2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) | 978-1-6654-0403-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/SNPD51163.2021.9704950

Interactive ChatBots for Software Engineering: A Case Study of Code Reviewer Recommendation

Noppadol Assavakamhaenghan, Raula Gaikovina Kula and Kenichi Matsumoto
Nara Institute of Science and Technology, Nara, Japan
Email: {noppadol.assavakamhaenghan.mt8 , raula-k, matumoto}@is.naist.jp

***Abstract*—Recommendation systems have played a large role in the Software Engineering research landscape. Applications have ranged from source code elements, APIs and reviewer recommendations, with techniques borrowed from the Information Retrieval, and Machine Learning domains. In recent times, there has been work into a new method of interaction, which is ChatBots, especially for Software Engineering. Early work has been aimed at using bots for mining software repositories, providing task-oriented feedback for the software developer. In this work, we would like to take the ChatBots one step forward, but using them inconjunction with recommendation systems to provide an interactive experience for recommendations. As a case study, we focus on the existing reviewer recommendation systems, and propose how using a ChatBot may enhance the solution, to provide a more accurate and realistic recommendation for the practitioner. In the end, we highlight the potential and next steps to utilize ChatBots into existing Software Engineering recommendation systems.**

I. Introduction

Recommendation Systems has have a large impact in the Software Engineering research landscape [15]. This is due to the growth of public and private data stores and data-mining technology for software engineering data. Applications have ranged from source code elements, APIs and reviewer recommendations, with techniques borrowed from Information Retrieval, and Machine Learning. Using historical data, recommendation systems typically return a ranked list of recommendations based on certain objectives. Furthermore, using other optimizations techniques, such as search-based, recommendations can provide solutions based on either a single or multiple objectives that the user would like. One of the key issues, is that one solution for one person might not work for another. For instance, [8] showed that reviewer recommendations systems fall into two categories of either effort prioritization or expertise recommendation. Hence, there is a need for interaction between a software developer to get the optimal solution.

In recent times, there has been work into automated interaction, which is ChatBots for Software Engineering. As highlighted by Abdellatif et al. [1], ChatBots are envisioned to dramatically change the future of Software Engineering, allowing practitioners to chat and inquire about their software projects and interact with different services using natural language. From a research point of view, ChatBots are in their infancy, with early work has aimed at using bots for mining software repositories, providing vital information for the software developer. On the other hand, Open Source has already implemented bots, with limited interactions. A popular example includes the popular *Dependabot*[1], that is used to help with automatic updating of an application dependency to patch a vulnerability in a exposed dependency. Although most of these ChatBots are task-oriented, they lack the interaction with the user to extract features needed for the recommendation.

In this work, we use the reviewer recommendation system as an example to demonstrate how ChatBots need to be more interactive, as there are complex algorithms and features to get the most optimal solution. First, we present key related work on recommendation systems and ChatBots in the Software Engineering context. As a case study, we focus on the existing reviewer recommendation systems, and propose how using a ChatBot may enhance the solution, to provide a more accurate and realistic recommendation for the practitioner.

II. Background

In this section, we present key related work for recommended systems and ChatBots in Software Engineering.

A. Recommendation Systems in SE

In this section, we briefly describe how recommendation system are begin used in a Software Engineering context. Also all techniques are aimed at providing developers with a list of best solutions for developer, based on historical data that is mined from a software repository. Furthermore, the techniques employed use machine learning (e.g., supervised learning and non-supervised learning). For instance, Tuarob et al. [18] propose the team recommendation algorithm for software development tasks by considering team collaboration and expertise feature stored as a knowledge graph. They adopt a machine learning model (e.g., random forest) to the data from Moodle, Apache, and Atlassian projects.

Other works have employed more sophisticated techniques, largely due to the recent availability and diversity of data that has become available. Zhang et al. [23] propose a developer recommendation system for Topcoder challenge, a crowd-sourcing software development platform. By using data from Topcoder, their approach adopts a meta-learning-based policy model (e.g., ExtraTrees, XGBoost, and Neural Network) with Topcoder challenge and developer features such as challenge duration, developer technical skill, and historical performance. Li et al. [11] introduces a recommendation system for recommending third-party libraries for mobile app developers. The

[1]https://github.com/dependabot

TABLE I
SUMMARY OF RECENT REVIEWER RECOMMENDATION TECHNOLOGIES

Paper	Method	Reviewer Recommendation Features
Asthana et al. [3]	Heuristic	Reviewer experience on file and directory Workload
Al-Zubaidi et al. [2]	Machine Learning	Historical review participation
Thongtanunam et al. [17]	Heuristic	File location similarity
Xia et al. [19]	Hybrid	File location similarity Textual information in review
Yang et al. [21]	Hybrid	File location similarity Textual information in review Reviewer expertise Type of review (technical or managerial)
Rahman et al. [14]	Heuristic	Library usage experience
Xia et al. [20]	Machine Learning	Comment in review Date of comment
Chouchen et al. [5]	Machine Learning	Reviewer experience on file Developers collaboration Workload
Zanjani et al. [22]	Heuristic	Reviewer experience on file
Sülün et al. [16]	Heuristic	Path length in software artificial network

system use graph neural network-based approach with feature extracted from interactions between mobile applications and third-party libraries. He et al. [9] proposed an algorithm for recommending Python API for developers in real-time by using data from 8 popular Python projects. They used random forest, a machine learning model with the features extracted from data-flow, token similarity, and token co-occurrence, in the context of the program point where a recommendation is solicited. Izadi et al. [10] propose topic tags recommendation algorithm for GitHub repositories. They adopt multi-label classification techniques and textual features (e.g., description, a README file, wiki pages, and file names.)

B. ChatBots (Bots) in Software Engineering

An emerging research field in Software Engineering is ChatBot. By incorporating ChatBot's ability to interpret the natural language, the software development process can be improved in many ways.

Most of the work performed for ChatBots is task-oriented, and is designed to provide solutions for developer tasks. Lin et al. [12] develop ChatBot system, namely MSABot (Microservice Architecture Bot), to assist in the development and operation of the microservice-based system. Users can ask the MSABot about the current status of microservice development and operation, including system errors or risks to users. Okanović et al. [13] introduce ChatBot, namely PerformoBot, which assists developers in configuring and running load tests, a process to test for the performance of the software under a certain amount of users. It helps to guide developers through the process of adequately specifying the parameters of a load test (e.g., number of the simulated user). PerformoBot can execute the load test and provides developers a report that answers the respective concern (e.g., latency). PerformoBot uses Dialogflow to process natural language and the Vizard framework to execute the test and generate the report. Dwitama and Rusli [6] propose an Android ChatBot application to support the requirements elicitation activity, a process to gather the software requirements, in software engineering. The ChatBot uses Nazief & Adriani stemming algorithm to preprocess the natural language it receives from the users and artificial mark-up language (AIML) as the knowledge base to process the ChatBot's responses. The ChatBot gathers the user stories from the user and reports them. Ed-Douibi et al. [7] present OpenAPI Bot, a ChatBot able to read an OpenAPI definition, which is the documentation that contains all of the available API operations and corresponding parameters. OpenAPI Bot can answer the questions regarding the API (e.g., "How to use get this information.") It utilizes Xatkit, a flexible multi-platform ChatBot development framework. Cerezo et al. [4] propose the ChatBot system that can recommend experts for a software development task. The ChatBot uses the term frequency technique for sentence classification and inverse document frequency technique to identify the keywords (e.g., skills and expertise) user asking. Then it uses the keywords to recommend experts based on the historical tasks. We find that most of these works do not include a ChatBot with deep interactive features.

III. CASE STUDY: REVIEWER RECOMMENDATION IN SE

To understand the need for more interactive solutions, we describe the case studies on review recommendation techniques. Recent studies proposed the approach of reviewer recommendation using various techniques which can be categorized into three types: heuristic, machine learning, and hybrid approach.

Most of studies use heuristic approach, which is a simple approach that involve an assumption of the suitability of reviewer. For example, Thongtanunam et al. [17] highlight an idea that files that are located in similar file paths would be managed and reviewed by similar experienced code-reviewers. They then propose a heuristic approach of reviewer recommendation by utilizing the historical reviewed file path and recommend reviewer based on similarity of the previously reviewed file path. Improving on this, Asthana et al. [3] propose the

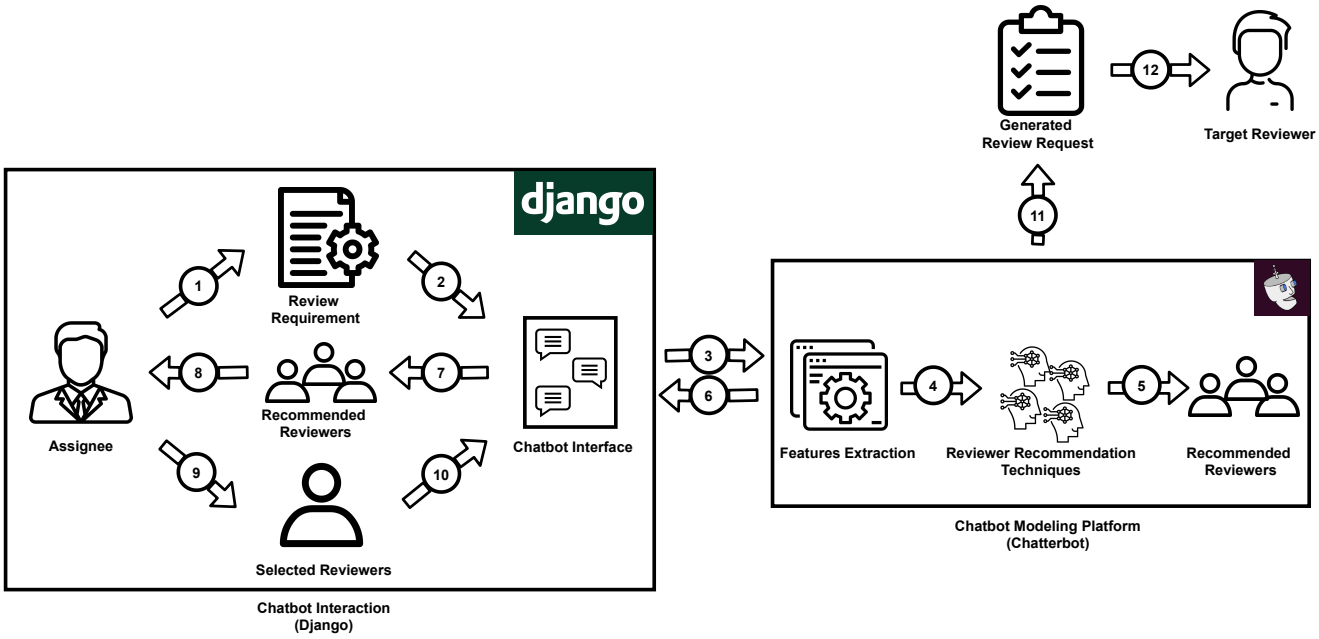

Fig. 1. Overview of Our ChatBot system for Reviewer Recommendations in Software Engineering.

automating reviewer recommendation approach utilizing the historical data of review and file change. It also incorporates the load-balancing system to consider the reviewer's workload before recommending the reviewer. Rahman et al. [14] propose a code reviewer recommendation technique that considers the relevant cross-project work history (e.g., external library experience) and the experience of a developer in certain specialized technologies (i.e., library usage) associated with a Pull-Requests for determining their expertise. Xia et al. [20] introduces a recommendation algorithm that considers the relationship between reviewers and historical reviews. They extract explicit relations (e.g., number of review comment reviewer give in the review and the date of review) from the dataset. They also generate implicit relations if explicit relations are not found. They adopt a hybrid approach that combines latent factor models (e.g., Singular Value Decomposition) and neighborhood methods to calculate implicit relation. They use stochastic gradient descent to optimize their feature weight. Zanjani et al. [22] present an approach, namely cHRev, to automatically recommend reviewers based on their historical contributions. The approach considers reviewer experience on changed files and the number of reviewed change files. Sülün et al. [16] propose an idea of recommending reviewer by using software artificial network (i.e., a network that represents the relationship of software artifact such as developer, file, build, bug report, use case diagram, and test case.) The approach recommends the reviewer with the highest sum of the inverse length of available paths to the new review files and considers modification time.

With the emergence of software repositories and historical data, there has been a growth in applying machine learning techniques to reveal key features for recommendation. For example, Al-Zubaidi et al. [2] introduce the reviewer recommendation approach utilizing the reviewer historical review participation feature (e.g., experience, familiarity, and participation rate) and use NSGA-II, which is a multi-objective meta-heuristic algorithm, to find the reviewer that has a high chance of participating in a review and maintaining the reviewing workload distribution. Chouchen et al. [5] propose the reviewer recommendation approach that adopts Indicator-Based Evolutionary Algorithm (IBEA), which is a genetic algorithm, to find the best set of code reviewers based on specific criteria: reviewer that has most experienced with the code change to be reviewed by considering changed file recency and frequency, and review collaboration social network; considering reviewers' current workload by using the size of code change, average time spend in files, and the number of open code reviews the reviewers are working on.

Finally, there has been methods that combine both heuristics and machine learning. Xia et al. [19] propose reviewer recommendation utilizing the combination of the similarity of file path and text mining model (e.g., naive Bayes) for textual contents in a review request. They use review data of Android Open Source Project. Yang et al. [21] develop a two-layer reviewer recommendation model to recommend reviewers for Pull-Requests in GitHub projects based on the technical and managerial perspectives. In the first layer, they recommend suitable developers to review the Pull-Requests based on a hybrid recommendation method (i.e., TF-IDF similarity of textual information, file location similarity, and expertise). For the second layer, they use the recommended developer from the first layer. They specify whether the target developer will technically or managerially participate using the Support Vector Machine classifier trained on the Expertise

and experience feature in the reviewing process.

IV. Our Proposed Approach

As shown in Figure 1, the system is comprised of two parts, the ChatBot interaction and the ChatBot Modeling Platform. For the ChatBot interaction, there is an assignee responsible for a reviewer assignment. The process is triggered when a new review request has arrived into the system. As shown in the figure, the ChatBot then should hold a conversation with the assignee to determine which is the appropriate recommendation algorithm to employ for the recommendation. These requirements could be captured from the conversation as features (e.g., the skills required for reviewing, the files/directories that are changed, etc.) from the assignee.

Based on the interaction, the ChatBot system should decide which features (cf. Table I) to use. Since each method will output a different set of reviewers, we use the ChatterBot ability to select the most suitable response to choose the sets of reviewers. The ChatBot will return a list of reviewers that the Assignee then can select from. For practicality, we assume that the ChatBot will then create a review request to a target reviewer, once the Assignee has approved the reviewer. In the case where the target reviewer has not agreed to accept the review request, the system will then inform the assignee to restart the process.

Figure 1 presents an overview of our proposed platform for our ChatBot. Our system is built on Django[2], which is one of a website development framework using Python. We develop our ChatBot system using ChatterBot[3], a Python framework. Currently, our implementation runs on a Ubuntu OS and using CPU @ 4.00GHz, with 500 GB of storage, and 32 GB of RAM. Our goal is to provide a more interactive experience, so that the many different features as shown in Table I can be inferred from a conversation.

V. Challenges and Conclusion

In this work, we take a first-step to explore the potential of using ChatBots are conversational and interactive systems for developers, instead of just recommendation tools. Probably the biggest challenge for the work is the evaluation of our approach in a real-world setting, where the developer feels like they are having a conversation with an agent to get the best possible solution. We believe that this solution is not easy and there are several challenges that need to be solved, before we can realize this potential. These are:

- *Evaluation of the ChatBot.* This may include a controlled user study, and also a large empirical study, requiring a large dataset and collection of developer conversations. We plan to conduct controlled user study, as well as explore data-mining techniques for evaluation.
- *Implementation of each reviewer recommendation algorithm.* As shown in the case study, recommendation systems may incorporate complex pipelines that require much computational time. Once our ChatBot platform is implemented, we plan to run experiments on the feasibility of running these techniques in real-time.
- *Efficient conversational context to infer code reviewer features.* Prior studies have shown that standard natural language may not be appropriate in the Software Engineering context. Furthermore, inferring the software development task from a natural language model is not trivial. Our idea is to use the conversations from Pull Requests and other code review artifacts to understand how developers communicate with each other.
- *Extensions of the ChatBot*. Future extensions include a communication process between assignee and the reviewer, to ask whether or not the reviewer can accept the review. Furthermore, we believe that different parts of the system (e.g. User Interface and ChatBots's model) could be implemented by separate microservices, allowing for robust and flexible usage for different purposes.

[2]https://www.djangoproject.com/

[3]https://chatterbot.readthedocs.io/en/stable/

Acknowledgment

This research is partially supported by the Japan Society for the Promotion of Science [Grants-in-Aid for Scientific Research (S) (No.20H05706)].

References

[1] A. Abdellatif, K. Badran, D. Costa, and E. Shihab. A comparison of natural language understanding platforms for chatbots in software engineering. *IEEE Transactions on Software Engineering*, (01):1–1, may 5555. ISSN 1939-3520. doi: 10.1109/TSE.2021.3078384.

[2] Wisam Haitham Abbood Al-Zubaidi, Patanamon Thongtanunam, Hoa Khanh Dam, Chakkrit Tantithamthavorn, and Aditya Ghose. Workload-aware reviewer recommendation using a multi-objective search-based approach. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2020, page 21–30, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Sumit Asthana, Rahul Kumar, Ranjita Bhagwan, Christian Bird, Chetan Bansal, Chandra Maddila, Sonu Mehta, and B. Ashok. Whodo: Automating reviewer suggestions at scale. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 937–945, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. Building an expert recommender chatbot. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 59–63, 2019.

[5] Moataz Chouchen, Ali Ouni, Mohamed Wiem Mkaouer, Raula Gaikovina Kula, and Katsuro Inoue. Whoreview: A multi-objective search-based approach for code reviewers recommendation in modern code review. *Applied Soft Computing*, 100:106908, 2021.

[6] Ferliana Dwitama and Andre Rusli. User stories collection via interactive chatbot to support requirements gathering. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18:870, 04 2020.

[7] Hamza Ed-Douibi, Gwendal Daniel, and Jordi Cabot. Openapi bot: A chatbot to help you understand rest apis. In Maria Bielikova, Tommi Mikkonen, and Cesare Pautasso, editors, *Web Engineering*, pages 538–542, Cham, 2020. Springer International Publishing.

[8] Ian Gauthier, maxime lamothe, Gunter Mussbacher, and Shane McIntosh. Is Historical Data an Appropriate Benchmark for Reviewer Recommendation Systems? A Case Study of the Gerrit Community. In *The 36th IEEE/ACM International Conference on Automated Software Engineering*, 2021.

[9] Xincheng He, Lei Xu, Xiangyu Zhang, Rui Hao, Yang Feng, and Baowen Xu. Pyart: Python api recommendation in real-time. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1634–1645, 2021.

[10] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering*, 26(5):93, Jul 2021.

[11] Bo Li, Qiang He, Feifei Chen, Xin Xia, Li Li, John Grundy, and Yun Yang. Embedding app-library graph for neural third party library recommendation. ESEC/FSE 2021, page 466–477, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Chun-Ting Lin, Shang-Pin Ma, and Yu-Wen Huang. Msabot: A chatbot framework for assisting in the development and operation of microservice-based systems. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 36–40, New York, NY, USA, 2020. Association for Computing Machinery.

[13] Dušan Okanović, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn, and Fabian Beck. Can a chatbot support software engineers with load testing? approach and experiences. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, ICPE '20, page 120–129, New York, NY, USA, 2020. Association for Computing Machinery.

[14] Mohammad Masudur Rahman, Chanchal K. Roy, and Jason A. Collins. Correct: Code reviewer recommendation in github based on cross-project and technology experience. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, page 222–231, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. *Recommendation Systems in Software Engineering*. Springer Publishing Company, Incorporated, 2014. ISBN 3642451349.

[16] Emre Sülün, Eray Tüzün, and Uğur Doğrusöz. Rstrace+: Reviewer suggestion using software artifact traceability graphs. *Information and Software Technology*, 130: 106455, 2021.

[17] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 141–150, 2015.

[18] Suppawong Tuarob, Noppadol Assavakamhaenghan, Waralee Tanaphantaruk, Ponlakit Suwanworaboon, Saeed-Ul Hassan, and Morakot Choetkiertikul. Automatic team recommendation for collaborative software development. *Empirical Software Engineering*, 26(4):64, May 2021.

[19] Xin Xia, David Lo, Xinyu Wang, and Xiaohu Yang. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 261–270, 2015.

[20] Zhenglin Xia, Hailong Sun, Jing Jiang, Xu Wang, and Xudong Liu. A hybrid approach to code reviewer recommendation with collaborative filtering. In *2017 6th International Workshop on Software Mining (SoftwareMining)*, 2017.

[21] Cheng Yang, Xun-hui Zhang, Ling-bin Zeng, Qiang Fan, Tao Wang, Yue Yu, Gang Yin, and Huai-min Wang. Revrec: A two-layer reviewer recommendation algorithm in pull-based development model. *Journal of Central South University*, 25(5):1129–1143, May 2018.

[22] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*, 42(6), 2016.

[23] Zhenyu Zhang, Hailong Sun, and Hongyu Zhang. Developer recommendation for topcoder through a meta-learning based policy model. *Empirical Software Engineering*, 25(1):859–889, Jan 2020.