Building an Expert Recommender Chatbot

Jhonny Cerezo[1], Juraj Kubelka[1], Romain Robbes[2], Alexandre Bergel[1]

[1] ISCLab, Department of Computer Science (DCC) University of Chile
[2] SwSE Research Group, Free University of Bozen-Bolzano, Italy

Abstract—**This paper presents our experience in implementing a chatbot for expert recommendation tasks. The chatbot was developed for the Pharo software ecosystem, and is integrated with the Discord chat service, which is used by the Pharo Community. We also report on a preliminary evaluation for which; the recommendation system was welcomed, though the conversational behavior was not; users expected a fully conversational chatbot, capable of following the conversation flow that the user handles. We discuss that such expectations might be hard to meet because of the uncanny valley effect.**

I. Introduction

Contributing to an open-source project is often considered as a social activity [6]. The quality of communication among developers may significantly impact their productivity. When developers encounter issues that they cannot solve by themselves, they often look for assistance. However, identifying the most adequate person to seek assistance is a challenging task since: (i) developers work remotely and may not know each other personally; (ii) nicknames instead of real names are used in chats; (iii) expertise is usually not publicly exposed.

In a distributed software development context, in which developers coordinate tasks using chat platforms, easing the coordination through chatbots may impact the way a community interact. Indeed, researchers predict that the conversational chatbot interfaces could revolutionize the field of Human-Computer Interaction (HCI) [10]. An expert recommendation chatbot could fill the communication gap in distributed software teams, utilizing information that is hard to access (*e.g.,* commit history, developer identity relations between source code repositories and chat platforms) in order to provide accurate expert recommendations.

We developed a chatbot as an expertise recommendation system to help developers find the right person to contact within open source projects. The chatbot targets the Pharo software ecosystem and developer community, and is integrated with the Discord chat service that the Pharo community uses as one of its main channels of communication. Our solution relies on simple Natural Language Processing techniques for the tasks of *sentence-classification* and *key-concept* identification, using the term frequency (TF) and inverse document frequency (IDF) algorithms respectively. We also implemented an expertise identification algorithm based on source code mining techniques. Finally, we conducted a preliminary evaluation, in which users interacted with the chatbot; the evaluation results point out a series of challenges that we plan to address as a future work.

II. Background

Expertise. Several heuristics have been proposed to identify software artifact experts. The earliest technique is known as the "line 10 rule", stemming from the fact that in some version control systems, the 10th line of a commit log contains the developer user name who performed the commit, *e.g.,* the last person who modified a code. This concept was refined into *implementation expertise*, where expertise on a piece of code is approximated to be the proportion of changes that a developer contributed to the piece of code. Mockus and Herbsleb used implementation expertise as part of their Expertise Browser [16]. Another heuristic relies on the fact that developers gain knowledge about source code artifacts using them. Schuler and Zimmerman call this *usage expertise* [21], and claim that it is a viable alternative for projects with little or no historical information. Ma *et al.* [13] compared both metrics and found them to have comparable levels of accuracy, allowing usage expertise to substitute implementation expertise. Anvik and Murphy used data from bug reports [5]; Robbes and Roethlisberger [19] used interaction data [14].

Software bots. Leboeuf developed a taxonomy of software bots [12]. Følstad and Brandtzæg argue that HCI may transition from graphical to conversational interfaces via chatbots [10]. They also conducted a survey of chatbot users, finding that the most common reason, by far, was productivity (68%) [8]. As productivity is a key concern in software engineering, investigating chatbots in this context seems is an attractive idea. Several bots have been integrated in chat platforms for developers [11, 22], such as bots helping with tests or DevOps activities, or even bots that integrate into Stack Overflow. Beyond chatbots, bots were developed to contribute to software projects [23], Stack Overflow [17], and Wikipedia [15]. Chatbots enable software people to engage in development operations without needing technical knowledge [11].

Pharo. Pharo [7] is an open source programming language, with a strong community concentrated in Discord (a chat platform service) [2] and a mailing list. The Pharo community encourages new collaborators to join Discord to ask questions, give feedback, and collaborate. Chatbot recommendation systems therefore represent a resourceful tool to increase the communication quality.

III. Implementing the Chatbot Expert Recommender

This section covers: the integration with the Discord API; key-concept identification, recognizing main phrases (software artifact names); sentence-classification; expertise mining techniques; and result representations to users.

A. Chatbot Implementation

A chatbot design depends on chat platform APIs. Discord provides a GameSDK (as Discord targets game communities), Webhook, Gateway (WebSocket), and REST API [1]. We developed the chatbot in Pharo using the DiscordSt library [3], which connects to Discord using the Gateway and REST APIs. The Discord chatbot deployment requires a token that can be obtained at a Discord App console.

B. Sentence Classification

The first step towards recognizing *user* conversational intentions is categorizing user messages. The chatbot classifies sentences in the following empirically defined message categories:

- **greeting**, includes greeting and introductory messages, *e.g.,* "How are you doing?"
- **package expert**, includes questions explicitly asking for package experts, *e.g.,* "Who is Iceberg package expert?"
- **class expert**, includes questions explicitly asking for class experts, *e.g.,* "Who is GLMAction class expert?"
- **method expert**, includes questions explicitly asking for method experts, *e.g.,* "Who is method Object yourself expert?"
- **artifact expert**, includes questions about a source code artifacts without explicitly defining if the artifact is a package, class, or method, *e.g.,* "Who is the TestCase expert?"
- **informative**, includes questions supported by the chatbot, not fitting into the above mentioned categories, *e.g.,* "What can I ask you for?"
- **unrecognized message**, includes user messages not supported by the chatbot, *e.g.,* "Ok"

Next, we describe the sentence classification algorithm that takes a user message and identifies a message category. The algorithm uses a TF-dataset. Figure 1 exemplify the TF-dataset with three *tf-sentences*: two for the artifact expert category (*artifact-exp* lines); and one for the package expert category (the *package-exp* line). A tf-sentence is thus a collection of terms (words).

To compute weights for all ft-sentences we use a raw term frequency algorithm (TF) that takes a user message word (term, t), a tf-sentence (s) and computes the number of term occurrences:

$$\text{TF}(t, s) = \textit{number of term } t \textit{ occurrences in } s$$

The tf-sentence weight (see column weight) for a given user message (T) is then computed as an average value of all user message word weight values:

$$w(T, s) = \frac{\sum_{\forall t \in T} \text{TF}(t, s)}{|T|}$$

We classify the user message into the category with the highest weight. In case the maximum weight is less than 0.2, the user message falls into the unrecognized message category. A threshold is manually defined and the most appropriate value is a matter of future research.

The described sentence classification algorithm get worse results with an increasing number of message categories. We therefore use a few categories in our study and proper understanding of best sentence classification is a subject for future work.

User message:	Who	is	an	expert	in	Roassal2	package	?

TF-Dataset

category	tf-sentence	weight
artifact-exp	expert in	0.25
artifact-exp	some one who knows about	0.125
package-exp	an expert for package	0.375

Note. artifact-exp = artifact expert category. package-exp = package expert category.

Figure 1. Sentence classification computation

C. Key-Concept Identification

The next step (after the sentence classification) is a *key-concept* identification. The chatbot identifies source code artifact names (key-concepts) in user messages, *e.g.,* in "Who is GLMAction class expert?", the key-concept is "GLMAction". It may be a method, class, and package name.

We use the inverse document frequency (IDF) algorithm to compute the key-concept. IDF is a numerical statistical method reflecting how important a term (word) is in a user message [18, 20]. The algorithm uses an IDF-dataset (D) that includes all questions that we extracted from the Pharo Discord chat channels. We semi-automatically collected questions. A detailed description of this dataset exceeds the paper size. The IDF mathematical formula is:

$$\text{IDF}(t, D) = \log \frac{|D|}{1 + \text{TF}(t, D)}$$

where t is a user message word, D is the IDF-dataset, and $\text{TF}(t, D)$ is term frequency algorithm computing number of term t occurrences in D. Figure 2 exemplifies the IDF computation. It shows the number of term occurrences for each user message word (TF) in the IDF-dataset and the IDF function values. Roassal2 has the highest IDF value and is identified as the key-concept, a source code artifact name.

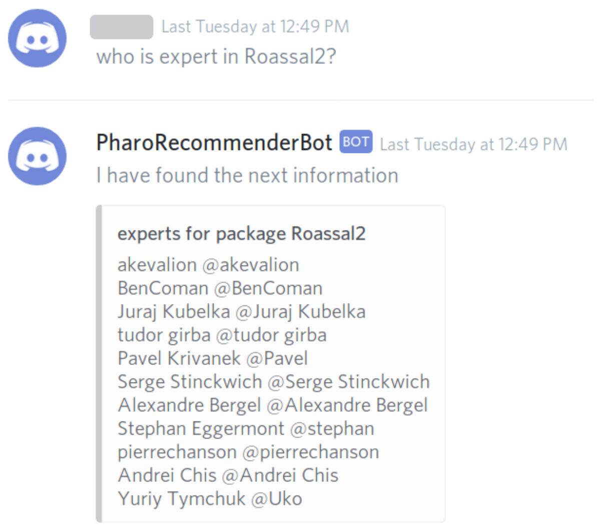
User message	Who	is	an	expert	in	Roassal2	?
TF(t, D)	78	1279	159	35	697	0	0
IDF(t, D)	1.706	0.496	1.399	2.047	0.760	3.604	0.0

Note. TF = term frequency. IDF = inverse document frequency.

Figure 2. Key-concept computation.

D. Expertise Recommendation System

Recommendation systems provide information to users according to their preferences. Expertise recommendation systems help developers to contact other developers, who can give them an information about what they seek for. This section describes how we collect the expertise information.



[redacted] Last Tuesday at 12:49 PM
who is expert in Roassal2?

PharoRecommenderBot BOT Last Tuesday at 12:49 PM
I have found the next information

experts for package Roassal2
akevalion @akevalion
BenComan @BenComan
Juraj Kubelka @Juraj Kubelka
tudor girba @tudor girba
Pavel Krivanek @Pavel
Serge Stinckwich @Serge Stinckwich
Alexandre Bergel @Alexandre Bergel
Stephan Eggermont @stephan
pierrechanson @pierrechanson
Andrei Chis @Andrei Chis
Yuriy Tymchuk @Uko

Figure 3. Chatbot recommendation system

1) Expertise mining: We mined expertise from historical Pharo source code changes, provided by the community in each release, and from several well supported community projects. The projects were selected considering Discord channels reserved to those projects. We use three techniques to mine expertise: *implementation* [5], *usage expertise* [13], and *alternative expertise*.

To mine implementation expertise from the Pharo project we collect methods authorship from Pharo historical releases and Pharo actual version. We included all Pharo releases from the last five years. All authors were treated equally (we do not apply any ranking).

We complement the expertise-dataset by including usage and alternative expertise. We compute the usage expertise by querying method references. If a method *a* calls a method *b*, then we define a method *b* author as the method *a* usage expert. Because of technical aspects, we had many methods that did not include any author (expert). We therefore compute *alternative expertise* considering an expert of an empty-author method, its class author.

2) Experts profile: The expert profile is composed of a source code expertise and a Discord nickname. In order to join the code expertise and the nickname we automatically associated Pharo contributors that used the same (or a similar) identification (name, nickname) in the source code and on Discord. We did manual source code expert and Discord account pairing in other cases, asking developers for help.

IV. Preliminary Evaluation

Our preliminary evaluation consisted of two different setups. In the first setup (Setup A), we asked three participants to interact with the chatbot without providing them specific tasks to achieve. The sessions were conducted remotely using Discord chat. In the second setup (Setup B), we asked three other participants to interact with the chatbot giving them specific tasks. We then conducted semi-structured interviews, and asked them to identify the emotions they felt while interacting with the chatbot.

A. Chatbot interaction

Study setup. We asked participants to converse with the chatbot. As this kind of interaction is rather informal and consulting a chatbot documentation (tutorial) is not an expected practice, we did not inform participants how to converse with the chatbot. Participant had to figure out how to formulate questions. Likewise, the chatbot did not provide any how-to-ask-me examples.

Study A participants were free to converse about anything. We asked Study B participants four tasks, each consisting finding experts for source code artifacts (package and class names). In total, we gathered 150 interactions of our 6 participants.

Successful interactions with the chatbot. Five out of the six participants were able to successfully interact with the chatbot and get expertise information from it. While some participants were successful very quickly (one in his second interaction, after greeting the chatbot), others had various unsuccessful interactions before they finally could complete tasks (one participant tried for more than 50 interactions before being able to find experts—occasionally showing frustration with the bot). The participant that did not complete the task did figure out how to formulate a query, but the queries made did not return results. After learning how the chatbot worked, half of the participants tried to discover whether the chatbot had additional functionality (see "Additional expectations" below), while the other half stuck to the patterns they knew.

Reasons for failure. There were several reasons for the failures; we mention the three most frequent. The first is that participants asked questions in a format the chatbot did not recognize (e.g., "Someone knows how to use GLMAction?"; "ask Iceberg"; "what experts do you know?"). The second was that the participant successfully formulated a query, but the format of the source code entity was incorrect (e.g., "Who is the Roassal expert?", when the actual entity was Roassal2; referencing to an entity in lowercase when it should be capitalized). The third one was that the participants asked correctly, but the expertise recommendation system missed information for some of the entities and was unable to find suitable experts in these cases.

Additional expectations. We also observed that some participants had additional expectations for the chatbot, beyond identifying experts. Participants asked about summaries of the capabilities of a source code entity ("what does GLMFormattedPresentation do?"), or directly asked questions meant for an expert to answer ("How may I capitalize a ByteString object?"; "How can I configure GLMAction?").

B. Semi-structured interview

The three participants we interviewed were asked for their previous experience with chatbots and any recommendation systems; their expectations about chatbot interactions and recommendation systems advices; whether the chatbot helps them in the task of finding an expert; and how they felt during the interaction with the chatbot, among other questions.

Overall, the interviews confirmed the observations gathered from the analysis of the chat interactions. Participants felt that the chatbot was too strict in which questions and source code entities it recognized, and missed some information. Participants mentioned that the chatbots should provide more assistance to the users, such as providing examples of questions to ask, and provide suggestions when a source code entity is misspelled. While some participants felt that the information that the chatbot gave them was useful, others felt that the chatbot could go further: returning a long list of experts does not help in choosing a specific one; nor in giving a summary of source code entities in addition to expert recommendations.

A theme that gained further importance in the interviews was the participants' expectations for chatbots. In particular, participants had strong expectations that the chatbot should be able to maintain conversations, not only answering queries. One participant expected interactions to be more human-like, while another raised the bar further, stressing that the chatbot should be designed to replace a person, and that users should not notice that they are interacting with a chatbot. Note that despite those expectations, many real-world chatbot implementations still maintain command-like interaction. Thus, participants pointed out that significant improvements needed to be made to the chatbot before it can be used in practice.

C. *Emotion test*

The *emotion test* in the study aims to reveal the subjective experience of participants interaction with the chatbot. Beyond the main purpose of the tool, emotion analysis allows us to see user's true perception of the chatbot. In this final part of the study the three participants were asked to choose 10 emotions, out of 114 *pleasant feelings* and 139 *unpleasant feelings* [4] thinking about the **chatbot interaction**. The feelings are aggregated into 8 positive and 8 negative categories to ease analysis. They then had to rank the list, starting from which emotion had been stronger through the interaction.

Overall, participants expressed **indifference** about the chatbot, likely due to them not using it in a real context. Participants were **confused** while learning how to use the chatbot, and also when it did not answer what the user expected. On the other hand, the best ranked emotion category was **openness**, reflecting that participants saw the potential of expert recommendation beyond the chatbot's current shortcomings.

V. Discussion

Although the recommendation system was well accepted, the lack of conversational behavior of the chatbot was not. As any communication flow must be carried out to a satisfactory conclusion, misleading feedback did not allow the information to flow successfully. Note that our study setup was driven to not provide any information on how the questions must be formulated. Our intention was to discover which interaction problems must be addressed. Participants emphasized several improvements: they need guidance from the chatbot to accomplish the task; the chatbot must answer a wider variety of questions; they also expected functionality that was not intended to be the purpose of the chatbot. Participants expected a fully conversational chatbot able to answer a variety of questions and follow the flow of a conversation that the user handles, leading to a **user-driven conversations**. We note these high expectations may be very hard to meet, particularly due to the "uncanny valley" effect, which states that artificial behavior that is close (but not close enough) to human behavior can cause more discomfort to humans than behavior that is less human-like. Recent work has observed the uncanny valley in chatbots [9].

In light of this, perhaps alternative strategies ought to be explored. For instance, a chatbot that focuses on single-purpose conversational interaction and guides the user to accomplish the specific task for which the chatbot has been built for (**bot-driven conversations**). Another possibility would be to have a **proactive chatbot** that analyzes questions asked on public channels and provides expert information when it is able to do so. This would avoid the need for humans to initiate the conversation with the chatbot so they do not have to learn how to interact with the chatbot. The chatbot would be silent when unable to answer a question, which would not be as much of an issue if users are not explicitly interacting with it.

VI. Conclusion and future work

The chatbot expertise recommendation system we presented increases the communication quality by giving open source developers the ability to know who they can contact when facing issues. Our chatbot relies on several components: the Discord API, TF and IDF algorithms to perform sentence-classification and key-concept collection respectively, and an expertise recommendation system based on *implementation* and *usage* expertise.

We conducted a preliminary study, where we analyzed their interactions, and also conducted interviews and an emotion test. The results show that while participants are open to the potential of an expert recommendation system based on a chatbot, significant work is necessary to increase its acceptance. In particular, participants expected the chatbot to be able to conduct a conversation with them, rather than simply answer queries.

Meeting these expectations may be a high bar; perhaps simpler models of interaction may encounter higher acceptance in the future. As future work we will extract information from chat-logs (or other sources) to provide recommendation about already answered questions in the communication history of an open source project.

Acknowledgments We thank Renato Cerro for his English revisions, and three anonymous reviewers for their insights. Cerezo thanks ObjectProfile for sponsoring this work. Kubelka is supported by a Ph.D. scholarship from CONICYT, Chile. CONICYT-PCHA/Doctorado Nacional/2013-63130188. Bergel thanks Lam Research for sponsoring part of this effort.

References

[1] Discord API documentation. https://discordapp.com/developers/docs/intro.

[2] Discord Website. https://discordapp.com.

[3] DiscordSt project, Discord Interfaces for Pharo. https://github.com/JurajKubelka/DiscordSt.

[4] Emotion list. http://www.psychpage.com/learning/library/assess/feelings.html.

[5] J. Anvik and G. C. Murphy. Determining implementation expertise from bug reports. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 2. IEEE Computer Society, 2007.

[6] M. Bergquist and J. Ljungberg. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 11(4):305–320, 2001.

[7] A. Black, S. Ducasse, O. Nierstrasz, D. Pollet, D. Cassou, and M. Denker. *Pharo by Example*. Square Bracket Associates, 2009.

[8] P. B. Brandtzaeg and A. Følstad. Why people use chatbots. In *International Conference on Internet Science*, pages 377–392. Springer, 2017.

[9] L. Ciechanowski, A. Przegalinska, M. Magnuski, and P. Gloor. In the shades of the uncanny valley: An experimental study of human–chatbot interaction. *Future Generation Computer Systems*, 92:539–548, 2019.

[10] A. Følstad and P. B. Brandtzæg. Chatbots and the new world of hci. *interactions*, 24(4):38–42, 2017.

[11] C. Lebeuf, M.-A. Storey, and A. Zagalsky. Software bots. *IEEE Software*, 35(1):18–23, 2018.

[12] C. R. Lebeuf. *A taxonomy of software bots: towards a deeper understanding of software bot characteristics*. PhD thesis, 2018.

[13] D. Ma, D. Schuler, T. Zimmermann, and J. Sillito. Expert recommendation with usage expertise. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 535–538. IEEE, 2009.

[14] W. Maalej, T. Fritz, and R. Robbes. Collecting and processing interaction data for recommendation systems. In *Recommendation Systems in Software Engineering*, pages 173–197. Springer, 2014.

[15] L. McLaughlin. Bot software spreads, causes new worries. *IEEE Distributed Systems Online*, (6):1, 2004.

[16] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 503–512. IEEE, 2002.

[17] A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu. Among the machines: Human-bot interaction on social q&a websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1272–1279. ACM, 2016.

[18] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142, 2003.

[19] R. Robbes and D. Röthlisberger. Using developer interaction data to compare expertise metrics. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 297–300. IEEE Press, 2013.

[20] S. Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.

[21] D. Schuler and T. Zimmermann. Mining usage expertise from version archives. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 121–124. ACM, 2008.

[22] M.-A. Storey and A. Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931. ACM, 2016.

[23] S. Urli, Z. Yu, L. Seinturier, and M. Monperrus. How to design a program repair bot?: insights from the repairnator project. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 95–104. ACM, 2018.