RQ2), and *c)* analyzing the effectiveness and reliability of these methods and metrics (RQ3).

Concerning RQ1, our results showed that earlier work mainly dealt with bug prioritization, and more recent work has expanded to consider prioritizing pull requests and issues. We speculate that this may lead to a substantial growth of literature in the future. RQ2 revealed that the most popular method used for task prioritization is Naive Bayes, while the most popular metrics used (in descending order) are f-score, precision, recall, and accuracy. However, there is a lack of metrics used in recommender systems, which may indicate a potential direction for future research. RQ3 showed that the quality of task prioritization in software development has improved over time; however, there is still a sensible lack of standardized approaches for prioritizing pull requests and issues.

In light of these findings, we can assert that this SLR contributed to broadening the field of research on task prioritization in software development, while also providing a solid basis for future research. Our goal in the mid-term is to develop an empirical study based on the topic of this SLR. Our aim in such a study would be to find a practical way to implement the findings of this review. To this extent, we shall consider whether it could be possible to develop algorithms for predicting task prioritization in a project using ML methods. This may well lead to novel AI-based management strategies, which could improve people's well-being at work as well as foster moral and social good.

Nevertheless, IT practitioners should be cognizant of the relatively scarce amount of research conducted on task prioritization to date. They should also be aware of the absence of

established methods for prioritizing pull requests and issues. They should therefore use the results of this SLR as a springboard for further explorations aimed at the development of such methods and tools.

Supporting information

S1 Table. PRISMA 2020 checklist. Template is taken from: www.prisma-statement.org/documents/PRISMA_2020_checklist.pdf.
(PDF)

S2 Table. Quality scores assigned to the papers.
(PDF)

Author Contributions

Conceptualization: Yegor Bugayenko, Arina Cheverda, Mirko Farina, Yaroslav Plaksin, Giancarlo Succi.

Data curation: Ayomide Bakare, Yaroslav Plaksin.

Formal analysis: Yegor Bugayenko, Yaroslav Plaksin.

Funding acquisition: Giancarlo Succi.

Methodology: Yegor Bugayenko, Mirko Farina, Witold Pedrycz, Giancarlo Succi.

Software: Yegor Bugayenko, Artem Kruglov, Giancarlo Succi.

Supervision: Mirko Farina, Artem Kruglov, Witold Pedrycz, Giancarlo Succi.

Validation: Ayomide Bakare, Arina Cheverda.

Writing – original draft: Arina Cheverda, Mirko Farina, Artem Kruglov, Yaroslav Plaksin, Giancarlo Succi.

Writing – review & editing: Ayomide Bakare, Mirko Farina, Witold Pedrycz.

References

1. Kandemir C, Handley H. Employee Task Assignments for Organization Modeling: A Review of Models and Applications; 2014.
2. IEEE Standard for Software Project Management Plans;.
3. About issues; 2023. Available from: https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.
4. Anvik J. Automating Bug Report Assignment. In: Proceedings of the 28th International Conference on Software Engineering. ICSE'06. New York, NY, USA: Association for Computing Machinery; 2006. p. 937–940.
5. Cubranic D, Murphy G. Automatic bug triage using text categorization. SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. 2004; p. 92–97.
6. Zimmermann T, Premraj R, Bettenburg N, Just S, Schroter A, Weiss C. What Makes a Good Bug Report? IEEE Transactions on Software Engineering. 2010; 36(5):618–643. https://doi.org/10.1109/TSE.2010.63
7. Anvik J, Hiew L, Murphy GC. Who Should Fix This Bug? In: Proceedings of the 28th International Conference on Software Engineering. ICSE'06. New York, NY, USA: Association for Computing Machinery; 2006. p. 361–370.
8. Panjer LD. Predicting Eclipse Bug Lifetimes. In: Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007); 2007. p. 29–29.

9. Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of the 13th international conference on Software engineering—ICSE '08. ACM Press; 2008.
10. Menzies T, Marcus A. Automated severity assessment of software defect reports. In: 2008 IEEE International Conference on Software Maintenance; 2008. p. 346–355.
11. Sharma M, Bedi P, Chaturvedi KK, Singh VB. Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: 2012 12th International Conference on Intelligent Systems Design and Applications (ISDA). IEEE; 2012.
12. Tian Y, Lo D, Sun C. DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis. In: 2013 IEEE International Conference on Software Maintenance. IEEE; 2013.
13. Illes-Seifert T, Herrmann A, Geisser M, Hildenbrand T. The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey. In: Proceedings : 1st Global Requirements Engineering Workshop—Grew'07 : in conjuction with the IEEE Conference on Global Software Engineering (ICGSE), Germany, Munich, 27th August 2007; 2007. p. 55–65.
14. Niazi M, Mahmood S, Alshayeb M, Riaz MR, Faisal K, Cerpa N, et al. Challenges of project management in global software development: A client-vendor analysis. Information and Software Technology. 2016; 80:1–19. https://doi.org/10.1016/j.infsof.2016.08.002
15. Demir K. A Survey on Challenges of Software Project Management. In: Software Engineering Research and Practice; 2009. p. 579–585.
16. Gousios G, Storey MA, Bacchelli A. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE); 2016. p. 285–296.
17. Gousios G, Zaidman A, Storey MA, Deursen Av. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. vol. 1; 2015. p. 358–368.
18. Barr ET, Bird C, Rigby PC, Hindle A, German DM, Devanbu P. Cohesive and Isolated Development with Branches. In: Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg; 2012. p. 316–331.
19. Jordan M, Mitchell TM. Machine Learning: Trends, Perspectives, and Prospects. Science (New York, NY). 2015; 349(10):255–60. https://doi.org/10.1126/science.aaa8415 PMID: 26185243
20. Wang H, Ma C, Zhou L. A Brief Review of Machine Learning and Its Application. In: 2009 International Conference on Information Engineering and Computer Science. IEEE; 2009.
21. Achimugu P, Selamat A, Ibrahim R, Mahrin MN. A systematic literature review of software requirements prioritization research. Information and Software Technology. 2014; 56(6):568–585. https://doi.org/10.1016/j.infsof.2014.02.001
22. Bukhsh FA, Bukhsh ZA, Daneva M. A systematic literature review on requirement prioritization techniques and their empirical evaluation. Computer Standards & Interfaces. 2020; 69:103389. https://doi.org/10.1016/j.csi.2019.103389
23. Somohano-Murrieta JCB, Ocharan-Hernandez JO, Sanchez-Garcia AJ, de los Angeles Arenas-Valdes M. Requirements Prioritization Techniques in the last decade: A Systematic Literature Review. In: 2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT). IEEE; 2020.
24. Rashdan A. Requirement prioritization in Software Engineering : A Systematic Literature Review on Techniques and Methods; 2021.
25. Sufian M, Khan Z, Rehman S, Butt WH. A Systematic Literature Review: Software Requirements Prioritization Techniques. In: 2018 International Conference on Frontiers of Information Technology (FIT). IEEE; 2018.
26. Kaur A, Jain S, Goel S, Dhiman G. WITHDRAWN: Prioritization of code smells in object-oriented software: A review. Materials Today: Proceedings. 2021. https://doi.org/10.1016/j.matpr.2020.11.218
27. Alfayez R, Alwehaibi W, Winn R, Venson E, Boehm B. A systematic literature review of technical debt prioritization. In: Proceedings of the 3rd International Conference on Technical Debt. ACM; 2020.
28. Ijaz KB, Inayat I, Bukhsh FA. Non-functional Requirements Prioritization: A Systematic Literature Review. In: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE; 2019.
29. Pasikanti N, Kawaf C. Bugs Prioritization in Software Engineering : A Systematic Literature Review on Techniques and Methods; 2022.
30. Pan R, Bagherzadeh M, Ghaleb TA, Briand L. Test case selection and prioritization using machine learning: a systematic literature review. Empirical Software Engineering. 2021; 27(2). https://doi.org/10.1007/s10664-021-10066-6

31. Bajaj A, Sangwan OP. A Systematic Literature Review of Test Case Prioritization Using Genetic Algorithms. IEEE Access. 2019; 7:126355–126375. https://doi.org/10.1109/ACCESS.2019.2938260

32. Filho MS, Pinheiro PR, Albuquerque AB, Rodrigues JJPC. Task Allocation in Distributed Software Development: A Systematic Literature Review. Complexity. 2018; 2018:1–13. https://doi.org/10.1155/2018/6071718

33. Fatima T, Azam F, Anwar MW, Rasheed Y. A Systematic Review on Software Project Scheduling and Task Assignment Approaches. In: Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence. ACM; 2020.

34. Farina M, Kostin M, Succi G. Interest identification from browser tab titles: A systematic literature review. Computers in Human Behavior Reports. 2022; 7:100187. https://doi.org/10.1016/j.chbr.2022.100187

35. Farina M, Fedorovskaya A, Polivtsev E, Succi G. Software Engineering and Filmmaking: A Literature Review. Frontiers in Computer Science. 2022; 4. https://doi.org/10.3389/fcomp.2022.884533

36. Ciancarini P, Farina M, Okonicha O, Smirnova M, Succi G. Software as storytelling: A systematic literature review. Computer Science Review. 2023; 47:100517. https://doi.org/10.1016/j.cosrev.2022.100517

37. Xiao Y, Watson M. Guidance on Conducting a Systematic Literature Review. Journal of Planning Education and Research. 2017; 39(1):93–112. https://doi.org/10.1177/0739456X17723971

38. Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. BMJ. 2021; p. n71. https://doi.org/10.1136/bmj.n71 PMID: 33782057

39. Kitchenham BA, Charters S. Guidelines for performing Systematic Literature Reviews in Software Engineering. Keele University and Durham University Joint Report; 2007. Available from: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

40. Laranjeiro N, Agnelo J, Bernardino J. A Systematic Review on Software Robustness Assessment. ACM Computing Surveys. 2021; 54(4):1–65. https://doi.org/10.1145/3448977

41. AbuHassan A, Alshayeb M, Ghouti L. Software smell detection techniques: A systematic literature review. Journal of Software: Evolution and Process. 2020; 33(3). https://doi.org/10.1002/smr.2320

42. Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software. 2007; 80(4):571–583. https://doi.org/10.1016/j.jss.2006.07.009

43. Basili VR, Caldiera G, Rombach HD. The goal question metric approach. Encyclopedia of software engineering. 1994; p. 528–532.

44. Patino CM, Ferreira JC. Inclusion and exclusion criteria in research studies: definitions and why they matter. Jornal Brasileiro de Pneumologia. 2018; 44:84–84. https://doi.org/10.1590/S1806-37562018000000088 PMID: 29791550

45. Shoham Y, Perrault R, Brynjolfsson E, Clark J, Manyika J, Niebles JC, et al.. The AI Index 2018 Annual Report; 2018. Available from: https://hai.stanford.edu/sites/default/files/2020-10/AI_Index_2018_Annual_Report.pdf.

46. Lauhakangas I, Raal R, Iversen J, Tryon R, Goncharuk L, Roczek D. QA/Bugzilla/Fields/Severity; 2021. Available from: https://wiki.documentfoundation.org/QA/Bugzilla/Fields/Severity.

47. Kanat-Alexander M, Miller D, Humphries E. Bugzilla:Priority System; 2021. Available from: https://wiki.mozilla.org/Bugzilla:Priority_System.

48. Sharma G, Sharma S, Gujral S. A Novel Way of Assessing Software Bug Severity Using Dictionary of Critical Terms. Procedia Computer Science. 2015; 70:632–639. https://doi.org/10.1016/j.procs.2015.10.059

49. Yang G, Zhang T, Lee B. Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports. In: 2014 IEEE 38th Annual Computer Software and Applications Conference. IEEE; 2014.

50. Freedman D, Pisani R, Purves R. Statistics (international student edition). Pisani, Purves R, 4th edn WW Norton & Company, New York. 2007;.

51. Bayes T. An essay towards solving a problem in the doctrine of chances. Phil Trans of the Royal Soc of London. 1763; 53:370–418.

52. Jeong G, Kim S, Zimmermann T. 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. ACM; 2009.

53. Murgia A, Concas G, Tonelli R, Ortu M, Demeyer S, Marchesi M. On the influence of maintenance activity types on the issue resolution time. In: Proceedings of the 10th International Conference on Predictive Models in Software Engineering. ACM; 2014.

54. Yu Y, Wang H, Yin G, Ling CX. Reviewer Recommender of Pull-Requests in GitHub. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE; 2014.
55. Thabtah F, Hammoud S, Kamalov F, Gonsalves A. Data imbalance in classification: Experimental evaluation. Information Sciences. 2020; 513:429–441. https://doi.org/10.1016/j.ins.2019.11.004
56. Roy NKS, Rossi B. Cost-Sensitive Strategies for Data Imbalance in Bug Severity Classification: Experimental Results. In: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE; 2017.
57. Guo S, Chen R, Li H, Zhang T, Liu Y. Identify Severity Bug Report with Distribution Imbalance by CR-SMOTE and ELM. International Journal of Software Engineering and Knowledge Engineering. 2019; 29 (02):139–175. https://doi.org/10.1142/S0218194019500074
58. Awad MA, ElNainay MY, Abougabal MS. Predicting bug severity using customized weighted majority voting algorithms. In: 2017 Japan-Africa Conference on Electronics, Communications and Computers (JAC-ECC). IEEE; 2017.
59. Opitz D, Maclin R. Popular Ensemble Methods: An Empirical Study. J Artif Int Res. 1999; 11(1):169–198.
60. Sharma M, Bedi P, Singh V. An empirical evaluation of cross project priority prediction. International Journal of System Assurance Engineering and Management. 2014; 5. https://doi.org/10.1007/s13198-014-0219-4
61. Sharma M, Kumari M, Singh VB. Bug Priority Assessment in Cross-Project Context Using Entropy-Based Measure. In: Algorithms for Intelligent Systems. Springer Singapore; 2020. p. 113–128.
62. Zhang TL, Chen R, Yang X, Zhu HY. An uncertainty based incremental learning for identifying the severity of bug report. International Journal of Machine Learning and Cybernetics. 2019; 11(1):123–136. https://doi.org/10.1007/s13042-019-00961-2
63. Hernández-González J, Rodriguez D, Inza I, Harrison R, Lozano JA. Learning to classify software defects from crowds: A novel approach. Applied Soft Computing. 2018; 62:579–591. https://doi.org/10.1016/j.asoc.2017.10.047
64. Alenezi M, Banitaan S. Bug Reports Prioritization: Which Features and Classifier to Use? In: 2013 12th International Conference on Machine Learning and Applications. IEEE; 2013.
65. Sharmin S, Aktar F, Ali AA, Khan MAH, Shoyaib M. BFSp: A feature selection method for bug severity classification. In: 2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC). IEEE; 2017.
66. Sabor KK, Hamdaqa M, Hamou-Lhadj A. Automatic prediction of the severity of bugs using stack traces and categorical features. Information and Software Technology. 2020; 123:106205. https://doi.org/10.1016/j.infsof.2019.106205
67. Zhang Y, Yin G, Wang T, Yu Y, Wang H. Evaluating Bug Severity Using Crowd-based Knowledge. In: Proceedings of the 7th Asia-Pacific Symposium on Internetware. ACM; 2015.
68. Kikas R, Dumas M, Pfahl D. Using dynamic and contextual features to predict issue lifetime in github projects. In: 2016 ieee/acm 13th working conference on mining software repositories (msr). IEEE; 2016. p. 291–302.
69. Kallis R, Sorbo AD, Canfora G, Panichella S. Predicting issue types on GitHub. Science of Computer Programming. 2021; 205:102598. https://doi.org/10.1016/j.scico.2020.102598
70. Dhasade AB, Venigalla ASM, Chimalakonda S. Towards Prioritizing GitHub Issues. In: Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference. ACM; 2020.
71. van der Veen E, Gousios G, Zaidman A. Automatically Prioritizing Pull Requests. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE; 2015.
72. Yu Y, Wang H, Filkov V, Devanbu P, Vasilescu B. Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE; 2015.
73. Yu S, Xu L, Zhang Y, Wu J, Liao Z, Li Y. NBSL: A Supervised Classification Model of Pull Request in Github. In: 2018 IEEE International Conference on Communications (ICC). IEEE; 2018.
74. Azeem MI, Peng Q, Wang Q. Pull Request Prioritization Algorithm based on Acceptance and Response Probability. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). IEEE; 2020.
75. Gousios G, Pinzger M, van Deursen A. An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering. ACM; 2014.
76. Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences. 1982; 79(8):2554–2558. https://doi.org/10.1073/pnas.79.8.2554 PMID: 6953413

77. Jeni LA, Cohn JF, Torre FDL. Facing Imbalanced Data–Recommendations for the Use of Performance Metrics. In: 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction. IEEE; 2013.

78. Zhang W, Challis C. Automatic Bug Priority Prediction Using DNN Based Regression. In: Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery. Springer International Publishing; 2019. p. 333–340.

79. Herlocker JL, Konstan JA, Terveen LG, Riedl JT. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems. 2004; 22(1):5–53. https://doi.org/10.1145/963770.963772

80. Farina M, Gorb A, Kruglov A, Succi G. Technologies for GQM-Based Metrics Recommender Systems: A Systematic Literature Review. IEEE Access. 2022; 10:23098–23111. https://doi.org/10.1109/ACCESS.2022.3152397

81. Ricci F, Rokach L, Shapira B. Introduction to Recommender Systems Handbook. In: Recommender Systems Handbook. Springer US; 2010. p. 1–35.

82. Lamkanfi A, Perez J, Demeyer S. The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information. In: 2013 10th Working Conference on Mining Software Repositories (MSR). IEEE; 2013.

83. Fang S, shuai Tan Y, Zhang T, Xu Z, Liu H. Effective Prediction of Bug-Fixing Priority via Weighted Graph Convolutional Networks. IEEE Transactions on Reliability. 2021; 70(2):563–574. https://doi.org/10.1109/TR.2021.3074412

84. Kanwal J, Maqbool O. Managing Open Bug Repositories through Bug Report Prioritization Using SVMs. 123. 2010;.

85. Pushpalatha MN, Mrunalini M, Sulav Raj B. Predicting the Priority of Bug Reports Using Classification Algorithms. Indian Journal of Computer Science and Engineering. 2020; 11(6):811–818. https://doi.org/10.21817/indjcse/2020/v11i6/201106076

86. Choudhary PA, Singh S. Neural Network Based Bug Priority Prediction Model using Text Classification Techniques. International Journal of Advanced Research in Computer Science. 2017; 8(5).

87. Zhang T, Yang G, Lee B, Chan ATS. Predicting severity of bug report by mining bug repository with concept profile. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM; 2015.

88. Tian Y, Lo D, Sun C. Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction. In: 2012 19th Working Conference on Reverse Engineering. IEEE; 2012.

89. Hamdy A, El-Laithy A. SMOTE and Feature Selection for More Effective Bug Severity Prediction. International Journal of Software Engineering and Knowledge Engineering. 2019; 29(06):897–919. https://doi.org/10.1142/S0218194019500311

90. Pundir P, Singh S, Kaur G. A Machine Learning Based Bug Severity Prediction using Customized Cascading Weighted Majority Voting. International Journal of Computer Sciences and Engineering. 2019; 7:1345–1350. https://doi.org/10.26438/ijcse/v7i5.13451350

91. Zhang T, Chen J, Yang G, Lee B, Luo X. Towards more accurate severity prediction and fixer recommendation of software bugs. Journal of Systems and Software. 2016; 117:166–184. https://doi.org/10.1016/j.jss.2016.02.034

92. Kukkar A, Mohana R, Kumar Y. Does Bug Report Summarization Help in Enhancing the Accuracy of Bug Severity Classification? Procedia Computer Science. 2020; 167:1345–1353. https://doi.org/10.1016/j.procs.2020.03.345

93. Happel HJ, Maalej W. Potentials and challenges of recommendation systems for software development. In: Proceedings of the 2008 international workshop on Recommendation systems for software engineering. ACM; 2008.

94. Kitchenham B. Procedures for Performing Systematic Reviews; 2004.