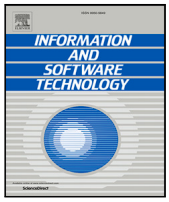
Information and Software Technology 164 (2023) 107305

Contents lists available at ScienceDirect

Information and Software Technology

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



A data-driven approach for understanding invalid bug reports: An industrial case study

Muhammad Laiq [a,*], Nauman bin Ali [a], Jürgen Börstler [a], Emelie Engström [b]

[a] Blekinge Institute of Technology, Department of Software Engineering, SE-371 79, Karlskrona, Sweden
[b] Lund University, Department of Software Engineering, SE-221 00, Lund, Sweden

ARTICLE INFO

Keywords:
Software maintenance
Invalid bug reports
Bug management
Topic modeling
LDA
Bug classification
Software analytics

ABSTRACT

Context: Bug reports created during software development and maintenance do not always describe deviations from a system's valid behavior. Such invalid bug reports may consume significant resources and adversely affect the prioritization and resolution of valid bug reports. There is a need to identify preventive actions to reduce the inflow of invalid bug reports. Existing research has shown that manually analyzing invalid bug report descriptions provides cues regarding preventive actions. However, such a manual approach is not cost-effective due to the time required to analyze a sufficiently large number of bug reports needed to identify useful patterns. Furthermore, the analysis needs to be repeated as the underlying causes of invalid bug reports change over time.

Objective: In this study, we propose and evaluate the use of Latent Dirichlet Allocation (LDA), a topic modeling approach, to support practitioners in suggesting preventive actions to avoid the creation of similar invalid bug reports in the future.

Method: In an industrial case study, we first manually analyzed descriptions of invalid bug reports to identify common patterns in their descriptions. We further investigated to what extent LDA can support this manual process. We used expert-based validation to evaluate the relevance of identified common patterns and their usefulness in suggesting preventive measures.

Results: We found that invalid bug reports have common patterns that are perceived as relevant, and they can be used to devise preventive measures. Furthermore, the identification of common patterns can be supported with automation.

Conclusion: Using LDA, practitioners can effectively identify representative groups of bug reports (i.e., relevant common patterns) from a large number of bug reports and analyze them further to devise preventive measures.

1. Introduction

Bug management is a costly and complicated process comprising activities such as reporting, assigning, and resolving bug reports [1,2]. In large projects, large amounts of bug reports are submitted daily [3–5] to describe erroneous behaviors of a software system. In the next step, developers use the submitted information to recreate the issue, identify root causes, and fix bugs. However, in many cases, bug reports do not describe erroneous system behavior. Such bug reports are called 'invalid.'

Invalid bug reports consume resources and time and make the prioritization and identification of valid bug reports difficult. In our previous work [6], we found that around 15% of all bug reports for two large products were invalid and that the resolution time for invalid bug reports was similar to the resolution time for valid bug reports. Likewise, Erfani Joorabchi et al. [7] found that non-reproducible bug reports (a type of invalid bug reports) remain active for more than three months and are treated similarly (i.e., in terms of the extent of discussion or number of people involved) as other types of bug reports. The prevalence, resolution time, and amount of resources consumed indicate the importance of addressing the problem.

Therefore, there is a need to identify preventive actions to reduce the inflow of invalid bug reports. The preventive actions relevant for a company at a particular time will depend on the nature of prevalent causes of invalid bug reports. Existing research has shown that manually analyzing invalid bug report descriptions provides cues regarding preventive actions [7–11]. However, due to the large number of bug

* Corresponding author.
E-mail addresses: muhammad.laiq@bth.se (M. Laiq), nauman.ali@bth.se (N.b. Ali), jurgen.borstler@bth.se (J. Börstler), emelie.engstrom@cs.lth.se (E. Engström).

https://doi.org/10.1016/j.infsof.2023.107305
Received 3 February 2023; Received in revised form 10 July 2023; Accepted 24 July 2023
Available online 28 July 2023

reports that need to be analyzed and the need for repeating such an analysis after any major change in the process, product, technology, or personnel, a manual approach is not cost-effective.

In this study, we propose and evaluate the use of topic modeling to identify common patterns in invalid bug reports. We use Latent Dirichlet Allocation (LDA) to automatically generate topics from the descriptions of invalid bug reports. Practitioners can then further analyze relevant topics to suggest preventive measures to avoid creating similar invalid bug reports in the future.

In software engineering, topics generated using topic modeling techniques such as LDA are typically interpreted and named solely by researchers [12]. Thus, there is a lack of rigorous evaluation of topics with domain experts. It is, therefore, unclear whether topics generated through LDA or similar techniques make sense to developers or managers [13]. In this work, we use LDA to generate topics from the descriptions of invalid bug reports. We use expert-based validation to evaluate the practical relevance of the LDA-generated topics. Practitioners will validate two aspects: firstly, if there are common patterns in the descriptions of the invalid bug reports within an LDA-generated topic, and second, if the identified common patterns are relevant for devising preventive measures for reducing the future inflow of similar invalid bug reports.

In this study, the proposed approach is implemented and evaluated at a large-scale company using data from one of their mature products and involving engineers working on the product.

The outline of this paper is as follows. Section 2 presents related work on categorizing and predicting invalid bug reports and topic modeling in software engineering and bug reports. Section 3 presents our research approach. In Section 4, we present the study's results and analysis. Section 5 discusses study findings, and Section 6 discusses the validity threats to the study. In Section 7, we describe our conclusions and future work.

2. Related work

In this section, we describe studies on classifying invalid bug reports, predicting bug reports' validity, and applying topic modeling to bug reports.

2.1. Studies categorizing invalid bug reports

We found only a few studies [7–11] that analyze descriptions of invalid bug reports to identify their underlying causes. These studies are summarized in Table 1 and compared to our work (last row in Table 1). All studies used manual approaches without involving practitioners and three of them investigated only a single type of invalid bug reports (non-reproducible bug reports [7,10] and wontfix bug reports [9], respectively).

In this study, we apply an automatic approach to identify common patterns of invalid bug reports using the descriptions of the bug reports. Practitioners can then evaluate the relevance and usefulness of the identified patterns to devise preventive measures to decrease the inflow of such invalid bug reports in the future.

2.2. Studies predicting bug report validity

Few studies have focused on predicting the validity of bug reports [3,6,8,14,15]. Zanetti et al. [14] used a collaborative network on bug reports to predict the validity of bug reports using a support vector machine (SVM) classifier.

Fan et al. [3] extracted 33 features using five dimensions, including submitter experience, collaboration network, and a bug report's completeness and readability. Then, they used SVM and Random Forest classifiers to predict the validity of bug reports. Laiq et al. [6] used the bug reports' text and submitter experience to predict the validity of bug reports in a closed-source context.

Zanetti et al. [14], Fan et al. [3], and Laiq et al. [6] used an ML-based approach for predicting the validity of bug reports. However, He et al. [15] applied a deep learning-based approach using the summaries and headings of bug reports to predict their validity.

This study takes a different approach from previous studies [3,6,8,14,15]. While previous studies have focused on using ML-based classifiers to predict the validity of newly submitted bug reports, the aim of this study is to prevent the creation of invalid bug reports in the future.

To achieve this aim, the study employs LDA topic modeling to identify common patterns in the description of invalid bug reports. The results of this analysis can then be used by domain experts to suggest preventive measures that can reduce the inflow of invalid bug reports. This approach differs from previous studies in that it focuses on identifying and addressing the common causes of invalid bug reports, rather than just predicting their likelihood of being valid or invalid.

2.3. Topic modeling in software engineering

Topic modeling is an unsupervised machine learning technique that aims to produce semantically similar topics in the form of clusters for a given set of documents [16]. Contrary to supervised machine learning, it does not require labeled data (i.e., classified documents or taxonomies). It uses the frequencies and co-occurrences of words within one or more documents to generate semantically similar clusters.

Topic modeling has proven to be a valuable technique in software engineering due to its usefulness in identifying semantically similar representative clusters from large datasets. As a result, several studies have applied topic modeling for a variety of tasks in software engineering. [16,17]:

- Requirements: To support tasks related to requirements, for instance, requirements evolution or suggesting new features (e.g., [18]).
- Architecting: To support tasks related to architecture decisions, for instance, the selection of mash-up or cloud services (e.g., [19]).
- Documentation: To support tasks related to software documentation, for instance, localization of features in the documentation and automatic documentation generation (e.g., [20]).
- Coding: To support tasks related to coding, for instance, code clone detection, code refactoring, and developer behavior prediction (e.g., [21]).
- Testing: To support tasks related to software testing, for instance, test cases prioritization (e.g., [22]).
- Maintenance: To support tasks related to software maintenance, for instance, bug management (e.g., bug localization [23–26] and duplicate bug report detection [27–31]).

Among the topic modeling techniques (e.g., Latent Semantic Indexing (LSI), Probabilistic Latent Semantic Indexing (PLSI), and Latent Dirichlet Allocation (LDA)), LDA is the most popular and has been widely used in the software engineering literature [16,17]. In a systematic literature on the use of topic modeling in software engineering, Silva et al. [17] found that the majority of the papers either used LDA (80 out of 111), or an LDA-based technique (30 out of 111) for topic modeling, and 10 papers applied more than one technique.

On the one hand, LDA has shown promising results in software engineering tasks (see Section 3.4.1). On the other hand, LDA has some limitations. Lin et al. [32] report that the performance of LDA on informal documents and short texts, such as tweets, maybe be sub-optimal.

Zhao et al. [33] proposed Twitter-LDA to effectively discover meaningful topics from short text documents, such as tweets. They also reported that topics generated by standard LDA from the short texts (i.e., tweets) were less meaningful than Twitter-LDA topics.

Table 1
An overview of studies analyzing invalid bug report descriptions.

Author (year)	Context	Data	Investigated type of bug reports	Approach	Purpose
Sun [8] (2011)	Closed-source	613 bug reports of more than 4 years.	Invalid	Manual	To categorize the reasons for invalid bug reports.
Erfani Joorabchi et al. [7] (2014)	Closed and Open-source	Classified 1643 bug reports of five open-source and one proprietary repository.	Non-reproducible	Manual	To categorize non-reproducible bug reports.
Su et al. [11] (2017)	Closed-source	231 invalid bug reports of three server applications.	Invalid	Manual	To categorize invalid bug reports.
Rahman et al. [10] (2020)	Open-source	576 bug reports of Firefox and Eclipse.	Non-reproducible	Manual	To identify the factors for non-reproducible bug reports and investigate how professional developers cope with non-reproducible bugs.
Panichella et al. [9] (2021)	Open-source	667 GitHub wontfix bug reports.	Wontfix	Manual	To find common reasons for wontfix bug reports.
Our work	Closed-source	More than 600 bug reports.	Invalid	Manual and Automatic	To investigate if the descriptions of invalid bug reports can be used for identifying any shared causes and whether this helps in suggesting preventive measures.

Approaches such as pooling (i.e., aggregating semantically or temporally similar documents into a single document [34]) and contextualization (i.e., making subsets of documents based on their context, e.g., time or hashtags for tweets [35]) could be utilized to improve the performance of LDA on short texts. Furthermore, hyperparameters can be tuned to optimize LDA performance [17].

Topic modeling in bug reports

Several studies have applied topic modeling on bug reports to support practitioners in various bug management-related tasks, such as bug localization [23–26], duplicate bug report detection [27–31], bug severity prediction [36–39], automatic bug triage [38,40,41], and bug report categorization [42–44].

This study uses LDA, a topic modeling approach to cluster invalid bug reports based on their descriptions. This approach produces two matrices: (a) a documents to topics matrix and (b) a topics to keywords matrix. In previous research in software engineering [17], the topics produced by topic modeling are commonly named based on keywords only. We use both the keywords and the actual descriptions of some of the bug reports assigned to a topic to name the topics. In this study, experts from the case company performed the naming.

3. Research methodology

We aim to investigate the usefulness of topic modeling to support the identification of common patterns in bug reports. Furthermore, we want to investigate whether the identified patterns help to devise preventive measures for reducing the inflow of invalid bug reports.

To achieve our aim, we conducted an industrial case study [45] with a two-step research process (see Fig. 1). First, we manually analyzed a subset of invalid bug reports to assess the feasibility of the idea of identifying relevant common patterns of invalid bug reports using descriptions of invalid bug reports. This step was guided by the following research question:

RQ1: To what extent can common patterns in invalid bug reports be identified and used to suggest improvements?

The likely explanation of invalid bug reports can be expected in their descriptions. Thus, in this research question, we investigate the descriptions of invalid bug reports to identify their causes and possibly categorize bug reports in the same class (i.e., a common pattern for bug reports belonging to the same cause category).

Furthermore, we are interested in investigating the relevance and usefulness of identified common patterns, i.e., do they make sense to domain experts, and to what extent they are helpful in suggesting preventive measures to decrease the inflow of invalid bug reports.

In the second step, we used LDA to automatically cluster invalid bug reports based on their descriptions. The clustering reduces the number of invalid bug reports that need to be manually analyzed. Furthermore, each cluster has a set of weighted keywords that are representative of the cluster, which can further assist in labeling the common pattern found in a cluster. Thus, to investigate the role of automation in the identification of common patterns, we posed the following research question:

RQ2: To what extent can automation support the identification of common patterns in invalid bug reports?

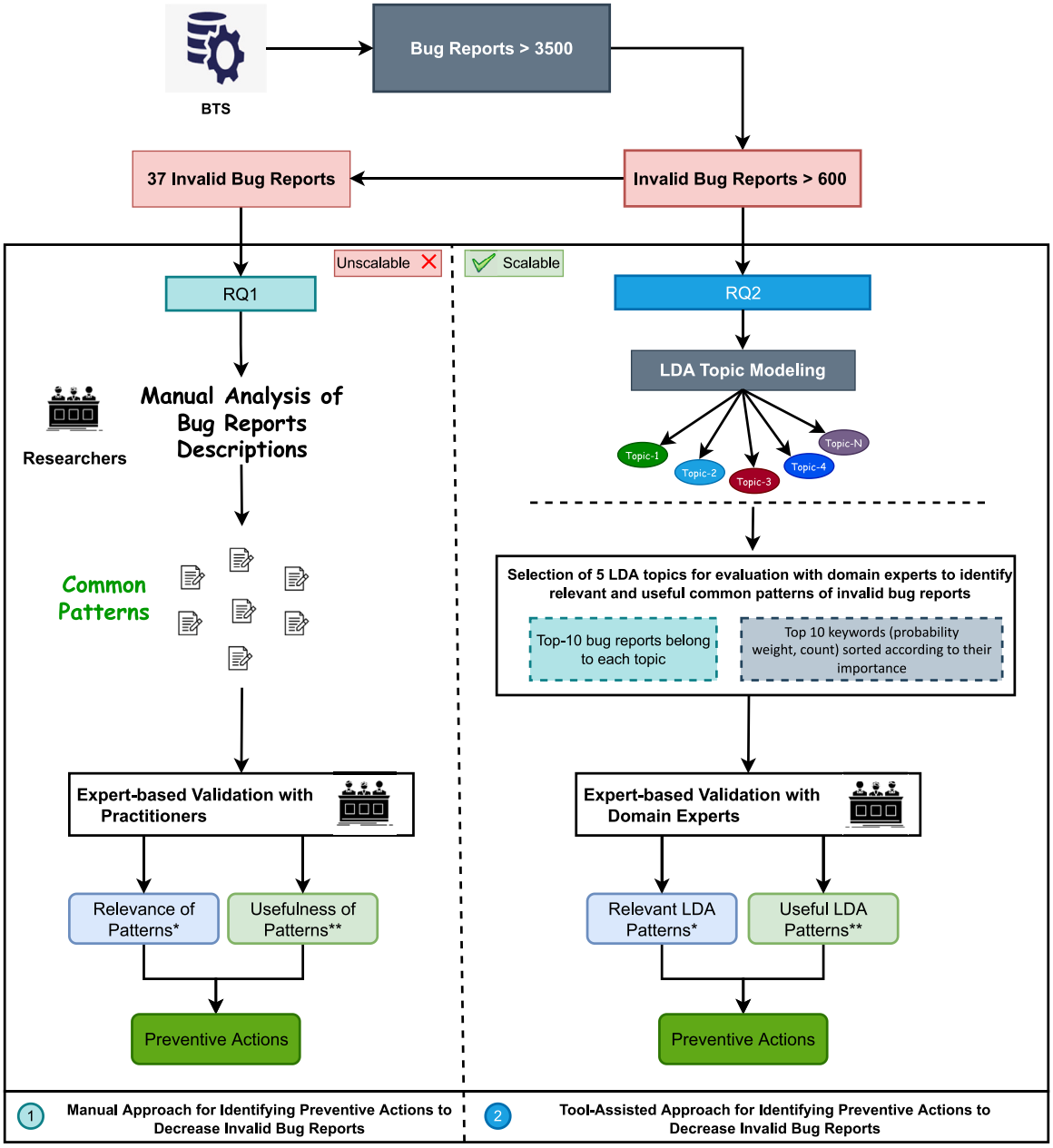
3.1. Case description

The case company is a large multinational globally distributed Telecommunication vendor. Their context can be categorized as large-scale software-intensive system development.

Several programming languages are used in product development at the company. However, a majority of the code at the case company is written in Java, JavaScript, and C++. The studied product is mature and has a large code base older than ten years. The product team uses a central Bug Tracking System (BTS) for managing bug reports.

The company follows agile practices and principles in software development, e.g., sprint planning meetings and self-organizing teams.

As shown in Fig. 2, at the case company, bug reports can be submitted by customers, testers, and developers themselves. Submitted bug reports are first screened by the Change Control Board (CCB). When the CCB assesses a bug report as valid, it is assigned to a relevant team. The team then inspects the assigned bug report in more detail. If the team still assesses the bug report as valid, it assigns it to a developer for resolution.



***Relevant:** Do they make sense to domain experts?
****Useful:** To what extent they are helpful in suggesting preventive measures?

Fig. 1. Overview of our research approach.

3.1.1. Invalid bug reports

In this study, we have used the working definition of invalid bug reports at the case company. At the case company, invalid bug reports are defined as follows:

- **No such requirement exists:** The submitted bug report does not describe any requirement that a system must fulfill.
- **Configuration issues:** The submitted bug report is not a fault in a system but occurred due to faulty configuration settings used by the bug submitter.
- **Misunderstanding:** The submitted bug report describes a misunderstanding of the working procedure of the system.
- **Future requirement:** The submitted bug report does not describe a fault in the system but could be considered as a future requirement.
- **Insufficient information:** The submitted bug report does not provide sufficient information to reproduce or analyze the submitted bug report.

3.2. Data collection

In this study, we use the company's bug management tool BTS as our main data source. We collected more than 3500 bug reports for the studied product (see Section 3.1) spanning 5 years (2016–2021), of which around 17% were invalid bug reports. For our analysis, we used the following fields of the invalid bug reports: heading (a short description of the bug), observation (a detailed description of the bug, including replication steps), answer (text describing the resolution of the bug report or the reason for considering it invalid), and resolution

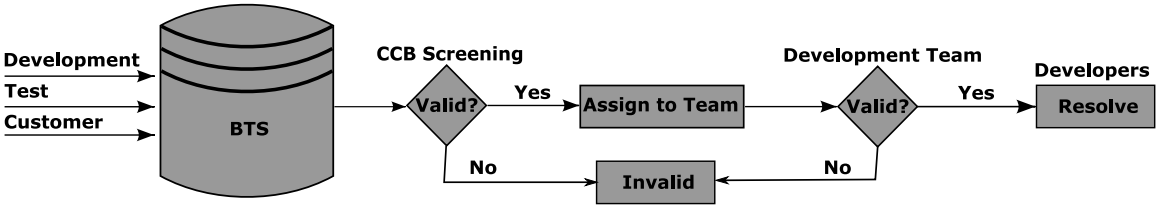


Fig. 2. Bug management flow at the case company.

Table 2
Practitioner's profiles, Focus Group Meeting-1 (FG1), Focus Group Meeting-2 (FG2), expert-based validation of automatic approach (EV).

Position	Description	Experience	FG1	FG2	EV
Domain Expert	The practitioner is a domain expert and software architect who works with a cross-functional team and is mainly responsible for the product release.	~8 years	✓	✓	✓
Manager	As a product manager, the practitioner is mainly responsible for communication and product quality assurance. However, the practitioner also plays a role in product development decisions.	~20 years	✓	✓	
Manager	The practitioner is an architect in analytics and Business Intelligence (BI). He is responsible for services delivery tools, i.e., supporting the products teams with BI in that area.	~20 years	✓	✓	
Data Scientist	The practitioner is a team leader of the data science team at the case company supporting multiple product units, i.e., developing tools to boost research and development. Further, the practitioner is qualified with PhD in applied computing.	~10 years		✓	
Junior Data Scientist	The practitioner is qualified with a master's degree in software engineering and currently works at the case company as a junior data scientist.	~3.5 years		✓	
Domain Expert and Quality Architect	The practitioner is a quality manager for a couple of products at the case company and an experienced software engineer.	~10 years	✓	✓	✓

status (see Section 3.1). We further discuss the justification for these fields in Section 3.4.

In general, practitioners' feedback is essential not just for validating a proposed technique but also facilitates technology adoption in practice [46]. Moreover, practitioner involvement is especially important for technologies that require humans in the loop. One of the shortcomings of existing research on invalid bug reports is its lack of involvement of software maintenance practitioners (see Sections 1 and 2.1). Thus, this case study is conducted with the active involvement of industry participants. We conducted two focus group meetings and two expert-based validations with practitioners (see Table 2) to collect their feedback and validate our approach. Focus group meetings involve several roles at the case company, including managers, domain experts, and data scientists.

In the first focus group meeting, we evaluated the results of the manual analysis. In the second focus group meeting, we presented our automatic approach and collected feedback from practitioners on the approach.

In the two expert-based validations with two domain experts, we evaluated the results of the automatic approach, i.e., the relevance of automatically identified common groups of invalid bug reports and their usefulness in suggesting preventive measures.

Moreover, to understand the case company's management process, we read their internal documentation and asked practitioners questions regarding the processes and terminology used at the case company.

3.3. Design and analysis approach for RQ1 (i.e., To what extent can common patterns in invalid bug reports be identified and used to suggest improvements?)

To answer RQ1, the second author manually coded invalid bug reports submitted for the selected product in the last 6 months (i.e., 37 invalid bug reports). For the manual coding, we used the full final description of bug reports, including the final verdicts (i.e., invalid or valid), comments, and communication between the submitters and development teams. We carefully reviewed their descriptions and assigned codes for each indication of causes for why a bug report was considered an invalid bug at the case company. We followed an iterative process of adding new codes and revising the previous codes as we coded each additional bug report. The themes related to invalid bug report causes emerged by analyzing repeating codes.

To assess the validity of the common patterns for causes of invalid bug reports identified by the researchers, we conducted a focus group with practitioners from the case company. During the focus group, we briefly presented our approach. Afterward, we presented our findings regarding common issues identified in the data and collected practitioners' feedback.

3.4. Design and analysis approach for RQ2 (i.e., To what extent does automation support the identification of common patterns in invalid bug reports?)

As described above, we first manually analyzed and identified relevant and useful common patterns of invalid bug reports using their descriptions. This established the feasibility of the idea of using the descriptions of invalid bug reports to inform the suggestion of preventive actions at the case company. However, such a manual approach is impractical as the analysis needs to be done repeatedly and requires substantial effort to manually code a large number of code bug reports. Thus, we used LDA, a topic modeling approach, to support the identification of common patterns of invalid bug reports from more than 600 bug reports.

We applied LDA on the answer and observation fields of bug reports, as these fields together provide a sufficiently detailed description of both the problem and the resolution. Furthermore, the practitioners recommended using these two fields, which we also found appropriate during our manual analysis for RQ1. The resulting topics from LDA are each a collection of words that frequently co-occur in the corpus of documents. However, it is important to note that not all of the generated topics may address a particular subject. For example, it could be a cluster filled with common or trivial terms [13] that are irrelevant to the properties of interest of a bug report. Thus, we are interested in investigating whether the topics generated by LDA can assist practitioners in identifying common patterns of invalid bug reports, as we found

in the manual analysis (RQ1). Additionally, we seek to explore how the identified patterns can be utilized by practitioners to devise preventive measures to reduce the influx of invalid bug reports.

In the following subsections, we describe the motivation for using the LDA topic modeling approach, the selection of the LDA model, and the evaluation of LDA results.

3.4.1. LDA topic modeling

In this case study, we used LDA [47], a generative statistical technique, for topic modeling. We choose LDA because LDA has shown promising results in various software engineering tasks, such as software maintenance [48,49], software testing [22,50,51], software requirements [18,52], and source code refactoring [53,54]. In a systematic literature review on topic modeling in software engineering, Silva et al. [17] reported that 95 out of 111 papers used LDA or LDA-based techniques for topic modeling.

LDA makes an assumption that the entire corpus of documents can be described with a set of topics and that each document belongs to one or more topics. Likewise, each term in the corpus belongs to one or more topics [16]. Using these assumptions, LDA is able to discover topics that suitably describe the entire corpus [47]. Further, LDA is robust to over-fitting compared to other topic models, such as Probabilistic Latent Semantic Indexing (pLSI) [47].

3.4.2. LDA model selection

We generated different models consisting of 20, 15, 10, and 8 topics, using the observation and answer fields of invalid bug reports. Based on the coherence values and our subjective assessment, we selected a topic model with 10 topics. The coherence value started decreasing after 10 topics. Furthermore, we observed noise (i.e., repetition of words and words having negligible weights and very low frequencies) in topic models for more than 10 topics.

3.4.3. Qualitative evaluation

In software engineering, topics generated using topic modeling techniques are commonly evaluated/named using keywords only [17]. Often these topics are named/interpreted solely by the researchers [12]. Thus, there is a lack of rigorous evaluation of topics with domain experts. It is, therefore, unclear whether the topics generated through LDA or similar techniques make sense to developers or managers [13].

In this work, we first apply LDA and generate topics using descriptions of invalid bug reports. Then we backtrack to get the original bug reports (i.e., documents in LDA terminology) belonging to each topic. Finally, we use expert-based validation to evaluate those topics, i.e., to assess if there are relevant common patterns of invalid bug reports in those topics and to what extent we can use those common patterns to devise preventive measures.

Topic naming/interpretation takes time and can be tedious [13]. To conclude the evaluation sessions within an hour with domain experts, we validated only 5 of the 10 topics, namely the three largest topics, one medium sized, and one smaller topic. Note that there is a significant difference in size also between the three largest topics. To investigate whether LDA-generated topics can help practitioners identify relevant patterns for suggesting preventive actions, we considered five topics sufficient for validating the practical utility of our approach.

For the validation process (i.e., topic naming to identify relevant common patterns from the LDA-generated topics and suggesting preventive measures based on them), the domain experts were provided with the following information:

- Instructions about the task and its purpose.
- Top 10 bug reports belonging to each topic. The top 10 bug reports were selected based on their contribution to a topic, measured as percentage.
- A bar chart (see Fig. 3) for each topic depicting the top ten keywords (i.e., LDA-generated keywords) for the topic together with the frequency and weight for each of the keywords.

The feedback for each topic was recorded.

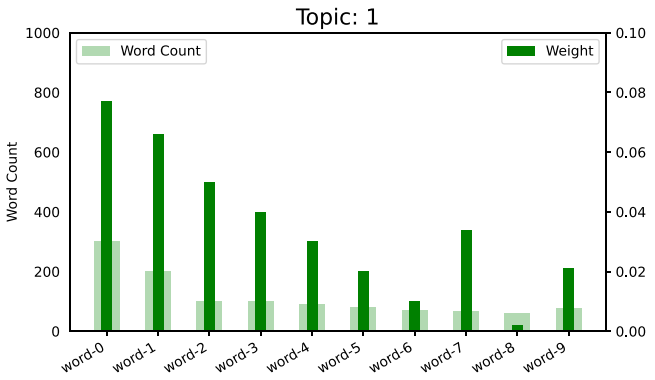


Fig. 3. Top 10 keywords with frequency and weight. The actual keywords have been anonymized.

3.4.4. Data preparation

In order to apply LDA, we followed standard pre-processing steps for topic modeling [17]. As input to LDA, we used the observation and answer fields of the bug reports (as described in Section 3.4). In a pre-processing step, we first removed the headers of the fields (belonging to the template), extra white spaces from the text, and stop-words. We then applied lemmatization and tokenization. In the final step, we converted the token into a document term matrix (i.e., the text as a matrix where rows are documents and columns are terms) and created a dictionary as an input for the model with other parameters (i.e., corpus, number of topics, etc.).

4. Results and analysis

In this section, we present the results and analysis of our study.

4.1. RQ1: To what extent can common patterns in invalid bug reports be identified and used to suggest improvements?

As a result of the manual analysis (see Section 3.3 for analysis procedure), we identified the following relevant common patterns of invalid bug reports and the corresponding number of bug reports for each pattern: *misunderstanding of functionality (5), working as expected (15), non-reproducible (6), wrong version (4), faulty configuration (5), and faulty test data (2)*.

Below we briefly describe common patterns[1] identified using the manual approach:

- **Misunderstanding of functionality:** The bug reports in this category are attributed to misunderstandings by the submitter of a bug report regarding how certain tasks ought to be performed by the system. For example, a common issue in this category relates to dependencies between functions, e.g., X and Y. To activate Y, X must be started and enabled. Otherwise, a user is unable to activate Y. This category of invalid bug reports has also been identified in previous work by Sun [8], Erfani Joorabchi et al. [7], and Rahman et al. [10].
- **Working as expected:** The bug reports in this category are quite similar to category *misunderstanding of functionality*. However, in this category, bug reports point to system functionalities that work as defined according to the documentation. While the bug reports in the *misunderstanding of functionality* pattern could be considered a general misunderstanding of systems functionalities,

[1] Due to company restrictions, we cannot report names of components/features and exact terms. However, we report the main common patterns identified as a result of manual analysis.

the invalid bug reports in this category are primarily related to user expectations for functionality to behave according to their suggested definition.

- **Non-reproducible:** The bug reports in this category describe bugs that developers cannot reproduce. Such bugs could be reported due to network issues or some unknown problem on the side of the submitter of the bug report. This category of invalid bug reports has also been identified in previous work by Panichella et al. [9], Erfani Joorabchi et al. [7], and Rahman et al. [10].
- **Wrong version:** The bug reports in this category describe issues that are caused by using a wrong version of a component or system by the submitter of the bug report. Either support for that version is no longer available, or the version needs to be updated. This category of invalid bug reports has also been identified in previous work by Panichella et al. [9], Erfani Joorabchi et al. [7], and Su et al. [11].
- **Faulty configuration:** The bug reports in this category describe issues that are caused by faulty configurations. This is a well-known reason for invalid bug reports. However, in reality, faulty configurations are not bugs. The problem is caused by misconfiguration, and the system will work after proper configuration. This category of invalid bug reports has also been identified in previous work by Sun [8], Panichella et al. [9], Erfani Joorabchi et al. [7], and Su et al. [11]. They found that bug reports were submitted in this category because the submitter used the wrong configuration. Once the correct configuration is used, bugs are no longer present.
- **Faulty test data:** The bug reports in this category can be traced to the use of faulty data while testing the system or using it to achieve certain tasks.

The practitioners' general responses were that these common patterns are plausible and align with their subjective judgment. One of the domain experts commented that invalid bug reports related to two of the above-listed categories ("*working as expected*" and "*misunderstanding of functionality*") are quite prevalent at the case company.

Another domain expert acknowledged the relevance of the identified common patterns for proposing corrective actions. One of the practitioners commented, *"After looking at those common patterns, I would suggest improving documentation for our complex features"*.. Another participant commented, *"We should consider educating people to decrease invalid bug reports belonging to common patterns: working as expected and misunderstanding of functionality"*.. Preventive measures proposed by domain experts to decrease the inflow of invalid bug reports are described in Section 4.2.2.

Managers also acknowledged the relevance of common patterns. They showed interest in scaling up the approach to investigate if the approach can be used on larger data-sets and for other products.

4.2. RQ2: To what extent does automation support the identification of common patterns in invalid bug reports?

To support the identification of common patterns in invalid bug reports through automation, we applied LDA as described in Section 3.4. Then, we validated the automated approach with two domain experts (see Table 2 for their profiles).

The domain experts were given five topics, including the top 10 bug reports belonging to each topic and the most frequent words with their respective weights as described in Section 3.4.3.

In Table 3, we describe the feedback of the two domain experts on the five topics for identifying relevant common patterns of invalid bug reports. Section 4.2.2 describes the feedback on the usefulness of common patterns to devise preventive measures. In Section 4.2.3, we present the feedback on the usefulness of term frequency and weight in interpreting the topics.

In Table 3, we can see that a topic can be viewed from varied angles by domain experts. For example, feedback on the first topic indicates that both domain experts notice a pattern related to a specific feature of a system. Whereas, the first domain expert attributes the invalid bug reports in this cluster to inexperienced people, the second domain expert attributes them to faulty test data. This does not indicate a problem with the topics or the evaluation but differences in the knowledge and interests of the domain experts.

4.2.1. Common patterns identified using LDA

During the expert-based validation of the LDA topic modeling approach (see Section 3.4.3), practitioners identified the following seven patterns: *faulty test data, working as expected, complex features, specific features, lack of system knowledge, reported due to inexperience, and new requirement*. Of those seven, two were already identified during manual analysis (faulty test data and working as expected) and are described in Section 4.1. Below we describe the remaining five patterns[2]:

- **Complex features:** This category captures invalid bug reports that are related to complex features of a system. One interesting finding by domain expert-1 (see Table 3) was identifying a common pattern of invalid bug reports related to a complex feature. He further added that there are certain complex features in their system, and bug reports in this topic are related to that specific feature.
- **Specific features:** This category captures invalid bug reports that are related to specific features of a system. During the validation of the automatic approach, both domain experts identified patterns (i.e., from topics 1, 2, and 5, see Table 3) of invalid bug reports related to specific features of the system. Due to company restrictions, we cannot share the names of those features.
- **Lack of system knowledge:** This category captures invalid bug reports that are reported due to a lack of a system of knowledge. In the feedback, practitioners highlighted that people often submit such invalid bug reports due to their lack of knowledge about complex and latest functionalities of the system. Sun [8] also found that some invalid bug reports were submitted because of the submitters' lack of system knowledge. This category is closely related to the pattern *misunderstanding of functionality*, i.e. bug reports submitted due to a misunderstanding of system functionality. However, the bug reports in this category are due to insufficient knowledge about the system.
- **Reported due to inexperience:** This category captures invalid bug reports that inexperienced individuals mainly submit. The practitioners highlighted that inexperienced individuals lack general knowledge of functionalities of the system and its complex features. Therefore, they report such kind of invalid bug reports. This category is related to the *lack of system knowledge* category with an additional aspect: inexperienced people only submit invalid bug reports in this category.
- **New requirement:** This category captures invalid bug reports that are submitted as bugs; however, these bug reports point to new requirements or features for the system to support. This category of invalid bug reports has also been identified in previous work by Panichella et al. [9], Erfani Joorabchi et al. [7], and Rahman et al. [10]. They discovered that new requirements or feature requests were submitted as bug reports.

[2] Due to company restrictions, we cannot report names of components/features, exact terms, and exact topics. However, we report the main common patterns identified as a result of LDA topic modeling.

Table 3
Feedback by domain experts on the automatic approach.

Topic Id	Domain expert-1	Domain expert-2
#1	These bug reports make sense together, and I can see a couple of patterns of invalid bug reports in this topic: (a) specific feature (also complex feature) and (b) reported due to inexperience.	I see a couple of patterns, one related to test data and the other related to a specific feature.
#2	These bug reports do not describe issues in the system and can be categorized as working as expected, i.e., working as expected pattern.	Bug reports in this topic point to issues of a specific feature and its test-related data (i.e., related to test data).
#3	In this topic, submitters are reporting bug reports that are not implemented and could be future requests or new requirements.	These bug reports are reported due to lack of system knowledge, i.e., lack of system knowledge pattern.
#4	Multiple themes not dominated by any specific.	No distinct pattern.
#5	In this topic, I can see two unique patterns of invalid bug reports, i.e., both patterns are related to specific features of the system.	No distinct pattern.

4.2.2. Improvement suggestions

In order to validate the usefulness of identified common patterns for suggesting preventive measures to decrease the number of invalid bug reports in the future, we used expert-based validation as described in Section 3.4. The domain experts suggested the following preventive measures:

(a) **Co-reviewer:** One of the highlighted approaches proposed by practitioners was the use of a co-reviewer for bug report writing at the early stages. It is important to ensure only bug reports describing faults in the code proceed to the next step of the bug handling process. Otherwise, a lot of resources will be consumed. However, this idea will only work for testers and developers but not for customers.

(b) **Improve documentation:** The practitioners also agreed on the idea of improving the documentation for some parts of software systems. In our manual analysis, we identified that a number of invalid bug reports could be avoided by improving the system's documentation in some areas.
It should be noted that our approach helps in identifying those parts of the system where such improvements will likely have the most positive effects.

(c) **Training of new employees:** It is highly likely that new employees will make mistakes while writing the bug reports and eventually will submit invalid bug reports. Thus there is a need to train those people. This could, for example, be achieved by working in pairs, e.g., a junior and senior in combination. Further, training could be optimized for specific features identified in the automatic approach results.
Bettenburg et al. [55] also reported that inexperienced people are likely to submit invalid bug reports. Just et al. [56] pointed out that people with experience often submit high-quality bug reports. Thus, training is important for new employees, or they could work in pairs (i.e., junior and senior).

(d) **Decision support tool for invalid bug prediction:** Our discussions with practitioners indicated that an ML-based intelligent tool could be used to identify likely invalid bug reports early. Using such a tool could save resources and help to effectively manage bug reports (i.e., prioritize valid bug reports based on the prediction results of the ML-based tool).

4.2.3. Term frequency and weight for interpreting LDA topics

We collected practitioners' feedback on the usefulness and effectiveness of term frequency and weight associated with each keyword for naming topics and identifying the common patterns of invalid bug reports. We presented them with the top 10 words belonging to each topic (see Fig. 3 for an example) and collected their feedback. In Table 4, we present their response for the selected topics.

In general, practitioners' responses were positive regarding the usefulness of keywords for interpreting the topics. They also reported that some keywords helped identifying patterns, for instance, keywords related to the specific features of their system. One of the domain experts commented, "*I can see the keywords in this topic point to one of our features in the system. Thus, [I conclude] this cluster of invalid bug reports is related to that specific feature*".

5. Discussion

In this case study, we analyzed invalid bug reports in a large-scale closed-source software-intensive product development context. We collected more than 3500 bug reports data spanning 5 years, of which around 17% were invalid. This study was conducted with the active participation of industry practitioners.

In the following subsections, we discuss the findings of the case study.

Data-driven Approach for Decreasing Invalid Bug Reports: In this study, we propose and evaluate LDA, a topic modeling approach, to identify patterns in descriptions of invalid bug reports. We analyzed more than 600 invalid bug reports and identified a representative group of clusters that were further analyzed by domain experts (see Section 3.4). The feedback from domain experts indicates (a) that relevant and useful common patterns related to the causes of invalid bug reports can be identified using their descriptions (see Sections 4.1 and 4.2 for the identified common patterns) and (b) that these patterns can be utilized to suggest to preventive measures (see Section 4.1 and Table 3 for preventive measures).

The common patterns describing the causes of invalid bug reports will be highly context-specific. Thus, the preventive measures proposed based on the identified common patterns for one product may not be applicable to other products, even at the same company. Also, the preventive measures need to be changed, as the underlying reasons for invalid bug reports will change over time. This is where the manual approaches fail; however, our approach delivers this. Our approach can automatically identify a representative group of clusters that domain experts can further analyze to identify relevant common patterns and use them to devise preventive measures.

On the one hand, our approach is useful in identifying context-specific patterns such as *complex features and features where the documentation should be improved*, see Table 3. On the other hand, our approach also identified general patterns that are in line with the common reasons and categories of invalid bug reports identified in previous work [7–11], such as *misunderstanding of functionality, testing error, non-reproducible, environmental issue, and conflicting expectation.*

In those previous works, a validation with domain experts is missing; the researchers solely identified the common patterns describing the causes of invalid bug reports. Furthermore, the potential use of the

Table 4
Practitioners' feedback on term frequency and importance.

Topic Id	Feedback - domain expert-1	Feedback - domain expert-2
#1	Somewhat helpful.	Keywords were helpful.
#2	Yes, keywords were helpful in identifying the pattern.	Keywords were helpful.
#3	Not useful.	Keywords were somewhat helpful.
#4	Not useful.	Not much helpful.
#5	Useful.	Not helpful.

patterns for suggesting preventive measures has been overlooked. However, involving practitioners is essential since only they have sufficient knowledge and context information to assess whether the patterns are (a) relevant and (b) useful to suggest suitable preventive measures for a certain "class" of bug reports. Likewise, the LDA topics interpretation accuracy of non-experts is relatively low (50%, as reported by Hindle et al. [13]). Consequently, we involved software maintenance practitioners (see Table 2) in identifying relevant common patterns of invalid bug reports from LDA topics generated using the descriptions of invalid bug reports. Further, we also evaluated their usage for identifying preventive measures.

In this study, we do not evaluate the effectiveness of the proposed preventive measures for decreasing invalid bug reports. This will require a longitudinal study. Furthermore, it would be interesting to explore to what extent the identified common patterns and preventive measures for one product are relevant to other products in the same context.

Using our approach, practitioners can devise preventive measures targeting current issues of invalid bug reports. Furthermore, the approach can help in identifying specific areas for improvement that may help to decrease the inflow of invalid bug reports, such as complex features, features where documentation can be improved, and common patterns of bug reports submitted by inexperienced developers.

Topic Modeling in Invalid Bug Reports: Topic modeling techniques, such as LDA, are used to discover semantically similar clusters (i.e., topics) from a large collection of documents. However, not all topics may describe a subject of interest, and not all may make sense to domain experts. A cluster might just comprise common terms. Our findings reveal that LDA-generated topics from invalid bug reports descriptions make sense to domain experts (see Table 3). Thus, LDA can be used to effectively identify common patterns of invalid bug reports and that these patterns are relevant and useful for practitioners to devise preventive measures.

Due to practitioners' interest, we developed a tool to support our approach. The tool could help new product managers or product teams to explore and identify common patterns of invalid/invalid bug reports. The tool provides the following features (see column two of Fig. 1):

- The tool groups semantically similar bug reports.
- The tool shows the top ten bug reports and top 10 keywords belonging to each group.
- The tool provides a graphical representation of topics with their top keywords and weight, see Fig. 3.[3]
- The tool provides support for adding domain-specific stop-words.
- The tool has LDA inference support to evaluate the effects of the proposed preventive measures.

Recommendation for Evaluating LDA Topics: Based on our experience and feedback from the domain experts, we provide the following general recommendations for the evaluation of LDA topics in the context of bug reports:

- *Topic naming:* Topic naming/interpretation takes time and could be tedious. In our study, it took more than an hour for each domain expert to name topics. However, rigorous evaluation of topics with domain experts is necessary to have confidence in the findings.
- *Term frequency and weight for topic naming:* Our findings show that sole usage of terms with high frequency and weight may not be sufficient for naming/interpreting all topics, even for domain experts (see Table 4). Thus, we recommend using both the top topic words and the top documents belonging to each topic.

[3] Due to company restrictions, we cannot report actual words with their frequency and weight.

6. Limitations and threats to validity

This section discusses the limitations of the proposed approach and potential threats to the validity of the empirical investigations conducted as part of the design.

6.1. Applicability and relevance of solution

The proposed approach has been applied and evaluated in a large-scale software development context. However, the solution is relevant for any cases where the influx of invalid bug reports is a challenge, provided bug reports are sufficiently described (see Section 3.4).

The proposed approach uses the descriptions of invalid bug reports because the likely explanation for the causes of invalid bug reports can be expected there. Thus, common patterns describing the causes of invalid bug reports can be identified using their descriptions and can further likely be utilized by practitioners to devise preventive measures.

6.2. Internal validity

We extracted more than 3500 bug reports from the company's bug tracking system and analyzed the 17% of them (approx. 600 bug reports) that were flagged as invalid. The used data of invalid bug reports is not extensive but sufficient and realistic for the proposed approach because the number of invalid bug reports in the closed source is smaller [6] compared to the open source. However, the quality of bug reports in the closed source is relatively higher [57] compared to the open source. Thus the descriptions of bug reports would contain sufficient information. Also, the case company follows a rigorous process to manage bug reports. Thus, the bug reports used in this study contain a sufficient level of detail. Furthermore, we only used bug reports with practitioners' final verdicts (i.e., valid or invalid) as these labels represent judgments of more than one domain expert. The points mentioned above give us confidence that the proposed approach is data-driven and there is little likelihood of overfitting.

The extent of common patterns in the invalid bug reports was investigated through manual analysis, which imposes a risk of researcher bias. To mitigate this threat, we validated the identified common patterns with practitioners. Similarly, for the automatic approach, we validated the LDA-generated patterns with practitioners to validate the usefulness of the approach.

Further assumptions of usefulness are based on the possibility to identify preventive measures. The preventive measures for decreasing the number of invalid bug reports in the future were proposed by practitioners.

6.3. External validity

Since this is a single case study, statistical generalization to other cases is not possible. We assume that the extent of common patterns are similar in similar cases, where the scale of organization, bug management approach, programming languages used, maturity of the product, size of code-base, number of people involved in development, testing and support of the studied product, and type of system are important factors to consider (see Section 3.1).

Similarly, we assume that the benefits of applying the proposed data-driven approach applies to other cases managing a large inflow of reported bugs. However, to gain evidence for both these assumptions, further studies in more contexts are needed.

Nevertheless, some of our identified common patterns are in line with the common reasons, and categories of invalid bug reports identified in previous work [7–11], such as *misunderstanding of functionality, testing error, non-reproducible, environmental issue, and conflicting expectation.*

6.4. Construct validity

Similar to previous works [3,14], this case study defines valid and invalid reports according to their working definition at the case company, see Section 3.1.

To avoid bias and to improve the reliability of the interpretation of practitioners' feedback, two researchers participated in the focus group meetings with the practitioners. One was taking notes, and the other led the discussion. Later, both researchers discussed the feedback and presented it back to the practitioners for validation.

The feedback from two domain experts for each topic was collected using a form. We internally reviewed and revised the form before sending it to domain experts. Furthermore, we presented our results back to domain experts for validation.

6.5. Conclusion validity

In this study, we used an LDA topic modeling approach to identify common patterns of invalid bug reports. We choose LDA because LDA has shown promising results in various software engineering tasks, such as software maintenance [48,49], software testing [22,50,51], software requirements [18,52], and source code refactoring [53,54].

LDA makes an assumption that the entire corpus of documents can be described with a set of topics, and each document belongs to one or more topics. Likewise, each term in the corpus belongs to one or more topics [16]. Using this assumption, LDA is able to discover themes that suitably describe the entire corpus [47]. Furthermore, LDA is robust to overfitting compared to other topic models, such as Probabilistic Latent Semantic Indexing (pLSI) [47]. Notwithstanding, LDA has some limitations. For instance, LDA performance can be suboptimal for short and informal documents such as tweets [32]. In this study, we used invalid bug report descriptions (i.e., documents in LDA terminology) from a closed-source company. The bug reports contain sufficient information, as the quality of bug reports in the closed source is relatively higher than that in the open source [57], and the case company follows a rigorous process in bug management.

Moreover, LDA-generated topics may point to common words instead of describing a subject of interest and thus may not make sense to domain experts. However, our findings reveal that LDA-generated topics from the description of invalid bug reports make sense to domain experts (see Table 3) and can be used to identify common patterns of invalid bug reports and further be utilized by domain experts to suggest preventive measures for decreasing the inflow of invalid bug reports (see Section 4.2.2).

7. Conclusion and future work

In large-scale software development, a number of bug reports are submitted daily. However, not all of the submitted bug reports describe an erroneous behavior of a software system. Such invalid bug reports are treated the same way as valid bug reports. Thus, they consume a significant amount of time and resources and affect the bug management process and a company's ability to respond quickly to real problems adversely.

This study shows that relevant common patterns of invalid bug reports can be identified from their descriptions and can be used by practitioners to devise preventive measures. We also showed that it is possible to support the identification of relevant common patterns with automation in large-scale software development. Using LDA, practitioners can effectively identify relevant common patterns of invalid bug reports and analyze them further to devise preventive measures for decreasing the inflow of invalid bug reports.

Furthermore, our proposed approach to identifying common patterns of invalid bug reports from their descriptions and their usage to devise preventive measures is context-independent. Thus, it can be utilized in other contexts where large amounts of invalid bug reports are an issue.

In the future, we aim to evaluate our approach on other products at the case company and evaluate the effectiveness of the proposed preventive actions.

CRediT authorship contribution statement

Muhammad Laiq: Conceptualization, Data curation, Methodology, Software, Formal analysis, Investigation, Project administration, Validation, Visualization, Writing – original draft, Writing – review & editing. **Nauman bin Ali:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing. **Jürgen Börstler:** Conceptualization, Investigation, Methodology, Project administration, Supervision, Writing – review & editing. **Emelie Engström:** Conceptualization, Investigation, Methodology, Project administration, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgments

This work has been supported by ELLIIT, a strategic research environment in information technology and mobile communications funded by the Swedish government. The work has also been supported by a research grant for the GIST project (reference number 20220235) from the Knowledge Foundation in Sweden.

References

[1] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, B. Xu, How practitioners perceive automated bug report management techniques, IEEE Trans. Softw. Eng. 46 (8) (2018) 836–862.

[2] C. Parnin, A. Orso, Are automated debugging techniques actually helping programmers? in: Proceedings of the 2011 International Symposium on Software Testing and Analysis, 2011, pp. 199–209.

[3] Y. Fan, X. Xia, D. Lo, A.E. Hassan, Chaff from the wheat: Characterizing and determining valid bug reports, IEEE Trans. Softw. Eng. 46 (5) (2018) 495–525.

[4] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, H. Mei, A survey on bug-report analysis, Sci. China Inf. Sci. 58 (2) (2015) 1–24.

[5] O. Chaparro, J.M. Florez, U. Singh, A. Marcus, Reformulating queries for duplicate bug report detection, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, 2019, pp. 218–229.

[6] M. Laiq, N.b. Ali, J. Böstler, E. Engström, Early identification of invalid bug reports in industrial settings–a case study, in: International Conference on Product-Focused Software Process Improvement, Springer, 2022, pp. 497–507.

[7] M. Erfani Joorabchi, M. Mirzaaghaei, A. Mesbah, Works for me! characterizing non-reproducible bug reports, in: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 62–71.

[8] J. Sun, Why are bug reports invalid? in: 4th International Conference on Software Testing, Verification and Validation, IEEE, 2011, pp. 407–410.

[9] S. Panichella, G. Canfora, A. Di Sorbo, "Won't we fix this issue?" qualitative characterization and automated identification of wontfix issues on GitHub, Inf. Softw. Technol. 139 (2021) 106665.

[10] M.M. Rahman, F. Khomh, M. Castelluccio, Why are some bugs non-reproducible?:–an empirical investigation using data fusion–, in: 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2020, pp. 605–616.

[11] Y. Su, P. Luarn, Y.-S. Lee, S.-J. Yen, Creating an invalid defect classification model using text mining on server development, J. Syst. Softw. 125 (2017) 197–206.

[12] A. Hindle, N.A. Ernst, M.W. Godfrey, J. Mylopoulos, Automated topic naming to support cross-project analysis of software maintenance activities, in: Proceedings of the 8th Working Conference on Mining Software Repositories, 2011, pp. 163–172.

[13] A. Hindle, C. Bird, T. Zimmermann, N. Nagappan, Do topics make sense to managers and developers? Empir. Softw. Eng. 20 (2) (2015) 479–515.

[14] M.S. Zanetti, I. Scholtes, C.J. Tessone, F. Schweitzer, Categorizing bugs with social networks: A case study on four open source software communities, in: 35th International Conference on Software Engineering, ICSE, IEEE, 2013, pp. 1032–1041.

[15] J. He, L. Xu, Y. Fan, Z. Xu, M. Yan, Y. Lei, Deep learning based valid bug reports determination and explanation, in: 31st International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2020, pp. 184–194.

[16] T.-H. Chen, S.W. Thomas, A.E. Hassan, A survey on the use of topic models when mining software repositories, Empir. Softw. Eng. 21 (5) (2016) 1843–1919.

[17] C.C. Silva, M. Galster, F. Gilson, Topic modeling in software engineering research, Empir. Softw. Eng. 26 (6) (2021) 1–62.

[18] L.V.G. Carreno, K. Winbladh, Analysis of user comments: An approach for software requirements evolution, in: 2013 35th International Conference on Software Engineering, ICSE, IEEE, 2013, pp. 582–591.

[19] A.B. Belle, G. El Boussaidi, S. Kpodjedo, Combining lexical and structural information to reconstruct software layers, Inf. Softw. Technol. 74 (2016) 1–16.

[20] L.B. Souza, E.C. Campos, F. Madeiral, K. Paixão, A.M. Rocha, M. de Almeida Maia, Bootstrapping cookbooks for APIs from crowd knowledge on stack overflow, Inf. Softw. Technol. 111 (2019) 37–49.

[21] K. Damevski, H. Chen, D.C. Shepherd, N.A. Kraft, L. Pollock, Predicting future developer behavior in the IDE using topic models, in: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 932–932.

[22] S.W. Thomas, H. Hemmati, A.E. Hassan, D. Blostein, Static test case prioritization using topic models, Empir. Softw. Eng. 19 (1) (2014) 182–212.

[23] S.K. Lukins, N.A. Kraft, L.H. Etzkorn, Bug localization using latent dirichlet allocation, Inf. Softw. Technol. 52 (9) (2010) 972–990.

[24] X. Zhang, Y. Yao, Y. Wang, F. Xu, J. Lu, Exploring metadata in bug reports for bug localization, in: 2017 24th Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2017, pp. 328–337.

[25] Y. Wang, Y. Yao, H. Tong, X. Huo, M. Li, F. Xu, J. Lu, Bug localization via supervised topic modeling, in: 2018 IEEE International Conference on Data Mining, ICDM, IEEE, 2018, pp. 607–616.

[26] Y. Wang, Y. Yao, H. Tong, X. Huo, M. Li, F. Xu, J. Lu, Enhancing supervised bug localization with metadata and stack-trace, Knowl. Inf. Syst. 62 (6) (2020) 2461–2484.

[27] J. Zou, L. Xu, M. Yang, X. Zhang, J. Zeng, S. Hirokawa, Automated duplicate bug report detection using multi-factor analysis, IEICE Trans. Inform. Syst. 99 (7) (2016) 1762–1775.

[28] T. Akilan, D. Shah, N. Patel, R. Mehta, Fast detection of duplicate bug reports using LDA-based topic modeling and classification, in: International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2020, pp. 1622–1629.

[29] A. Panichella, A systematic comparison of search algorithms for topic modelling—a study on duplicate bug report identification, in: International Symposium on Search Based Software Engineering, Springer, 2019, pp. 11–26.

[30] J. Zou, L. Xu, M. Yang, M. Yan, D. Yang, X. Zhang, Duplication detection for software bug reports based on topic model, in: 2016 9th International Conference on Service Science, ICSS, IEEE, 2016, pp. 60–65.

[31] A.T. Nguyen, T.T. Nguyen, T.N. Nguyen, D. Lo, C. Sun, Duplicate bug report detection with a combination of information retrieval and topic modeling, in: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, IEEE, 2012, pp. 70–79.

[32] T. Lin, W. Tian, Q. Mei, H. Cheng, The dual-sparse topic model: Mining focused topics and focused terms in short text, in: Proceedings of the 23rd International Conference on World Wide Web, 2014, pp. 539–550.

[33] W.X. Zhao, J. Jiang, J. Weng, J. He, E.-P. Lim, H. Yan, X. Li, Comparing twitter and traditional media using topic models, in: Advances in Information Retrieval: 33rd European Conference on IR Research, ECIR 2011, Dublin, Ireland, April 18-21, 2011. Proceedings 33, Springer, 2011, pp. 338–349.

[34] R. Mehrotra, S. Sanner, W. Buntine, L. Xie, Improving lda topic models for microblogs via tweet pooling and automatic labeling, in: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2013, pp. 889–892.

[35] J. Tang, M. Zhang, Q. Mei, One theme in all views: Modeling consensus topics in multiple contexts, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2013, pp. 5–13.

[36] R. Bibyan, S. Anand, A. Jaiswal, Latent Dirichlet allocation (LDA) based on automated bug severity prediction model, in: Proceedings of Data Analytics and Management, Springer, 2022, pp. 363–377.

[37] G. Yang, K. Min, J.-W. Lee, B. Lee, Applying topic modeling and similarity for predicting bug severity in cross projects, KSII Trans. Internet Inform. Syst. (TIIS) 13 (3) (2019) 1583–1598.

[38] G. Yang, T. Zhang, B. Lee, 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, pp. 97–106.

[39] J. Kim, G. Yang, Bug severity prediction algorithm using topic-based feature selection and CNN-LSTM algorithm, IEEE Access 10 (2022) 94643–94651.

[40] T. Zhang, G. Yang, B. Lee, E.K. Lua, A novel developer ranking algorithm for automatic bug triage using topic model and developer relations, in: 2014 21st Asia-Pacific Software Engineering Conference, Vol. 1, IEEE, 2014, pp. 223–230.

[41] W. Zhang, G. Han, Q. Wang, Butter: An approach to bug triage with topic modeling and heterogeneous network analysis, in: 2014 International Conference on Cloud Computing and Big Data, IEEE, 2014, pp. 62–69.

[42] N. Limsettho, H. Hata, A. Monden, K. Matsumoto, Unsupervised bug report categorization using clustering and labeling algorithm, Int. J. Softw. Eng. Knowl. Eng. 26 (07) (2016) 1027–1053.

[43] M.F. Zibran, On the effectiveness of labeled latent dirichlet allocation in automatic bug-report categorization, in: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion, ICSE-C, IEEE, 2016, pp. 713–715.

[44] N. Limsettho, H. Hata, A. Monden, K. Matsumoto, Automatic unsupervised bug report categorization, in: 2014 6th International Workshop on Empirical Software Engineering in Practice, IEEE, 2014, pp. 7–12.

[45] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw, Eng. 14 (2) (2009) 131–164.

[46] R. Rana, M. Staron, J. Hansson, M. Nilsson, W. Meding, A framework for adoption of machine learning in industry for software defect prediction, in: 9th International Conference on Software Engineering and Applications, ICSOFT-EA, IEEE, 2014, pp. 383–392.

[47] D.M. Blei, A.Y. Ng, M.I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.

[48] M. Pettinato, J.P. Gil, P. Galeas, B. Russo, Log mining to re-construct system behavior: An exploratory study on a large telescope system, Inf. Softw. Technol. 114 (2019) 121–136.

[49] W. Martin, F. Sarro, M. Harman, Causal impact analysis for app releases in Google play, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 435–446.

[50] J. Shimagaki, Y. Kamei, N. Ubayashi, A. Hindle, Automatic topic classification of test cases using text mining at an android smartphone vendor, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 1–10.

[51] Q. Luo, K. Moran, D. Poshyvanyk, A large-scale empirical comparison of static and dynamic test case prioritization techniques, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 559–570.

[52] H. Jiang, J. Zhang, X. Li, Z. Ren, D. Lo, X. Wu, Z. Luo, Recommending new features from mobile app descriptions, ACM Trans. Softw. Eng. Methodol. (TOSEM) 28 (4) (2019) 1–29.

[53] G. Canfora, L. Cerulo, M. Cimitile, M. Di Penta, How changes affect software entropy: An empirical study, Empir. Softw. Eng. 19 (1) (2014) 1–38.

[54] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, Methodbook: Recommending move method refactorings via relational topic models, IEEE Trans. Softw. Eng. 40 (7) (2013) 671–694.

[55] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, T. Zimmermann, What makes a good bug report? in: 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, pp. 308–318.

[56] S. Just, R. Premraj, T. Zimmermann, Towards the next generation of bug tracking systems, in: IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE, 2008, pp. 82–85.

[57] A. Bachmann, A. Bernstein, Software process data quality and characteristics: A historical view on open and closed source projects, in: The Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, 2009, pp. 119–128.