Capitalizing on Developer-Tester Communication – A Case Study

Prabhat Ram[1], Pilar Rodríguez[2], Antonin Abherve[3], Alessandra Bagnato[3], Markku Oivo[1]

[1] M3S, Faculty of ITEE, University of Oulu, Oulu 90014, Finland
[2] Faculty of Computer Sciences, Universidad Politécnica de Madrid, 28040 Madrid, Spain
[3] Softeam, 75016 Paris, France
{prabhat.ram, markku.oivo}@oulu.fi
pilar.rodriguez@upms.es
{alessandra.bagnato, antonin.abherve}@softeam.fr

Abstract. Communication between developers and testers can be a rich source of insights into software development processes and practices, which may not be easily discoverable from other means like retrospectives or project roadmaps. With the objective of deriving and capitalizing on potential development-related insights, we analyzed developer-tester communication in an industrial setting. We conducted a case study at a software-intensive Agile company, within the context of the development of one of their flagship products from 2016 to 2018. We applied Latent Dirichlet Allocation (LDA) to analyze communication between developers and testers, and then invited two case-company practitioners to study the results for insights into their developments processes*:* The findings reveal the case company's efforts to improve their product stability, the growing emphasis on addressing end-user concerns and other quality-related issues. The practitioners interpreted these findings as indicators of evolution in their development process. Based on these findings and the state of the art, we propose an *insight classification* to highlight insights discoverable from developer-tester communication*:* Recognizing LDA's potential for deriving insights, the practitioners are keen on incorporating it into their software development practices. The findings from this study serve as evidence for use and benefits of text-mining techniques like LDA in industrial setting, which other practitioners could adapt to elicit their own context-influenced insights. Furthermore, the *insight classification* can serve as a foundation for further investigation into the extent and type of insights discoverable from developer-tester communication.

Keywords: LDA, Topic Modeling, Insights, Software evolution.

1 Introduction

During software development, communication between developers and testers is documented in issue/bug trackers like Jira[1] and Mantis[2] in unstructured format. Unstructured data are expressed in natural language [1], and so do not have a clear, semantically overt, and easy-for-a-computer structure [2]. Software engineering is a data-rich activity [3], and developer-tester communication are among the software artifacts that are produced in large volume, particularly as a result of modern software development methods like Agile software development (ASD) [1]. These artifacts may hold insights into software design, developers' knowledge and decisions, and overall software advancement [1]. Their analysis can produce actionable information [4] to complement, and even improve, the overall software development process [5, 6].

Developer communication can be used to classify developer emails based on their purpose [7], identify source code activities in mailing list discussions [8], summarize software artifacts [9], and identify software architecture knowledge [10] from discussions on forums like Stack Overflow[3]. Such insights have been used for recommending developers for bug triage [11], mentoring [12], and for code comprehension [13]. In these investigations, information retrieval (IR) techniques like Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA) techniques have been preferred [7, 14–16]. LDA is more suitable for handling unstructured text, while retaining semantic richness, than other techniques like clustering and bag of words [17]. This may be why LDA is the most popular topic modeling technique for indexing, searching, and clustering large amount of unstructured data [5, 18]. Based on the surveys by Chen et al. [16] and Sun et al. [4], LDA has been used on source codes, requirement documents, bug reports, commit messages, and developer communication. However, its potential to derive insights from developer-tester communication remains unexplored.

In view of the potential insights discoverable from developer communication, exploration and analysis of developer-tester communication is also very likely to produce comparable results. If conducted in an industrial setting and during software product development, the results could help practitioners capitalize on their development data better by generating insights from them. Owing to such demonstrable positive findings, practitioners could expand and benefit from the use of this text-mining technique on other unstructured data, like change logs. To our knowledge, developer-tester communication in an industrial setting has not been studied, especially for discovering development-related insights. A study [19] similar to ours has used data-mining techniques to recover lost project knowledge from the informal communication conducted over instant messaging, and involved stakeholders from two startups to validate the results. However, in contrast to our study, the authors used a text-summarization algorithm and applied it only to developer communication.

[1] https://www.atlassian.com/software/jira
[2] https://www.mantisbt.org/
[3] https://stackoverflow.com/

With the objective of exploring developer-tester communication for development-related insights, we conducted a case study at a large software-intensive company using ASD. We targeted the unstructured data captured by Mantis, a bug-tracking tool used by the case company (CC). The data relates to discussion between developers and the testing team while addressing *issues*[4] revealed during testing, highlighted by the internal quality team, or reported by end users. We collected the data during the development of one of their flagship products *Modelio*, and we used LDA to analyze those data. We addressed our research objectives with the research question (RQ): *What software development related insights could be discovered from the developer-tester communication at the case company?*

We collaborated with two practitioners (henceforth stakeholders) from the CC. This was necessary to validate the LDA results, and understand their implications, particularly for the ongoing development of *Modelio.* Stakeholder involvement adds weight to our findings, as they are likely to elicit insights that are influenced by their development context. Based on these rationales, following are our study's contributions:

- In collaboration with stakeholders, we present empirical evidence of using LDA to elicit insights from developer-tester communication. Stakeholder-driven validation of the results helps retain the embedded development context. Since the results relate to one of the flagship products *Modelio*, the inferred insights carry a higher likelihood of influencing their ongoing software development processes and practices.
- We demonstrate the use of LDA in an industrial setting to elicit development-related insights, which the stakeholders claim would have remained undetected otherwise, even with the use of their classical monitoring tools and project roadmaps.
- Driven by the findings and state of the art, we propose a non-exhaustive *insight classification* to highlight examples of insights discoverable from developer-tester communication, recorded in issue trackers like Mantis.

In the remainder of the paper, we discuss LDA and related work in Section 2, research method in Section 3, followed by the study's results in Section 4. Discussion of the results is presented in Section 5, with limitations and threats to our research's validity in Section 6, and conclusion and future research directions in Section 7.

2 Background and Related Work

We first describe the LDA algorithm, which is central to our study, followed by a discussion on how our study relates to, and differs from, the state of the art.

2.1 LDA

LDA is one of the best topic modeling techniques to automatically extract topics from a corpus of text *documents* [20]. It creates statistical models to infer latent *topics* to

[4] At the case company, the term *issue* is used to refer to bugs, anomalies, defects etc.

describe a corpus. As a result, an unstructured corpus can be organized by their discovered semantic structure, represented by the topics embedded within the *documents* [21]. LDA identifies topics by using words that co-occur frequently in the *documents* of the corpus. This is due to the nature of natural language use, where frequently co-occurring words that constitute a *topic* are often semantically related [22]. Each *document* is a multi-membership of *topics*, which in turn is a multi-membership of words. This implies that each *document* can contain multiple *topics*, and conversely, each *topic* can appear in more than one *document*. By extension, each word can then appear in more than one *topic*. In this way, LDA can discover a set of ideas or themes that succinctly describe an entire corpus [20].

Formally, LDA infers for each of T topics an N-dimensional word membership vector $z(\phi 1{:}N)$ that describes the extent to which words appear in topic z. This membership vector describes the probability that each unique word appears in topic z. In addition, LDA infers for each *document* d in the corpus a T-dimensional topic membership vector $d(\theta 1{:}T)$, describing the extent to which each *topic* appears in d. This describes the probability that each topic appears in *document* d [21]. LDA makes these inferences using Bayesian techniques like Gibbs sampling [20].

2.2 Related Work

Anvik et al. [11] used Support Vector Machines on open bug repositories to identify relationships between developers and the bugs they fix, with the aim to propose a developer recommender system for bug triage. Similarly, Zhang et al. [23] used LDA to extract topics from bug reports, capture developers' interests and experiences vis-à-vis these bug reports, to propose a developer recommender system. To identify potential software development knowledge embedded in developers' discussions in mailing lists, Shihab et al. [8] used various heuristics to explore 22 GNOME projects. They identified that only a small group of developers dominate mailing list activity, and drew a correlation between mailing list activity and code activity, concluding that developers rely heavily on mailing lists to discuss source code changes. Also focusing on developers mailing list activity, Di Sorbo et al. [7] used natural language parsing to classify the mail content according to purpose of communication. The authors demonstrate the use of this approach to mine method descriptions from developers' communication. In a similar study, Panichella et al. [13] used Vector Space Model to automatically extract method descriptions from developer communications recorded in bug tracking systems and mailing lists. The authors used the approach to produce method descriptions from developer communication, and argued that such analysis could be used for code comprehension, which can be further used for source re-documentation.

Our study is similar to the above studies in the objective of extracting insights from unstructured data, such as developer communication. However, we target developer-tester communication, which may generate insights into both development and testing. Moreover, the study is conducted in an industrial setting, in collaboration with two stakeholders. According to a survey by Sun et al. [4] and the study by Zhang et al. [23], LDA has been used mainly for developer recommendations, which is marginally in contrast to our application of LDA on developer-tester communication. Excluding the study by Lima et al. [19], we have not found investigation similar to ours that involved stakeholders for interpreting and validating results.

Bertram et al. [24] classified issue trackers based on their potential utility for its users. The authors posit that issue trackers can act as a *knowledge repository, boundary object, communication and coordination hub,* and *communication channel.* We adapt this classification to propose a non-exhaustive *issue classification,* and highlight the development-related insights discoverable from developer-tester communication, recorded in issue trackers like Mantis. Although the foundation for this *classification* is our single case study, but by adapting the knowledgebase from [24], we aim to extend its relevance, encouraging further research to review, refine, or refute it.

3 Research Method

We followed the guidelines recommended by Runeson and Höst [25] to conduct the case study and answer the RQ.

3.1 Research Context

The CC is a large-size company, offering commercial services and solutions across multiple domains. The CC claims to follow *customized agile,* as it uses various software development methods that adhere to Agile principles, such as iterative development, but does not have any predefined sprint cycles. For the case study, we focused on one of their flagship products, *Modelio,* a modeling tool for model-driven development. A collocated team of nine practitioners works on *Modelio's* development. During our period of interest, the CC worked on and released three different versions of this tool.

3.2 Data Collection

For our study, we used data from the bug-tracking tool *Mantis*. The *issueDescription* and *testFeedback* fields in this tool record communication between the developers and testing team in natural language. The *issueDescription* field records *issues* raised by the testing team and even end users, and the developers respond with a fix. The testing team attempts to resolve the *issue* based on this response, and records the outcome in the *testFeedback* field. The data were collected for the years 2016 (189 entries), 2017 (571 entries) and 2018 (493 entries). *Mantis* 2019 dataset was too inadequate (66 entries) to be included in our study. By 'entries', we mean the total unique textual entries extracted from each year's dataset, both *issueDescription* and *testFeedback* fields taken together.

3.3 LDA Application and Data Analysis

We divided the *Mantis* dataset into three subsets, year wise. The year-based division approximates well to the three *Modelio* versions developed in 2016, 2017, and 2018. Moreover, a division of less than 12 months would have resulted in too few entries to produce any meaningful results, as we learned from the unintelligible topics produced from the analysis of *Mantis* 2019 dataset. Another reason for the year-based division is our previous study [26], where the same division logic was adopted to provide empirical evidence for the use and benefits of a metrics program in an industrial setting.

For applying LDA, we used the *tidytext*[5] format, where the text to be analyzed is stored as a table with one-*token*-per-row. Generally, a *token* is a single word, but can even be an *n-gram* (*n* words taken together), sentence, or paragraph [27]. A representative example of how we applied LDA to our dataset can be found here[6]. We created a *tidytext* data-frame for *issueDescription* and *testFeedback* corpora for each of the three subsets. In the context of LDA, *issueDescription* is the corpus, and the individual entries therein are the *documents*. This means that the 2016, 2017 and 2018 *Mantis* subsets have 189, 571 and 493 *documents*, respectively. Next, we preprocessed the *issueDescription* corpus by performing *tokenization*, splitting the *documents* into individual *tokens* (words). We used the *tidytext* R package[7] to perform this step, converting the text into *tidytext* format. Next, we removed stopwords, which are common English-language words like *"the", "of", "it"*, etc. Typically, numbers are also removed, but the corpus contained mentions of *Modelio's* different versions (e.g. *3.8.00*, *3.8.01*), instruction set architecture (e.g. *x86*, *64*), and operating system platforms (e.g. *10.0*). We retained them to avoid losing *tokens* of potential significance. Next, *tokens* like *'xmldiagramreader.java'* would typically be split into '*xmldiagramreader'* and *'java'* before applying LDA. However, we decided against it, because the original text holds more meaning, and is easily identifiable and interpretable for the stakeholders.

The preprocessing steps helped standardize the *issueDescription* corpus, which was done for every *testFeedback* corpus as well. Next, we calculated terms frequency–inverse document frequency (TF-IDF) for *issueDescription* corpora. TF measures how frequently a word occurs in a *document*. IDF also measures word frequencies, but by decreasing the weight for commonly used words and increasing it for words rarely used in the corpus. Combined, TF-IDF measures frequency of a word, adjusted for how rarely it is used, which helps identify how important a word is to a *document* in a corpus [27]. Although not necessary for topic modeling in general, TF-IDF is useful in exploring data and deriving information that can help inform topic modeling.

Next, we applied LDA to every *issueDescription* subset separately, to elicit *topics* that best describe each subset. The most essential input for LDA is the number of topics, which are typically user defined. After several attempts, we settled on different number of topics for different subsets. This decision was dictated by 'γ' distribution, which measures the probability of each *document* belonging to a topic. Higher number of topics result in too sparse distribution, indicating that the *documents* are not being sorted well into different topics. Decreasing the number of topics results in less clear division among topics, with multiple concepts clubbed under one topic. In addition, the stakeholders reviewed and validated the *topics*, aiding our decision on the number of *topics*.

LDA application divided every *issueDescription* corpus into *x* semantically similar but distinct *issues*-related topics. We wanted to explore if patterns observed in these

[5] https://www.tidytextmining.com/tidytext.html
[6] https://www.tidytextmining.com/nasa.html
[7] https://cran.r-project.org/web/packages/tidytext/index.html

topics had corresponding patterns in how the testing team addressed them. Based on the γ distribution, we joined each *issues*-related topic, and their corresponding *documents*, with the tokens generated from the *testFeedback* corpus. Independent analysis of the *issueDescription* corpus would have produced *topics* about only the *issues* the development team worked on in a given year, without any insight into their possible causes and how they were addressed. Similarly, analyzing the *testFeedback* corpus in isolation would have produced *topics* that provide some visibility into the testing efforts, but without the key insight into the *issues* those efforts were directed at. By combining the two corpora, we could explore one-to-many relationship between the distinct *issue*-related *topics* and how they were addressed by the testing team.

The LDA results were shared with the *Product Development Team Lead* and the *R&D Head* at the CC, the two stakeholders that we collaborated with. We asked them to study the findings to determine their significance from their development perspective, identify *issue*-related *topics* and the one-to-many relationship between these *topics* and the *testFeedback tokens*. Since LDA generates *topics* without labeling them, we asked the stakeholders to label them manually. Automatic labeling is an objective exercise [28]. Due to highly contextual knowledge embedded in unstructured data, we argue that manual labeling is preferable to automatic labeling, and that the stakeholders are in an ideal position to identify and interpret the *topics'* significance. The stakeholders provided their interpretations in under a week. We posit that this effort spent would be less if the LDA results are reviewed full-time, as part of daily work, instead of as a non-urgent task for an industry-academia collaboration. After receiving the stakeholders' interpretations of the results, we held an hour-long joint meeting with both the stakeholders for further clarification on their interpretations and claims made therein, which helped us answer the *RQ*.

4 Results

We first present the *topics* and their significance for each Mantis subset, interpreted by the stakeholders. Next, based on this empirical evidence, and the software development knowledge that issue trackers tend to capture [24], we present an *insight classification*, to characterize the potential of developer-tester communication for development-related insights. The LDA results the stakeholders studied and validated are available in the Appendix[8]. The R code for LDA application can be found here[9], but the Mantis raw data cannot be shared due to confidentiality reasons.

4.1 Development-related insights from developer-tester communication

The *topics* for all the Mantis subsets and their interpreted significance, based on their relation with the *testFeedback* tokens, is presented in Table 1. A more detailed table with a sample of both *issueDescription* and *testFeedback* tokens the stakeholders used

[8] https://doi.org/10.5281/zenodo.4761727
[9] https://github.com/prabhatram/devtester_topicmodeling

to infer the *topics'* significance is available in the Appendix. The '*NA*' entries mean the stakeholders could not find any meaningful one-to-many relationship between *test-Feedback tokens* and the corresponding *issues*-related *topics.*

Table 1. *Topics* extracted from *Mantis* dataset and practitioners' interpretation

No.	Topic	Practitioners' Interpretation
	Mantis 2016 subset	
1	Diagram	Implementation of the tool's *diagram* component
2	*Modelio* extensions	*Modelio* API to integrate new functionality
3	Core feature / Integration	Integration aspects of the tool and model storage layers
4	Project configuration	Project configuration and its external elements
5	Interoperability, Import/export	OS incompatibilities and issue reproducibility in different environment / version of *Modelio*
6	Model creation	*NA*
	Mantis 2017 subset	
1	Project lifecycle	*NA*
2	Diagram	*Diagrams* worked on during the development in 2017
3	Workbench support	Feature containing *workbench* implementation
4	BPMN metamodel evolution	*NA*
5	BPMN metamodel evolution	BPMN diagram implementation and import/export feature
6	ArchiMate metamodel support	*NA*
7	BPMN diagrams	
	Mantis 2018 subset	
1	General customer support	Customer relations
2	BPMN metamodel	*NA*
3	Methodological links	
4	*Modelio* module (extensions for *Modelio*)	Solution dedicated for the tool's module development
5	Document view	*NA*
6	Collaborative work / constellation	Collaborative work with Constellation
7	Diagrams	*Diagrams* definition, implementation, commands and controllers in Eclipse RCP

***Mantis* Topics.** The stakeholders claim that the six topics for the 2016 subset reflected their project structure and implementation of their product's lower layers. There was major work for redesigning the model kernels, and the related *issues* were identified in the topic '*Core feature / Integration*'. Stakeholders also claim that *issues* identified by the above topic and '*Project configuration*' highlight several non-development related anomalies that were relevant to the key components of *Modelio*. Overall, these topics reveal development process themes that could not have been discovered from sources like a project roadmap. The stakeholders regard these topics as evidence of development activities to improve stability of the core components of *Modelio*.

Of the seven topics for the 2017 subset, stakeholders could not find any relation between four of the topics and the corresponding *testFeedback* tokens. The tokens were too generic to provide visibility into how the *issues* were addressed. Still, the stakeholders identified an overarching theme, characterizing their development activities for that year. The focus was on implementation of new *Modelio* meta-model, as evidenced in the *'BPMN metamodel evolutions'* and *'ArchiMate metamodel support'* topics. Stakeholders identified two topics for the same *issue* of *'BPMN metamodel evolutions'.* Reducing the number of topics led to failure in identifying any relationship between the *issues* and the *testFeedback tokens,* and so we decided to retain this redundancy. Doing so helped the stakeholders identify a relation between one of the *'BPMN metamodel evolutions'* topics (#5) and the corresponding *testFeedback tokens,* thereby validating the decision to have seven topics for the 2016 subset. Next, the *'Workbench support'* topic also suggested the focus on development of new features. The topics *'BPMN metamodel evolution'* and *'BPMN diagrams'* highlighted the significant work that had begun on integrating BPMN standard. Stakeholders also claim that *issues* highlighted by the *'Diagram'* topic would have remained undetected without the use of LDA. Stakeholders also found *issues* related to user interface components, while *issues* related to *Modelio* core components, present in the 2016 subset, did not recur. This suggested that the development activities in 2017 grew closer to addressing end-user concerns.

Of the seven topics for the 2018 subset, no relation were found among three of the topics and the corresponding *testFeedback* tokens. Overall, these topics suggested a mixture of development activities for the year. The *'General customer support'* topic suggested an emphasis on addressing end-user reported *issues*, which were absent from both 2016 and 2017 findings. The stakeholders claim that this insight would have gone unnoticed if they had relied on their classical monitoring tools. The topics of '*Methodological links'* and '*Document view'* indicate new features development. However, the four topics of *'BPMN metamodel', 'Modelio module (extensions), 'Collaborative work / Constellation'* and *'Diagrams'* carry more significance. Stakeholders interpreted that the team emphasized on general quality improvement of several features, a development activity missing from both 2016 and 2017. Overall, 2018 development activity focused on general quality improvements of the products delivered.

Mantis is used to manage *issues* discovered by the quality team and reported by end users. *Issues* discovered by the quality team on newly developed features are more critical than those reported by end users, as density of the former is directly proportional to the quality of the development team's work. In 2016 and 2017, most topics point towards development of new features, which stakeholders interpret as quality problems with the products delivered, and under-representation of *issues* related to the quality of existing features (*GUI, end-user reported issues*). This may suggest either poor management or smooth resolutions of these *issues*. Conversely, there is an under-representation of *issues* related to development of new features in 2018. Stakeholders view this as their development process evolving from addressing issues affecting *Modelio's* core components, to addressing issues reported by end users and implementing general improvements. This is indicative of improvement in *Modelio's* quality, with its core components stabilizing, giving stakeholders more time to address end user reported concerns and the overall product quality.

Based on the stakeholders' interpretation, and the overarching development themes identified therein, we posit three development-related insights. First, bulk of the development activities in 2016 centered on developing new features and improving *Modelio's* stability by working on its core components. Second, in 2017 and particularly 2018, development activities focused on addressing overall product quality and *issues* reported by the end users. Viewing these development activities together, the third insight relates to how the development process evolved from emphasizing development of new features and *Modelio's* internal quality (core components stability) to emphasizing *Modelio's* external quality (end-user *issues*). Stakeholders point out that this evolution was natural, but would have remained undetected without this study.

Insights Classification. Based on the insights interpreted from the topics and the state of the art, we propose the following non-exhaustive *insight classification*, to characterize insights discovered from developer-tester communication recorded in *Mantis*.

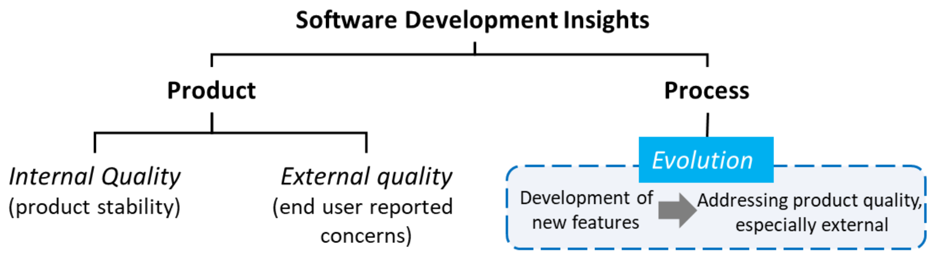
Software Development Insights
Product
Process
Internal Quality
(product stability)
External quality
(end user reported concerns)
Evolution
Development of new features
Addressing product quality, especially external

Fig. 1. *Insight Classification*

Stakeholders were specific in their interpretation of how different topics for different years afforded them visibility into their development processes and practices. Piecemeal, the insights highlighted the efforts to address and improve the *Modelio's* quality, both internal (core component stability) and external (end-user *issues*). Cumulatively, the insights are indicative of an evolution in the CC's development process from 2016 to 2018, as development efforts evolved from focusing on new features development to addressing product's quality, especially external quality.

5 Discussion

We first elaborate on the development-related insights the stakeholders inferred, and how they improve upon the benefits reported in the state of the art. We also discuss the '*insight classification'* from the standpoint of existing literature, and the relevance and utility that companies similar to the CC may derive from it.

5.1 Development-related insights from developer-tester communication

Despite the abundance of data generated as a result of modern software development methods like ASD, software development is still a risky endeavor, as existing tools are still inadequate at facilitating decision-making [3]. Software analytics can help address this shortcoming by automating processes to extract actionable information from data [29]. Still, companies struggle to capitalize on their development data, as a clear purpose for data collection [30, 31] and interpreting that data's significance [3] still remain a challenge. Our study's findings can help with both these challenges, especially for unstructured data that represent most of the development data produced [1]. The development-related insights that can be generated from unstructured data, with the help of text-mining techniques like LDA, could help companies capitalize on their data.

In contrast to the LDA benefits reported in the current literature, our exploration-centric approach helped identify insights of developers' and testers' engagement in addressing product stability, *issues* reported by end users, and general product quality, in addition to an overarching theme of process evolution. Stakeholders argue that these insights would have gone unnoticed without the study. These findings may be case specific, but they are evidence of LDA's use for extracting development-related insights in an industrial setting. LDA also helped the stakeholders discover transversal activities that was not documented in their project plan, something that their classical monitoring tools could not have detected. Insights related to addressing end-user reported *issues* and general quality improvements are part of background work and difficult to assess over time, as they are unplanned and, therefore, not monitored. Consequently, the stakeholders are now open to using LDA to analyze their development data at regular intervals. Incorporating a text-mining technique into the development process can help the CC create awareness among other stakeholders about the topics that may provide an early indicator of changes or problems warranting further attention.

In general, stakeholders are interested a system's history and how it evolves to understand their software development process [4]. Within the context of software evolution, how and why a software system changes can provide insights into both the specific software system and the software development as a whole [16]. LDA applied to the three-year Mantis data provided the CC stakeholders the historical knowledge, and the LDA *topics* helped them understand how and why their product development evolved, highlighting an evolution in their development process. Stakeholders' tacit contextual knowledge contributed to their interpretation of the LDA results, which lends plausibility to the above claim. Although LDA was applied to old data, the findings offer the stakeholders visibility into the development process of one of their flagship products, which can be leveraged to manage the development of future *Modelio* versions.

LDA results could be seen as summarization [15] of the developer-tester communication at the CC, which could be a useful way of presenting reusable software engineering knowledge to stakeholders [32]. Even though stakeholders-driven validation of LDA topics is seen as a critical requirement [33], this practice is not widely adopted [34], which reinforces our decision to include the stakeholders from the CC to study and validate the LDA results. In agreement with Hindle et al. [35], the stakeholders identified most of the relevant topics due to their familiarity of the concepts (conveyed by *tokens*), but had difficulty interpreting and labelling some, due to the *topics'* lack of significance. Furthermore, our study supports another finding from Hindle et al. [35], where most topics labelled by the stakeholders matched their perception of the development processes and activities for *Modelio* between 2016 and 2018. Overall, our study's findings are indicative of the importance and the necessity of including stakeholders to validate LDA results, when conducted in an industrial setting.

Based on the classification of issue trackers' utility by Bertram et al. [24], Mantis' use as a *coordination and communication hub* and *communication channel* is typical of any issue tracker. Mantis is used by developers, testers, and the quality team for addressing *issues*, and so its utility as a *boundary object* is also evident at the CC, where stakeholders utilize the stored data based on their custom needs and purposes. Most importantly, the overall process evolution the stakeholders interpreted points to Mantis serving as a *knowledge repository,* storing organizational knowledge that is difficult to detect otherwise. With this insight, and the potential future use of LDA at the CC to derive more such insights, stakeholders can plan their future development of *Modelio* better. For example, the stakeholders claim that once *Modelio's* core components were stabilized, the team could focus on the *issues* reported by the end users. *Modelio's* core components stability points to the product's *internal quality*, as the related *issues* were identified before the use of the tool by end users [36]. Conversely, *issues* reported by the end users relate to *Modelio's* external quality, as these *issues* were identified by them while using it [36]. The stakeholders can leverage this knowledge to estimate and allocate their efforts optimally. The *classification* is more relevant for other researchers and practitioners, interested in investigating insights that developer-tester communication may hold, and which could be extracted using text-mining techniques like LDA.

6 Limitations and Threats to validity

LDA is typically applied to large datasets, with thousands and millions of documents [37, 38]. Our dataset may be very small, but there exists no standard for sample size. The dataset should be large enough to generate distinct but not redundant topics, and small enough for topics that are not too broad and heterogeneous [39]. Although the small dataset is a limitation, stakeholders were able to extract distinct *topics* and non-trivial insights characterizing their development process from 2016 to 2018.

The number of topics is another limitation of our study. Similar to ideal sample size, there is no standard for ideal number of topics in LDA. Sbalchiero and Eder [39] provide a guide to aid in this decision, but those recommendations appear to be for sample sizes that are larger than ours. Informed by the γ distribution and stakeholders' validation, we have tried to address this limitation to some extent.

We address threats to our study's validity based on the guidelines recommended by Runeson and Höst [25]. We acknowledge that the developer-tester communication documented in Mantis does not capture the process of identifying and addressing *issues* in their entirety. Since the CC employs multiple tools in their software development, investigation of Mantis' data can provide only partial visibility into their development processes and practices. Even though the stakeholders claimed that the LDA topics helped highlight an evolution in their development process, the findings are still limited by the extent of information shared among the developers and the testers through Mantis. By including the *Product Development Team Lead* and the *R&D Head* in our study, we mitigate the threat to our study's *construct validity* to some extent. These stakeholders were better judges of the significance and validity of the *topics*, and how these *topics* represent the process of identifying and addressing *issues*.

Absence of contextual knowledge and other confounding factors interfere with the validity of interpretations of LDA *topics*. In our study, we relied on the CC stakeholders to interpret our findings and comment on its validity, as they possessed the necessary contextual knowledge to justify proper interpretation of their data. This helps us strengthen the *internal validity* of our study, but the potential of *confirmation bias* threatens it at the same time [40], and we acknowledge this tradeoff.

Being a single case study, *external validity* of our findings is affected. However, the objective of our study is to explore development-related insights that could be extracted from unstructured data using LDA, in an industrial setting. We hope that the positive findings help trigger more exploratory investigation to discover the extent of knowledge that can be extracted from such unstructured data. To this effect, we proposed an *insight classification*, which may be of utility to organizations with context similar to the CC. Similarly, interested researchers could build upon the *classification,* and conduct similar investigations to refine and supplement it.

Only one author was involved in collecting the data and applying LDA, and in creating the *classification*, which can affect the *reliability* of the study. However, our findings have been studied and validated by the collaborating stakeholders and the co-authors of this study, which helps mitigate the threat to our study's *reliability*.

7 Conclusion

Modern software development methods like ASD produce voluminous unstructured data on a daily basis, which may hold insights not easily discoverable from other means. Data related to developer communication have been leveraged to classify developer communication, propose developers for bug triage, aid in code comprehension, etc. However, the potential of developer-tester communication in generating development-related insights remains unexplored, especially in industrial setting.

We conducted a case study at a large software-intensive Agile company to explore development-tester communication for their potential to generate development-related insights. We applied LDA on this communication data from year 2016 to 2018, and invited two stakeholders to study and validate the results. Unexpectedly, they were able to find insights related to the development efforts to address their flagship product's internal quality, external quality and general improvements. The stakeholders identified another insight of an overarching theme of process evolution, as their development efforts evolved from addressing new features and internal quality to emphasizing on its external quality. The stakeholders claim that without the study, these insights may not have been discovered. Now, stakeholders are keen on incorporating LDA in their development process to keep track of these insights, despite the additional efforts the technique demands. We also proposed an *insight classification* to characterize the insights that we discovered. Companies will similar development context could use these insights to guide their analysis of developer-tester communication unstructured data. Interested researchers are encouraged to critique and improve upon this *classification*.

As part of future work, we plan to develop software metrics that help the stakeholders monitor and track the LDA *topics* they are interested in. This real-time tracking of the significant *topics* can remove the requirement of conducting LDA every *x* months to derive insights. Instead, stakeholders can apply LDA only when there are legitimate and major changes to the project, resulting in different sets of *topics*, which in turn could be used to update the metrics.

References

1. Haiduc, S., Arnaoudova, V., Marcus, A., Antoniol, G.: The use of text retrieval and natural language processing in software engineering. In: Proceedings - International Conference on Software Engineering. pp. 898–899 (2016).
2. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval. Cambridge university press. (2008).
3. Buse, R.P.L., Zimmermann, T.: Information Needs for Software Development Analytics. In: In 2012, the 34th International Conference on Software Engineering (ICSE). pp. 987–996 (2012).
4. Sun, X., Liu, X., Li, B., Duan, Y., Yang, H., Hu, J.: Exploring topic models in software engineering data analysis: A survey. In: 2016 IEEE/ACIS 17th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2016. pp. 357–362. Institute of Electrical and Electronics Engineers Inc. (2016).
5. Thomas, S.W., Hassan, A.E., Blostein, D.: Mining Unstructured Software Repositories. In: Evolving Software Systems. pp. 139–160 (2014).
6. Bettenburg, N., Adams, B.: Workshop on mining unstructured data (MUD): ···Because “mining unstructured data is like fishing in muddy waters”! In: Proceedings - Working Conference on Reverse Engineering, WCRE. pp. 277–278. IEEE (2010).
7. Di Sorbo, A., Panichella, S., Visaggio, C.A., Di Penta, M., Canfora, G., Gall, H.C.: Development emails content analyzer: Intention mining in developer discussions. Proc. - 2015 30th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2015. 12–23 (2016).
8. Shihab, E., Bettenburg, N., Adams, B., Hassan, A.E.: On the central role of mailing lists in open source projects: An exploratory study. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). pp. 91–103 (2010).
9. Vassallo, C., Panichella, S., Penta, M. Di, Canfora, G.: CODES: mining source code descriptions from developers discussions. In: In Proceedings of the 22nd International Conference on Program Comprehension. pp. 106–109 (2014).

10. Soliman, M., Galster, M., Salama, A.R., Riebisch, M.: Architectural knowledge for technology decisions in developer communities: An exploratory study with StackOverflow. In: Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016. pp. 128–133. IEEE (2016).
11. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? Proc. - Int. Conf. Softw. Eng. 2006, 361–370 (2006).
12. Canfora, G., Di Penta, M., Oliveto, R., Panichella, S.: Who is going to mentor newcomers in open source projects? Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng. FSE 2012. 1–11 (2012).
13. Panichella, S., Aponte, J., Di Penta, M., Marcus, A., Canfora, G.: Mining source code descriptions from developer communications. IEEE Int. Conf. Progr. Compr. 63–72 (2012).
14. Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshynanyk, D., De Lucia, A.: How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms. In: Proceedings - International Conference on Software Engineering. pp. 522–531 (2013).
15. Nazar, N., Hu, Y., Jiang, H.: Summarizing Software Artifacts: A Literature Review. J. Comput. Sci. Technol. 31, 883–909 (2016).
16. Chen, T.H., Thomas, S.W., Hassan, A.E.: A survey on the use of topic models when mining software repositories. Empir. Softw. Eng. 21, 1843–1919 (2016).
17. Sinoara, R.A., Scheicher, R.B., Rezende, S.O.: Evaluation of latent dirichlet allocation for document organization in different levels of semantic complexity. In: 2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017 - Proceedings. pp. 1–8 (2018).
18. Blei, D.M.: Introduction to Probabilistic Topic Models. Commun. ACM. 55, 77–84 (2012).
19. Lima, M., Ahmed, I., Conte, T., Nascimento, E., Oliveira, E., Gadelha, B.: Land of Lost Knowledge: An Initial Investigation into Projects Lost Knowledge. Int. Symp. Empir. Softw. Eng. Meas. 2019-Septemer, (2019).
20. Blei, D.M., Lafferty, J.D.: Topic Models. In: Text Mining. pp. 101–124. Chapman and Hall/CRC (2009).
21. Thomas, S.W., Adams, B., Hassan, A.E., Blostein, D.: Studying software evolution using topic models. Sci. Comput. Program. 80, 457–479 (2014).
22. Thomas, S.W., Adams, B., Hassan, A.E., Blostein, D.: Modeling the evolution of topics in source code histories. In: Proceedings of the 8th working conference on mining software repositories. pp. 173–182 (2011).
23. Zhang, T., Yang, G., Lee, B., Lua, E.K.: A novel developer ranking algorithm for automatic bug triage using topic model and developer relations. In: Proceedings - Asia-Pacific Software Engineering Conference, APSEC. pp. 223–230. IEEE (2014).
24. Bertram, D., Voida, A., Greenberg, S., Walker, R.: Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In: Proceedings of the 2010 ACM conference on Computer supported cooperative work. pp. 291–300 (2010).
25. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study

research in software engineering. Empir. Softw. Eng. 14, 131–164 (2009).
26. Ram, P., Rodríguez, P., Oivo, M., Bagnato, A., Abherve, A., Choraś, M., Kozik, R.: An Empirical Investigation into Industrial Use of Software Metrics Programs. In: International Conference on Product-Focused Software Process Improvement. pp. 419–433 (2020).
27. Silge, J., Robinson, D.: Text mining with R: A tidy approach. O'Reilly Media, Inc. (2017).
28. Allahyari, M., Kochut, K.: Automatic topic labeling using ontology-based topic models. Proc. - 2015 IEEE 14th Int. Conf. Mach. Learn. Appl. ICMLA 2015. 259–264 (2016).
29. Krishna, R., Agrawal, A., Rahman, A., Sobran, A., Menzies, T.: What is the Connection Between Issues, Bugs, and Enhancements? In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). pp. 306–315 (2018).
30. Bizer, C., Boncz, P., Brodie, M.L., Erling, O.: The meaningful use of big data: Four perspectives - Four challenges. SIGMOD Rec. 40, 56–60 (2011).
31. Holmström Olsson, H., Bosch, J.: Towards data-driven product development: A multiple case study on post-deployment data usage in software-intensive embedded systems. Lect. Notes Bus. Inf. Process. 167, 152–164 (2013).
32. Silva, C., Mariane, C.: Reusing software engineering knowledge from developer communication. In: ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 1682–1685 (2020).
33. Asuncion, H.U., Asuncion, A.U., Taylor, R.N.: Software traceability with topic modeling. In: Proceedings - International Conference on Software Engineering. pp. 95–104 (2010).
34. Hindle, A., Bird, C., Zimmermann, T., Nagappan, N.: Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers? In: 2012 28th IEEE International Conference on Software Maintenance (ICSM). pp. 243–252 (2012).
35. Hindle, A., Bird, C., Zimmermann, T., Nagappan, N.: Do topics make sense to managers and developers? Empir. Softw. Eng. 20, 479–515 (2015).
36. Gezici, B., Tarhan, A., Chouseinoglou, O.: Internal and external quality in the evolution of mobile software: An exploratory study in open-source market. Inf. Softw. Technol. 112, 178–200 (2019).
37. Pettinato, M., Gil, J.P., Galeas, P., Russo, B.: Log mining to re-construct system behavior: An exploratory study on a large telescope system. Inf. Softw. Technol. 114, 121–136 (2019).
38. Noei, E., Zhang, F., Zou, Y.: Too Many User-Reviews, What Should App Developers Look at First? IEEE Trans. Softw. Eng. 1–12 (2019).
39. Sbalchiero, S., Eder, M.: Topic modeling, long texts and the best number of topics. Some Problems and solutions. Qual. Quant. 54, 1095–1108 (2020).
40. Salman, I., Turhan, B., Vegas, S.: A controlled experiment on time pressure and confirmation bias in functional software testing. Empir. Softw. Eng. 24, 1727–1761 (2019).