10. November 2021

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4 Informatik

Prof. Dr. Ralf Lämmel
Hakan Aksu

Master Thesis Exposé

Amir Reza Javaher Dashti

The Influence of API Experience on Bug Assignment

1. Context and Motivations

Bug assignment is a crucial step of a bug's life cycle in software projects [1]. Bug assignment can be defined as the task of finding a ranked list of developers that are most qualified to resolve a newly arrived bug based on their previous contributions to the project [2, 3]. This task can be done manually by a developer or project lead known as triager [1, 4] however with the growing number of incoming bug reports [5], triaging becomes an increasingly time-consuming and costly process [6, 15].

A broad range of automated BA approaches exists in the literature. Initial BA approaches used text categorization to calculate a score for each developer using their historical bug reports [5]. Since expanding the sources of expertise improves the accuracy of BA [10] various machine learning techniques have been applied to source code activities [9] and other forms of contribution metadata, such as Stack Overflow contributions [8], pull request comments [10], and source interactions [7] to calculate the scores. Some researchers introduced a profile creation phase as a step to extract features needed for weighting from each source of expertise [4, 7, 11]. For instance, profiles include a mapping between a developer and tokens they used in commits and assigned bug reports along with their frequency and recency [13, 15]. We claim pure terms [15], modules [3], or locations [16] do not always reflect the semantics behind the code. Thus, a deeper insight into the code is required to present the expertise in developer profiles.åå

This thesis presents an API-aware [12] bimodal [25] approach, which extends the profiles to also track the frequency and recency of APIs. Here, we introduce a new module to dynamically calculate the API expertise of developers based on their previous commits. For a given bug report, the API expertise indexer gives a higher score to developers assuming they have the required experience to solve the issue. To the best of our knowledge, this level of code abstraction for deciding suitable fixers is unique. We hypothesize that this study provides a deeper semantic overview for the expertise of developers compared to existing methods. Finally, MAP and Precision@N measures [10, 20] are used to evaluate our approach against BSBA [13], MSBA [10], and ABA-Time-tf-idf [15].

2. Research Question

The key objective of this thesis is to highlight the impact of developers' API experience on assigning bug reports to them. Precisely, the following question is being addressed:

- RQ1: Can a Multi-source algorithm (MSBA) that focuses on the role API Expertise outperform state-of-the-art Bug Assignment Techniques?

- RQ2: What is the impact of API experience compared to other metrics of a developer's profile in bug assignment?

Hence, this work provides the undermentioned research contributions:

- Proposing a new Bug Assignment ranking approach for scoring the source code activities according to API usage.

- Creating a dynamic profile model for developers in the selected projects.

- Comparing the performance of the proposed algorithm with existing code-aware approaches.

3. Related Work

DevElect is one of the first BA algorithms that compared the vocabulary of the bug report with the source code contribution of developers [6]. This procedure involves parsing the repository changes (using diff command) to produce a term-author matrix and calculating vector space similarities. This contrasts with previous ML techniques that learned to classify reports using the vocabulary of existing reports [5]. Bugzie is a fuzzy technique that uses a set of significant technical terms that are present in assigned bug reports to construct developer profiles [11].

For a given textual description, Information Retrieval techniques can locate relevant code entities [24]. A location-based approach proposed the way to predict developers by first predicting the location of a new bug in the code using unigram language models and then suggesting to the developers that most recently fixed an issue in those files [16]. Similarly, iMacPro locates the relevant parts of the source code through Latent Semantic Indexing [9]. Next, these parts "are sorted based on their change proneness" [9]. Finally, the developers that maintained these units are retrieved and ranked as potential assignees.

BugFixer introduced Developer-Component Bug (DCB) network, as an abstraction for the relation between the developers, bug reports, and the source code modules they contributed to [3]. The mapping developers to components is close to the idea of mapping developers to APIs yet this algorithm does not rely on processing the code and constructs the network only with the chosen component(s) in the bug report.

ABA-Time-tf-idf emphasized the importance of time and decay of experience [6] when estimating the developer scores [15]. It extracts nouns from the newly arrived report and finds the frequency and recency of usage of them for each project member to ultimately calculate their time-aware tf-idf score. This implementation of BA does not take advantage of similar bug reports that happened in the past.

When a new change request arrives, iHdev employs Machine Learning techniques, K-nearest neighbor, and vector similarities to locate relevant files and then predict capable users according to the interaction traces (such as reading or editing) of the files [7]. The interaction metadata required for this technique should be collected separately and attached to each bug report. KSAP is another approach using KNN that initially searches for nearest similar (already resolved) bug reports. In the second phase, it ranks developers contributed to those chosen issues [23].

EDR_SI's authors introduced collaborative topic modeling to recommend developers according to their habits and files they changed before [22]. This method consists of a preprocessing step for the historical commits to rank developers and to provide personalized files that can help solve the issue.

VTBA is a time and vocabulary-aware BA that uses the text elements of the bug report [2]. The Multi-source approach investigated the information value of bug reports, bug comments, pull request text, PR comments commit message and its comments using the text analysis explained in VTBA and found this additional information improves BA [10]. A novel BA method used bug priority, average resolution time, and information gain (entropy) to generate developer profiles [4]. The procedure is like our way of forming profiles although it does not involve tracking user source-code contributions.

BSBA (Bug fixing and Source commit activity-based Bug Assignment) tracks both historical bug reports, and source entities used in commits to rank the developers [13]. Our approach extends BSBA to include the impact of API experience while calculating the developer weights. We also define a modular abstraction of profiles for this technique so other researchers can easily modify or add new weighting measures to their BA algorithm.

LCBPA employed content-based filtering to learn from previous features or behaviors of the developers based on their assigned tickets [21]. This work assumes the component in which the new bug has occurred is already provided in the report.

4. Methodology

4.1. Approach

Figure 1 illustrates the overall scheme of the present BA approach. When a new bug arrives, it first extracts the title and description texts and starts tokenization. This involves applying the usual pre-processing steps such as removing stop words, stemming, and converting the text to lower case. Next, the Profiler is triggered. We assume every user in the project can be potentially assigned to solve the bug therefore the Profiler loops through each developer in the project to create or update their profile. The Profiler has a modular setup which means each of the existing three parts can be independently calculated.

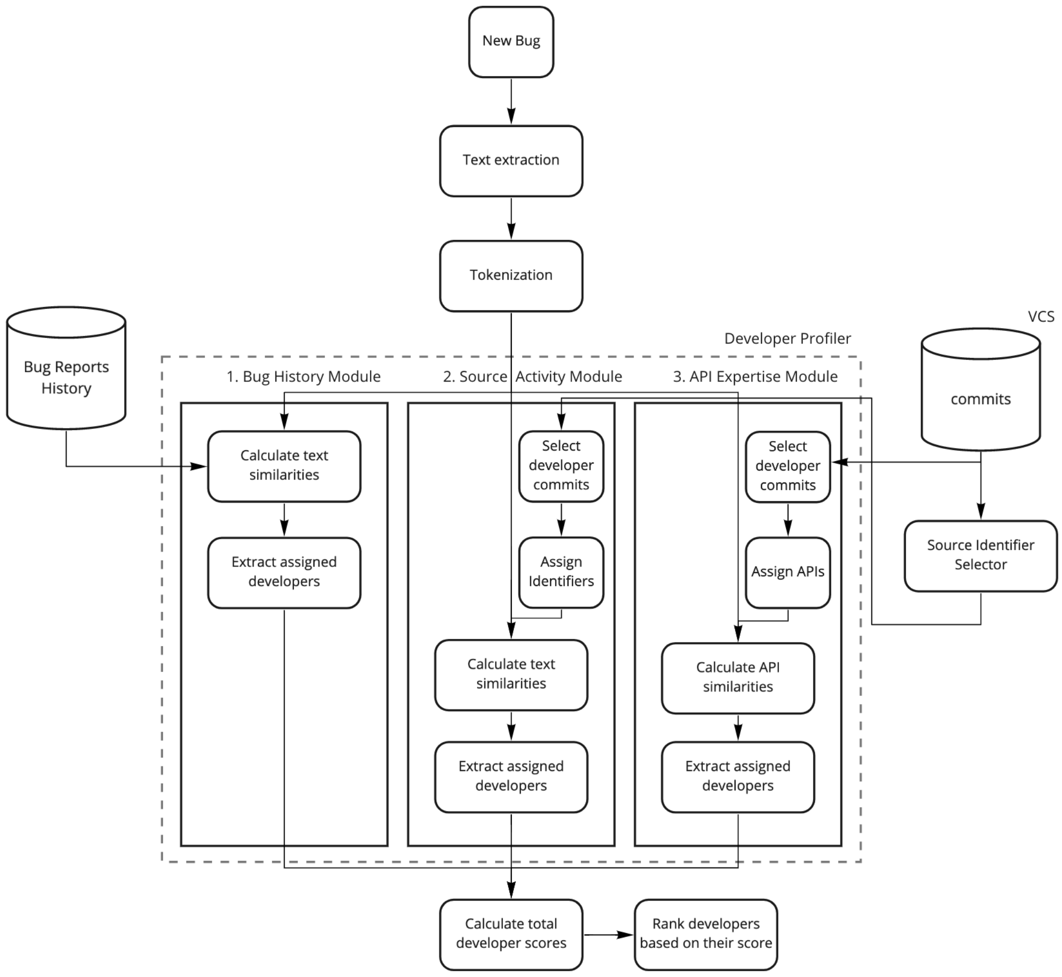
New Bug
Text extraction
Tokenization
Bug Reports History
Developer Profiler
VCS
commits
1. Bug History Module
2. Source Activity Module
3. API Expertise Module
Calculate text similarities
Extract assigned developers
Select developer commits
Assign Identifiers
Calculate text similarities
Extract assigned developers
Select developer commits
Assign APIs
Calculate API similarities
Extract assigned developers
Source Identifier Selector
Calculate total developer scores
Rank developers based on their score

Figure1. Overall schema of the modular API-aware BA

For each developer, the bug history module captures the tokens used in the previous reports that were assigned to them along with the frequency and recency (last usage time). These tokens are to calculate the similarity between the new bug report and the profile. The top k similar developer profiles are returned at the end of this module. It is noteworthy that the result of this similarity is equivalent to calculating the vector space similarity of the new bug report with each historical report and then returning the list of their assignees in other approaches [4, 13].

The source activity module estimates the developer score based on the BSBA algorithm [13]. It extracts class names, methods, and parameters as source entities then it loops through the commits and stores the token's frequency and time in the user profile. We have placed the extractor outside of the Profiler since it is an independent function that processes and source code and returns a set of tokens.

Lastly, the API expertise module is called to find each developer's top k API expertise. This module parses the commits for traces of API or library usage and then stores them on the profile. Like other steps, the number of times an API has been used and the recency of usage is also stored. To calculate the similarity of API expertise between the new bug report and the profiles, we must know what APIs are assumably required to resolve it. There are two possible implementations:

- Find similar historical bug reports to the given bug and assume the top API expertise of the assignee to those bugs are relevant for solving the issue.

- The second solution also starts by finding similar historical bug reports to a given bug then uses the code that was committed to resolve the bug and extracts the API expertise.

Since we already know the similar bug reports from the output of the bug history module the first solution does not need much computation power, but it may not always suggest correct APIs. Some developers may not have recorded API expertise when a new bug is reported yet be able to successfully resolve it. The alternative would be extracting the commits that solved similar bugs. There is no widely accepted format to determine a link between the commits and bug reports. An ad-hoc solution is to search for the keyword "fix" and bug id in the commit message [16]. We implement both similarity measures and report their performance.

After the Profiler module updates every profile, total developer scores can be calculated. Here, we compute the time-tf-idf using each token's frequency and last usage time [15]. Algorithm 1 describes how the total score is calculated using the three-profile metrics:

Algorithm 1: Developer Ranking

Input: a new bug report (B) and update developer profiles (profiles)

Output: sorted list of developers

```
1: for profile in profiles:
2:     fixationExperience = fixTfIdf() × fixRecency()
3:     usageExperience = usageTfIdf() × useRecency()
4:     apiExperience = apiTfIdf() × apiRecency()
5:     profile['score'] = fixationExperience + usageExperience + apiExperience
6: profiles.sortBy('score')
```

After calculating the total score for all developers, the top k highest-ranked profiles from the sorted list are returned as the suggested assignees.

4.2. Data

We have reviewed the BA approaches that have similar methodologies to this thesis. BSBA used Eclipse JDT and SWT bug reports [13]. ABA-Time-tf-idf studied bug reports from Eclipse, ArgoUML, and NetBeans [15]. The open-source project Mylyn and Eclipse are used by iHDev [4]. None of these papers have publicly shared the accumulated dataset employed for their algorithms. Further research also suggests that some existing approaches have reported biased results due to data leakage and misuse of the chronological order of events [18].

MSBA [10] is one of the BA algorithms that has published a dataset of bug reports and assignees along with the metadata (including pull request, report and commit comments) from thirteen open-source GitHub projects. To this day, this is one of the most comprehensive datasets available. We can easily adapt it for our task if we also pull the source codes for all the GitHub projects. Because forming a new dataset is a time-consuming and error prune job, we opt to use MSBA's dataset [17].

4.3. Evaluation

A variety of metrics have been proposed to evaluate the performance of the assignment. A guideline for evaluating BA concludes Mean Average Precision as the most informative metric [14]. Equation 1 denotes how MAP is computed [20]:

$$APi = \frac{\sum_k Precision@k}{\#\, of\ found\ relevant\ documents}$$

$$MAP = \frac{\sum_1^t APi}{t} \tag{1}$$

k: is the positions where a relevant document is found
APi: Average Precision for query i
t: number of queries

Top N rank [15, 19] and Precision@N [6, 20] are two other metrics that are widely reported in the literature. The value of N is arbitrarily defined but it is usually between 1 to 10 [14]. They refer to the first N results of the ranked list of developers to measure the accuracy by matching the expected assignee with the provided one. If the actual developer is present in the top N results, the answer is considered true and then the ratio of correct responses is calculated [14]. We use all three measures to compare the algorithms.

$$Precision@N = \frac{\#\, correct\ developers\ by\ rank\ N}{N} \tag{2}$$

5. Logistics

Duration of work: November 15, 2021 – May 15, 2022 (TBD).

Candidate: Amir Reza Javaher Dashti

Matriculation Number: 218203035

Email: javaher@uni-koblenz.de

Primary Supervisor: Professor Dr. Ralf Lämmel

Supervisor: Hakan Aksu

6. Preliminary Outline

1. Introduction
   1.1. Motivation
   1.2. Research Question
   1.3. Hypothesis
2. Literature Review
3. Methodology
   3.1. Approach
   3.2. Implementation
   3.3. Data
4. Evaluation
   4.1. Measures
   4.2. Results
   4.3. Comparison
5. Conclusion
6. Future Works
7. Threats to Validity
8. Bibliography

References

1. Lin, Zhongpeng, et al. "An Empirical Study on Bug Assignment Automation Using Chinese Bug Data." *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, https://doi.org/10.1109/esem.2009.5315994.
2. Sajedi-Badashian, Ali, and Eleni Stroulia. "Vocabulary and Time Based Bug-Assignment: A Recommender System for Open-Source Projects." *Software: Practice and Experience*, vol. 50, no. 8, 2020, pp. 1539–1564., https://doi.org/10.1002/spe.2830.
3. Hu, Hao, et al. "Effective Bug Triage Based on Historical Bug-Fix Information." *2014 IEEE 25th International Symposium on Software Reliability Engineering*, 2014, https://doi.org/10.1109/issre.2014.17.
4. Yadav, Asmita, and Sandeep Kumar Singh. "A Novel and Improved Developer Rank Algorithm for Bug Assignment." *International Journal of Intelligent Systems Technologies and Applications*, vol. 19, no. 1, 2020, p. 78., https://doi.org/10.1504/ijista.2020.10026839.
5. Anvik, John, et al. "Who Should Fix This Bug?" *Proceedings of the 28th International Conference on Software Engineering*, 2006, https://doi.org/10.1145/1134285.1134336.
6. Matter, Dominique, et al. "Assigning Bug Reports Using a Vocabulary-Based Expertise Model of Developers." *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009, https://doi.org/10.1109/msr.2009.5069491.
7. Zanjani, Motahareh Bahrami, et al. "Using Developer-Interaction Trails to Triage Change Requests." *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, https://doi.org/10.1109/msr.2015.16.
8. Sajedi Badashian, Ali, et al. "Crowdsourced Bug Triaging: Leveraging Q&a Platforms for Bug Assignment." Fundamental Approaches to Software Engineering, 2016, pp. 231–248, 10.1007/978-3-662-49665-7_14.
9. Hossen, Md Kamal, et al. "Amalgamating Source Code Authors, Maintainers, and Change Proneness to Triage Change Requests." *Proceedings of the 22nd International Conference on Program Comprehension - ICPC 2014*, 2014, https://doi.org/10.1145/2597008.2597147.
10. Sajedi-Badashian, Ali, and Eleni Stroulia. "Investigating the Information Value of Different Sources of Evidence of Developers' Expertise for Bug Assignment in Open-Source Projects." *IET Software*, vol. 14, no. 7, 2020, pp. 748–758., https://doi.org/10.1049/iet-sen.2019.0384.
11. Tamrawi, Ahmed, et al. "Fuzzy Set and Cache-Based Approach for Bug Triaging." *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering - SIGSOFT/FSE '11*, 2011, https://doi.org/10.1145/2025113.2025163.
12. Hakan Aksu, 2015, "Evolution-aware API analysis of developer skills", Master Thesis, University of Koblenz-Landau, Koblenz
13. Khatun, Afrina, and Kazi Sakib. "A Bug Assignment Approach Combining Expertise and Recency of Both Bug Fixing and Source Commits." *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2018, https://doi.org/10.5220/0006785303510358.
14. Sajedi-Badashian, Ali, and Eleni Stroulia. "Guidelines for Evaluating Bug-Assignment Research." *Journal of Software: Evolution and Process*, vol. 32, no. 9, 2020, https://doi.org/10.1002/smr.2250.

15. Shokripour, Ramin, et al. “A Time-Based Approach to Automatic Bug Report Assignment.” *Journal of Systems and Software*, vol. 102, 2015, pp. 109–122.,https://doi.org/10.1016/j.jss.2014.12.049.
16. Shokripour, Ramin, et al. “Why so Complicated? Simple Term Filtering and Weighting for Location-Based Bug Report Assignment Recommendation.” *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013,https://doi.org/10.1109/msr.2013.6623997.
17. https://github.com/TaskAssignment/MSBA
18. Tu, Feifei, et al. “Be Careful of When: An Empirical Study on Time-Related Misuse of Issue Tracking Data.” *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, https://doi.org/10.1145/3236024.3236054.
19. Rao, Shivani, and Avinash Kak. “Retrieval from Software Libraries for Bug Localization.” *Proceeding of the 8th Working Conference on Mining Software Repositories - MSR '11*, 2011, https://doi.org/10.1145/1985441.1985451.
20. Anders Hast, “Consensus Ranking for Increasing Mean Average Precision in Keyword Spotting”, roceedings of 2nd International Workshop on Visual Pattern Extractionand Recognition for Cultural Heritage Understanding co-located with16th Italian Research Conference on Digital Libraries {(IRCDL} 2020.
21. Tahir, Hassan, et al. “LCBPA: An Enhanced Deep Neural Network-Oriented Bug Prioritization and Assignment Technique Using Content-Based Filtering.” *IEEE Access*, vol. 9, 2021, pp. 92798–92814., https://doi.org/10.1109/access.2021.3093170.
22. Sun, Xiaobing, et al. “Enhancing Developer Recommendation with Supplementary Information via Mining Historical Commits.” *Journal of Systems and Software*, vol. 134, 2017, pp. 355–368., https://doi.org/10.1016/j.jss.2017.09.021.
23. Zhang, Wen, et al. “KSAP: An Approach to Bug Report Assignment Using KNN Search and Heterogeneous Proximity.” *Information and Software Technology*, vol. 70, 2016, pp. 68–84., https://doi.org/10.1016/j.infsof.2015.10.004.
24. Kagdi, Huzefa, et al. “Assigning Change Requests to Software Developers.” *Journal of Software: Evolution and Process*, vol. 24, no. 1, 2011, pp. 3–33., https://doi.org/10.1002/smr.530.
25. Miltiadis Allamanis, Daniel Tarlow, Andrew D. Gordon, and Yi Wei “Bimodal modelling of source code and natural language” In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15). JMLR.org, pp. 2123–2132.