Towards Interpretable Monitoring and Assignment of Jira Issues

Dimitrios-Nikitas Nastos[a], Themistoklis Diamantopoulos[b] and Andreas Symeonidis[c]
Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, Thessaloniki, Greece
diminast@ece.auth.gr, thdiaman@issel.ee.auth.gr, symeonid@ece.auth.gr

Keywords: Project Management, Task Management, Jira Issues, Topic Modeling

Abstract: Lately, online issue tracking systems like Jira are used extensively for monitoring open-source software projects. Using these systems, different contributors can collaborate towards planning features and resolving issues that may arise during the software development process. In this context, several approaches have been proposed to extract knowledge from these systems in order to automate issue assignment. Though effective under certain scenarios, these approaches also have limitations; most of them are based mainly on textual features and they may use techniques that do not extract the underlying semantics and/or the expertise of the different contributors. Furthermore, they typically provide black-box recommendations, thus not helping the developers to interpret the issue assignments. In this work, we present an issue mining system that extracts semantic topics from issues and provides interpretable recommendations for issue assignments. Our system employs a dataset of Jira issues and extracts information not only from the textual features of issues but also from their components and their labels. These features, along with the extracted semantic topics, produce an aggregated model that outputs interpretable recommendations and useful statistics to support issue assignment. The results of our evaluation indicate that our system can be effective, leaving room for future research.

1 INTRODUCTION

Nowadays, open-source projects are developed and maintained online using code hosting facilities like GitHub and monitored with issue tracking systems like Jira. This collaborative paradigm dictates that a project may have multiple contributors with different levels of expertise and experience, who must all work together in coordination to design and develop features, resolve issues/bugs, plan and craft releases, and generally monitor the development of the project.

As a result, contemporary issue tracking systems function as a hub of useful knowledge for project monitoring and decision making. The problem, however, is that, as the project grows, knowledge is harder to extract and transfer among existing and new contributors. And this may lead to several challenges. For instance, when a new bug arises, one must have a clear view of the project in order to be able to assess its impact, determine the relevant components that may be affected and assign it to the most relevant contributor.

In this context, several approaches have been developed for extracting information from issue tracking systems with the aim of helping developers better manage the software project under analysis. These approaches aspire to confront various challenges, including e.g. determining the most suitable developer for resolving a new issue (Murphy and Cubranic, 2004; Anvik et al., 2006; Matsoukas et al., 2020; Alkhazi et al., 2020), assessing the severity and/or the priority of a bug (Sharma et al., 2012; Tian et al., 2015; Kanwal and Maqbool, 2012; Diamantopoulos et al., 2021; Lamkanfi et al., 2010; Yang et al., 2012), or even extracting the roles of the different developers (Li et al., 2016; Onoue et al., 2013; Gousios et al., 2008; Lima et al., 2015; Papamichail et al., 2019).

Although these approaches are effective in certain scenarios, they also have important limitations. First of all, several approaches employ only the textual features of issues (i.e. titles and descriptions), thus disregarding features like the component hierarchy or the labels of the project, which may point to the expertise of the different contributors. Moreover, they do not always employ semantics-enabled methods, thus they may miss significant correlations between the different issues (and areas) of the project. Finally, and most importantly, they are usually built as black boxes and provide, at best, a probability. As a result, they do not help the contributors understand the reasoning behind

[a] https://orcid.org/0009-0007-2240-2835
[b] https://orcid.org/0000-0002-0520-7225
[c] https://orcid.org/0000-0003-0235-6046

recommendations so that they can make the optimal decisions for the project.

In this work, we present an issue monitoring and assignment methodology that helps contributors better understand the project and provides interpretable issue assignment recommendations. Our methodology uses a dataset of Jira issues and employs information both from the textual and from the grouping features (components, labels) of issues. Furthermore, we extract semantic knowledge from issues in the form of topics using the BERTopic topic modeling technique (Grootendorst, 2022) and build an aggregated model that assigns issues to contributors, while also providing useful statistics to support decision making.

2 RELATED WORK

As already mentioned, contemporary approaches that employ issue tracking systems confront various challenges, including automated issue assignment (Murphy and Cubranic, 2004; Anvik et al., 2006; Matsoukas et al., 2020; Alkhazi et al., 2020), bug severity or priority prediction (Sharma et al., 2012; Tian et al., 2015; Kanwal and Maqbool, 2012; Diamantopoulos et al., 2021; Lamkanfi et al., 2010; Yang et al., 2012), and even developer role extraction (Li et al., 2016; Onoue et al., 2013; Gousios et al., 2008; Lima et al., 2015; Papamichail et al., 2019). Our work lies in the scope of project monitoring and specifically in automated issue assignment, i.e. the problem of finding the most suitable developer for the task at hand.

Most approaches in automated issue assignment (or issue/bug triaging as it is also known) extract features from data lying in Jira or Bugzilla installations, use some type of textual model and employ classification algorithms to determine the most suitable developer based on any past issues he/she has resolved. One such approach is proposed by Murphy and Cubranic (2004), who employ a vector space model to map the bug reports of the Eclipse project and use Naïve Bayes to perform the classification, which is thus solely based on textual features. A similar approach is proposed by Anvik et al. (2006); the authors further incorporate the current workload and the vacation schedule of the different contributors, while they also use Support Vector Machines (SVM), managing to improve the accuracy of the assignments.

Though interesting, the aforementioned approaches rely on vector spaces based only on term frequencies, without incorporating the underlying semantics of issue text. To improve on this aspect, lately several researchers employ semantics-enabled methods, either using word embeddings (Guo et al., 2020; He and Yang, 2021) or using topic modeling techniques (Ahsan et al., 2009; Yang et al., 2014; Naguib et al., 2013). For instance, Guo et al. (2020) employ word embeddings to model issue titles and descriptions and use a Convolutional Neural Network (CNN) to produce a list of recommended developers along with the extracted probabilities. He and Yang (2021) further extend this line of thought by considering both word2vec and GloVe as representations and employing attention networks to make the final assignment.

Topic modeling techniques typically employ Latent Dirichlet Allocation (LDA) on the textual features of issues to extract semantic topics that are subsequently used to improve the classification. Yang et al. (2014) first use the extracted topics in order to detect similar bug reports, and then use the textual features of these reports in order to make the assignment. An interesting extension is proposed by Naguib et al. (2013), who further consider the issues resolved and the issues reviewed for every developer (instead of only the issues assigned to him/her). Ahsan et al. (2009) also extract semantic topics, however they employ Latent Semantic Indexing (LSI) and follow a slightly different approach for classification. Specifically, the authors first create a term-to-document matrix using a vector space model and then use the result of the LS in order to reduce the dimensionality of the matrix. Upon assessing different classifiers, they conclude that SVM provide the best performance.

Finally, there are also approaches that use code data. For instance, Alkhazi et al. (2020) process both issues and commits from the Eclipse project, while Matsoukas et al. (2020) use commits and issues extracted from GitHub (Diamantopoulos et al., 2020) to build models based on issue text, issue comments, labels, and commit comments. Though interesting, these approaches deviate from the scope of this work.

Although the aforementioned solutions are effective for automated issue assignment, they typically present assignment recommendations without considering interpretability. Most of them provide assignments and probabilities based on textual features, while certain systems do not capture the underlying semantics of the project, which can help understand the issues and even the expertise of contributors. Our methodology confronts the above limitations, by employing both textual and non-textual features, so that the recommendations incorporate the expertise of the developers in terms of the relevant components and semantic topics that each developer is familiar with. Furthermore, the output of our system is a ranked list, along with useful information for each developer in order to easily make an informed decision about the assignment of new issues.

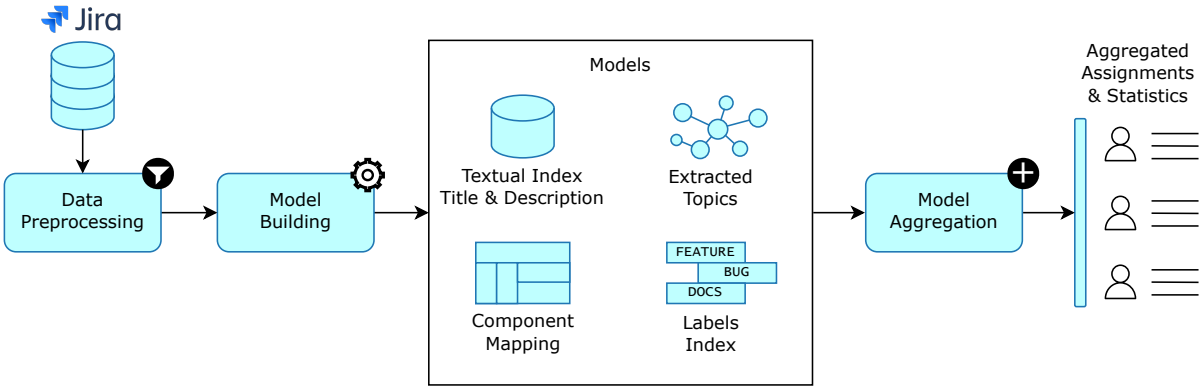


Figure 1: Overview of our Issue Mining Methodology

3 METHODOLOGY

The overview of our issue mining methodology is depicted in Figure 1. The input of our system is in the form of Jira issue tracking data, which are the issues of 656 projects of the Apache Software Foundation (Diamantopoulos et al., 2023). Each issue provides different types of information, including textual and non-textual features. In our case, we extract the title and description of the issue as well as the relevant labels and components that provide meta-data about it with respect to the project. Moreover, we extract the name of the assignee that solved the issue. As this information is necessary for the task of issue assignment, we filter out any issues that do not have data in those fields. Upon extraction, we preprocess the data and build different models: a textual index for the issue texts and description, a semantic topic model, and two mappings, one for components and one for labels. The semantic topic model is extracted using BERTopic, a topic modeling technique that uses the BERT deep learning language model so that we enable a deeper contextual understanding of the relationships between the different issues. Finally, upon extracting the information, we export useful diagrams that can be communicated to the users of the project (and especially the triager) and build an aggregated model that combines all extracted knowledge to produce interpretable issue assignments.

3.1 Data Preprocessing

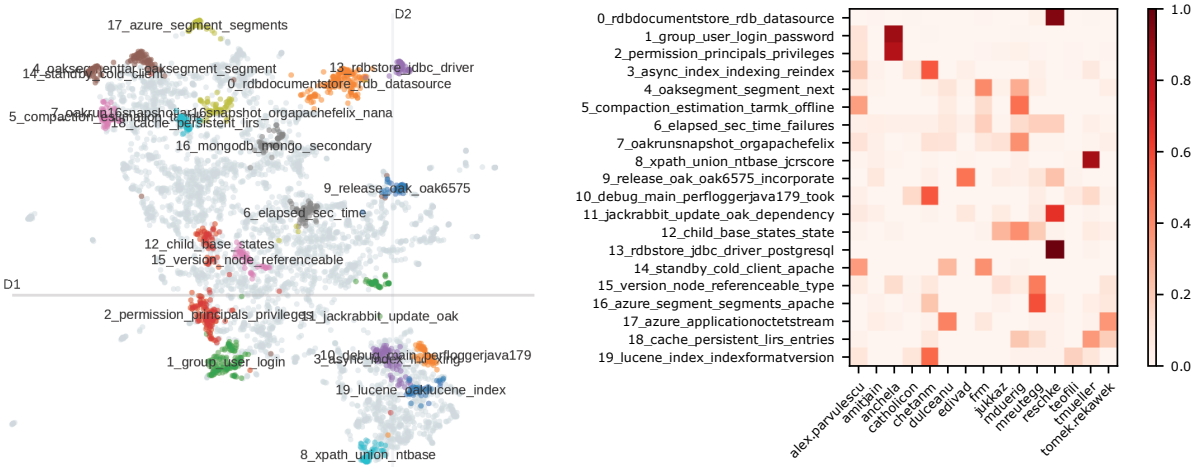
Our analysis is performed on a per-project basis. To effectively confront the issue assignment challenge, the issue data for each project must contain assignee information. Therefore, any issues lacking this information are excluded from the final dataset. Furthermore, to ensure meaningful results, we further filtered our dataset to only include developers who have resolved at least 100 issues and the corresponding issues. This cutoff ensures that the assignee has a sufficient history of issue resolution and thus the information can be used to extract better analytics.

For the textual features of our dataset, we combined two fields: the title (name summary in Jira) and the description. Before proceeding to our indexing and topic modeling methods, we first preprocess the textual data in two steps to produce two different versions of the text features, each suited for different tasks, the application of BERTopic and the application of Tf-Idf. The first step involves using regular expressions to remove HTML tags, special characters, and links, which are considered noise in our data. After this step, the data can be forwarded to the BERTopic topic modeling technique, which requires the full context of the text features. For the Tf-Idf model, we also proceed to the second step, which further refines the data to produce accurate representations of each text feature. It includes the conversion of all text to lowercase, the removal of stopwords, punctuation, and digits, and the lemmatization of the remaining words.

Following preprocessing, we split the issues for each project into training and test sets to train and evaluate the models we employ. The training set comprises 80% of the project issues, with the remaining 20% reserved for testing purposes.

3.2 Extracting Models for Issue Texts

To take into account the lexical overlap between issues and to improve issue assignment, we first use textual features. This involves transforming the text into vector representations and training a classifier to receive as input the text of an issue and provide a probability distribution of each potential assignee being the most suitable one for resolving that issue. We employ the Tf-Idf vectorizer to vectorize the text. This vectorizer creates a vocabulary of all the words in the issues



(a) Issue-Topic Distribution

(b) Topic-Developer Distribution

Figure 2: Distributions of the top 20 topics extracted

texts and then calculates the frequency of each word inside a document (Tf) and the inverse document frequency (Idf) of each word in the entire collection of documents. The Idf term is used to minimize the influence of very common words. Each document is represented by a vector of products of term frequencies and inverse document frequencies of each word. In specific, the value of each term t in a document d belonging to a collection of documents D is given by the equation:

$$Tf - Idf(t,d,D) = Tf(t,d) \cdot Idf(t,D) \qquad (1)$$

After implementing the Tf-Idf vectorizer and converting all text strings in the training set of the project to their corresponding vector representation, we train an SVM classifier. This classifier is designed to output a probability distribution that denotes the relevance of each contributor to the given issue.

3.3 Modeling Topics

Our methodology extracts topics from the issue texts to gain a better understanding of the semantics of the project under analysis and to facilitate the categorization of the issues. To achieve this, we have chosen to use BERTopic (Grootendorst, 2022), which is a topic modeling technique that outperforms conventional models such as LDA. BERTopic utilizes transformer-based language models like BERT, which enables it to identify semantic relationships between texts more effectively than bag-of-words models used by conventional models. BERTopic operates in three main stages: text embedding generation, dimensionality reduction, and cluster creation based on the new embeddings. To identify the most representative terms for each cluster, it employs class-based Tf-Idf (c-Tf-Idf) to generate topic representations.

We have applied BERTopic to the training set of projects, which has resulted in the extraction of topics for each project. By utilizing the trained BERTopic model, we can extract the number of issues assigned to each topic and the contribution of each assignee to each topic, indicating the number of issues assigned to each assignee for each topic. This information is then used to generate a distribution of each assignee across all topics. Each assignee gets a value between 0 and 1 for each topic, representing the percentage of the topic's issues that he/she has been assigned.

As an example, Figure 2 depicts the output of our BERTopic model (20 top topics) for the OAK project. Apache OAK is a hierarchical content repository for the Java platform, which is used as an example throughout our paper. It includes multiple features for storing and indexing structured and unstructured content and allows different querying methods (e.g. SQL, XPath). Indeed, using our visualizations, one can immediately see that the project has well-separated topics (Figure 2a) relevant to Lucene indexing (topic 19), to databases like MongoDB (topic 16), to XPath querying (topic 8), etc. Furthermore, given the topic-developer distribution (Figure 2b, triagers can easily identify the most relevant topics for each contributor at a glance. For instance, user tmueller seems to have extended expertise in XPath (topic 8), while user reschke is better acquainted with connecting databases (Postgres and JDBC in topic 13). Thus using this information, it is possible to classify issues to contributors according to their expertise, and, most importantly, to facilitate the final decision about issue assignment based on the relationship of the developers with the topic the issue belongs.

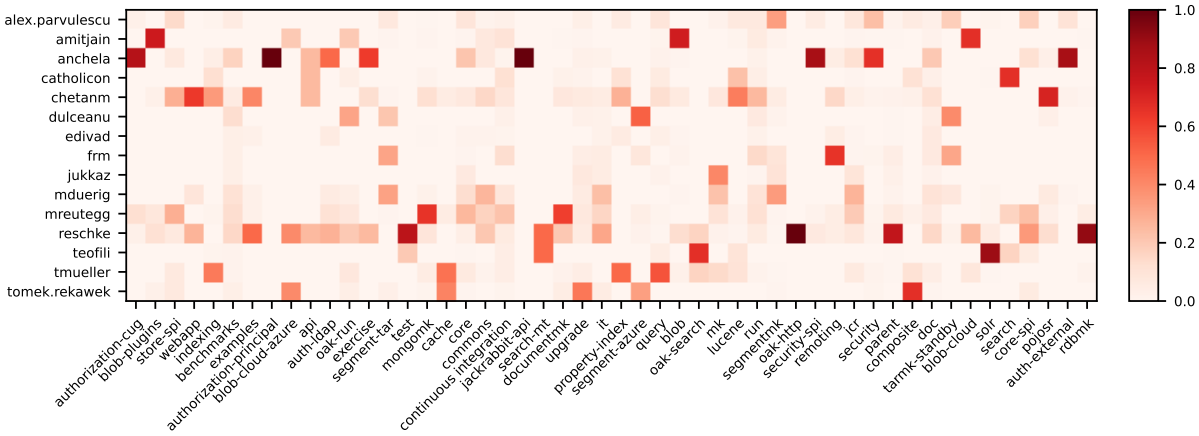


Figure 3: Heatmap of the Distribution between Components and Contributors

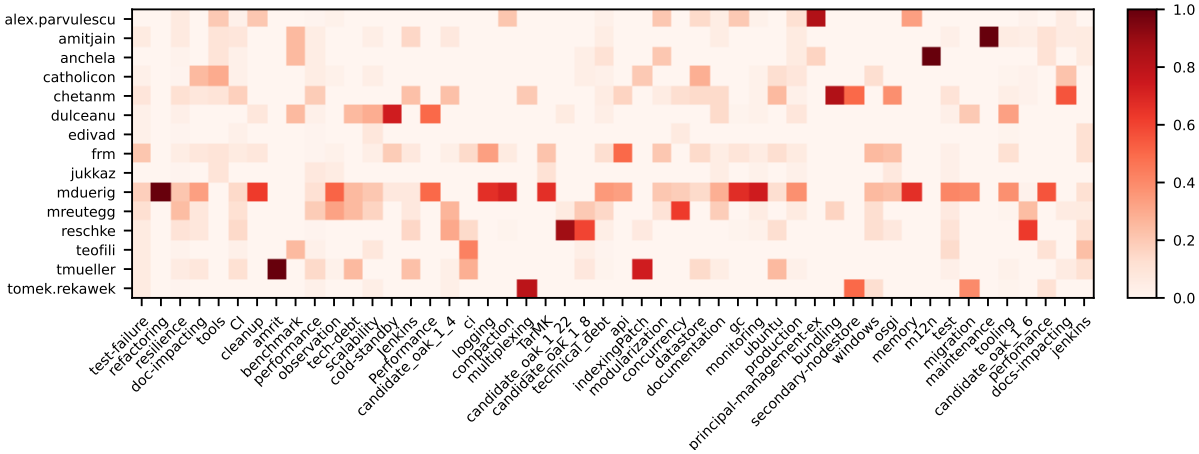


Figure 4: Heatmap of the Distribution between Labels and Contributors

3.4 Component Mapping

Another feature that can assist in the assignment of the most appropriate developer for an issue is the component to which the issue belongs. A component represents a subcategory of the project the issue is part of. For example, a database library could have components such as disk input/output, network communication, user interface, etc. When this information is available, we can use it to create a developer-component distribution, which corresponds to the percentage of issues for each component that have been assigned to each developer. In a similar manner to the assignee-topic distribution, this information can be utilized to make informed decisions about assigning issues to developers who are best suited to handle them based on their level of experience with the specific component. Figure 3 shows this distribution for project OAK. As before, one can also find useful connections; for instance, user teofili has handled most issues relevant to the SOLR indexing service, thus we may assume that the user has the relevant expertise. Interestingly, there are also cases where the expertise is shared. For example, concerning caching issues, these are mostly handled by two developers, tmueller and tomek.rekawek.

3.5 Indexing Labels

Labels are tags or categories that provide additional information which helps categorizing issues. Similarly to topics and components, they can provide useful insight when trying to make the best choice of developer for every issue in a project. In projects and issues where labels are available, they can be used for the creation of a label-contributor distribution. An example distribution for project OAK is shown in Figure 4. As depicted, labels can have several scopes; for instance, there are generic labels relevant to maintenance (e.g. ‘maintenance’ or ‘technical_debt’) or test-

Table 1: Example of Issue Assignment Monitoring

Issue Title: Prefetch external changes **Issue Description:** In a cluster with listeners that are registered to receive external changes, pulling in external changes can become a bottleneck. While processing those changes, local changes are put into the observation queue leading to a system where the queue eventually fills up. Instead of processing external changes one after another, the implementation could prefetch them as they come in and if needed pull them in parallel.
Contributor mreutegg (score 37.67% / issue text similarity 42.75%) Has resolved **31.82%** of the issues with topic **21_observation_events_listeners_listener** Has resolved **43.68%** of the issues in component **core** Has resolved **32.43%** of the issues with label **observation**
Contributor mduerig (score 30.27% / issue text similarity 26.84%) Has resolved **36.36%** of the issues with topic **21_observation_events_listeners_listener** Has resolved **6.52%** of the issues in component **core** Has resolved **51.35%** of the issues with label **observation**
Contributor chetanm (score 8.32% / issue text similarity 7.16%) Has resolved **18.18%** of the issues with topic **21_observation_events_listeners_listener** Has resolved **7.93%** of the issues in component **core** Has resolved **0%** of the issues with label **observation**

ing (e.g. 'test' or 'test_failure'), and of course there are also labels corresponding to different areas of the project (e.g. 'datastore' or 'osgi'). These can be used to identify the expertise of each developer and provide useful hints about his/her role(s) in the project (e.g. a developer that resolves issues with testing labels may be responsible for testing certain project modules).

3.6 Recommending Issue Assignments

After the previous steps, we now have 4 different distributions that can support issue assignment. When a new issue occurs in the system, the triager can use the models and the distributions produced for the corresponding project, combine their results and select the most suitable developer for the task according to the results of these calculations. Our system does this aggregation by computing the mean value of every potential assignee across the four distributions and recommends the top 3 most relevant assignees for the issue under triaging. Another important aspect of our system is that it does not only propose the most suitable assignees but it also indicates the reasons for their suitability based on the information that can be extracted from the distributions. This information can help the triager to select the most suitable developer, considering the characteristics of the issue. For example, he/she may choose to take into account only the topics-assignee distribution and no other information and thus assign the issue to the developer that has more experience in the topic of the issue. Or he/she may choose to base his/her decision on who has worked the most in the relevant component.

Table 1 depicts an example issue, along with the ranked list of potential assignees returned from our system. The issue originates from the OAK project and is relevant to improvements in an event processing scenario. It is part of the core component of the project (core component), while it is relevant to observation (label observation). Moreover, applying our topic modeling technique showed that it is related to topic 21 with top terms observation, events, listeners, etc. Upon producing the distributions for texts, topics, components, and labels, we compute also their aggregation by taking the average of the four different values for every assignee on each distribution.

The system returns the 3 assignees that have the highest average values, which in this example are mreutegg, mduerig and chetanm, along with useful statistics. Given the ranking of Table 1, one could immediately choose to assign the issue to mreutegg, who has the best aggregated score (37.67%). Indeed, mreutegg has also resolved issues that are textually similar, while also resolving a large fraction (43.68%) of issues related to the core component. However, concerning the topic of observation, mduerig seems to exhibit higher expertise, given that the contributor has resolved a significant amount of issues relevant to this topic, while also having resolved more than half of the issues with label observation. The actual ground truth assignment in this case is mreutegg, although mduerig would seem to be an acceptable choice. What is interesting, though, is that, using our methodology, one can truly understand which developer to choose and why. Thus, in this case, the triager could select the most active developer (component-wise) or the one

with the most expertise (topic and label-wise). And of course, he/she could also use the information provided in this ranked list to make a more complex assignment, e.g. assigning the task to mreutegg and set mduerig as the reviewer of the task.

4 EVALUATION

4.1 Evaluation Framework

In this section, we evaluate the performance of our system on 10 projects. These projects, the number of their issues and the number of their contributors that meet the requirements we have set in the previous section are shown in Table 2. To evaluate our system, we utilize the accuracy metric, which expresses the percentage of issues for which our system's first assignee choice is correct. Furthermore, we employ the Mean Reciprocal Rank (MRR) for all issues in the test set of each project. The MRR is computed as the average of the reciprocal rank of each issue assignment, where the reciprocal rank is the inverse of the rank of the correct assignee (e.g. if the assignee is in the second position, then the reciprocal rank for the issue is 1/2 = 0.5).

Table 2: Projects of the Evaluation Dataset

Project	#Issues	#Contributors
ARROW	7122	21
CXF	5987	12
FELIX	3941	10
GROOVY	6294	9
HDDS	2882	13
KARAF	5781	7
OAK	6879	15
OFBIZ	8477	20
SLING	7945	17
UIMA	5366	12

4.2 Evaluation Results

In Figure 5 we see that the accuracy results for all projects are higher that 50% and some of them reach almost 75%, which is a sufficient result, especially if we take into account the number of contributors. These results show that the proposed system can be very effective, even with returning only one choice as suitable. The results for MRR (Figure 6) are also quite encouraging; in all projects the MRR is larger than 0.6, meaning that on average the correct assignee is in the top 2 positions.

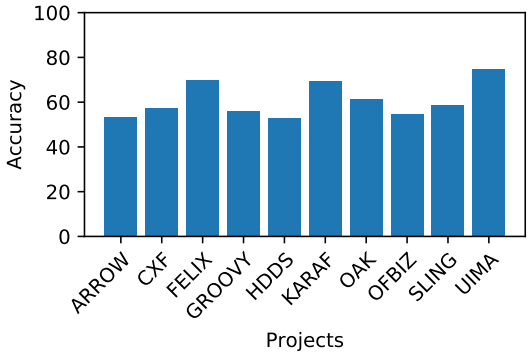


Figure 5: Accuracy of Issue Assignment per Project

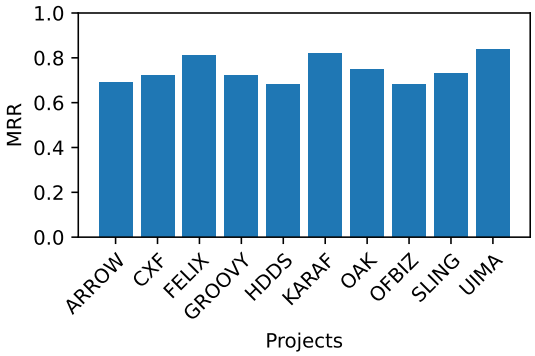


Figure 6: MRR of Issue Assignment per Project

5 CONCLUSION

Nowadays, effective collaboration in issue tracking systems can have significant influence on the software development process. In this paper, we focused on the challenge of automated issue assignment, extracting semantic topics from Jira issues with the aim of recommending the most suitable developer for resolving an issue. Unlike other approaches, our methodology employs information about the components, the labels, and the generated topics to produce a set of interpretable recommendations, thus truly supporting the decision making process. Upon assessing our system, we conclude that it can be effective for recommending assignees, while maintaining its intepretability.

Future work lies in several directions. First of all, we plan to build a graphical user interface in order to better illustrate the potential of our approach and better assess its impact. Moreover, one could test different combinations of features (e.g. issue type or severity) or even different of models, including e.g. word embeddings, and further assess the effectiveness of our methodology. Finally, an interesting idea would be to extend our system in order to cover other challenges, such as issue priority or severity prediction.

REFERENCES

Ahsan, S. N., Ferzund, J., and Wotawa, F. (2009). Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *Proceedings of the 2009 Fourth International Conference on Software Engineering Advances*, ICSEA '09, page 216–221, USA. IEEE Computer Society.

Alkhazi, B., DiStasi, A., Aljedaani, W., Alrubaye, H., Ye, X., and Mkaouer, M. W. (2020). Learning to rank developers for bug report assignment. *Applied Soft Computing*, 95:106667.

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA. ACM.

Diamantopoulos, T., Galegalidou, C., and Symeonidis, A. L. (2021). Software task importance prediction based on project management data. In *16th International Conference on Software Technologies*, ICSOFT 2021, pages 269–276, Held Online. SciTePress.

Diamantopoulos, T., Nastos, D.-N., and Symeonidis, A. (2023). Semantically-enriched jira issue tracking data. In *Proceedings of the 20th International Conference on Mining Software Repositories*, MSR '23, pages 218–222, Melbourne, Australia. IEEE.

Diamantopoulos, T., Papamichail, M., Karanikiotis, T., Chatzidimitriou, K., and Symeonidis, A. (2020). Employing contribution and quality metrics for quantifying the software development process. In *Proceedings of the IEEE/ACM 17th International Conference on Mining Software Repositories*, MSR '20, pages 558–562, Seoul, South Korea. ACM.

Gousios, G., Kalliamvakou, E., and Spinellis, D. (2008). Measuring developer contribution from software repository data. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, pages 129–132, NY, USA. ACM.

Grootendorst, M. (2022). Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*.

Guo, S., Zhang, X., Yang, X., Chen, R., Guo, C., Li, H., and Li, T. (2020). Developer activity motivated bug triaging: Via convolutional neural network. *Neural Process. Lett.*, 51(3):2589–2606.

He, H. and Yang, S. (2021). Automatic bug triage using hierarchical attention networks. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion*, pages 1043–1049.

Kanwal, J. and Maqbool, O. (2012). Bug Prioritization to Facilitate Bug Report Triage. *Journal of Computer Science and Technology*, 27(2):397–412.

Lamkanfi, A., Demeyer, S., Giger, E., and Goethals, B. (2010). Predicting the Severity of a Reported Bug. In *2010 7th IEEE Working Conference on Mining Software Repositories*, MSR '10, pages 1–10. IEEE Press.

Li, S., Tsukiji, H., and Takano, K. (2016). Analysis of Software Developer Activity on a Distributed Version Control System. In *Proceedings of the 30th International Conference on Advanced Information Networking and Applications Workshops*, pages 701–707. IEEE.

Lima, J., Treude, C., Filho, F. F., and Kulesza, U. (2015). Assessing developer contribution with repository mining-based metrics. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution*, pages 536–540, USA. IEEE.

Matsoukas, V., Diamantopoulos, T., Papamichail, M., and Symeonidis, A. (2020). Towards analyzing contributions from software repositories to optimize issue assignment. In *2020 IEEE International Conference on Software Quality, Reliability and Security*, QRS 2020, pages 243–253, Vilnius, Lithuania. IEEE Press.

Murphy, G. and Cubranic, D. (2004). Automatic Bug Triage using Text Categorization. In *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, SEKE '04, pages 92–97, USA. Knowledge Systems Institute.

Naguib, H., Narayan, N., Brügge, B., and Helal, D. (2013). Bug report assignee recommendation using activity profiles. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, page 22–30. IEEE Press.

Onoue, S., Hata, H., and Matsumoto, K.-i. (2013). A Study of the Characteristics of Developers' Activities in GitHub. In *Proceedings of the 20th Asia-Pacific Software Engineering Conference*, pages 7–12, USA. IEEE.

Papamichail, M. D., Diamantopoulos, T., Matsoukas, V., Athanasiadis, C., and Symeonidis, A. L. (2019). Towards extracting the role and behavior of contributors in open-source projects. In *14th International Conference on Software Technologies (ICSOFT)*, pages 536–543, Prague, Czech Republic. SciTePress.

Sharma, M., Bedi, P., Chaturvedi, K. K., and Singh, V. B. (2012). Predicting the Priority of a Reported Bug using Machine Learning Techniques and Cross Project Validation. In *2012 12th International Conference on Intelligent Systems Design and Applications*, ISDA 2012, pages 539–545. IEEE Press.

Tian, Y., Lo, D., Xia, X., and Sun, C. (2015). Automated Prediction of Bug Report Priority Using Multi-Factor Analysis. *Empirical Softw. Engg.*, 20(5):1354–1383.

Yang, C.-Z., Hou, C.-C., Kao, W.-C., and Chen, I.-X. (2012). An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection. In *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01*, APSEC '12, pages 240–249, USA. IEEE Computer Society.

Yang, G., Zhang, T., and Lee, B. (2014). Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, COMPSAC '14, page 97–106, USA. IEEE Computer Society.