Building a Bot for Automatic Expert Retrieval on Discord

Ignacio Nuñez Norambuena
ISCLab, Department of Computer Science (DCC),
University of Chile
inunezn@fen.uchile.cl

Alexandre Bergel
ISCLab, Department of Computer Science (DCC),
University of Chile
abergel@dcc.uchile.cl

ABSTRACT

It is common for software practitioners to look for experts on online chat platforms, such as Discord. However, finding them is a complex activity that requires a deep knowledge of the open source community. As a consequence, newcomers and casual participants may not be able to adequately find experts willing to discuss a particular topic.

Our paper describes a bot that provides a ranked list of Discord users that are experts in a particular set of topics. Our bot uses simple heuristics to model expertise, such as a word occurrence table and word embeddings. Our bot shows that at least half of the retrieved users are indeed experts.

CCS CONCEPTS

• **Information systems** → **Expert search**; • **Computing methodologies** → Information extraction; • **Software and its engineering** → *Open source model.*

KEYWORDS

Bot, Discord, Software, Word Embeddings, Expert Retrieval Systems.

ACM Reference Format:
Ignacio Nuñez Norambuena and Alexandre Bergel. 2021. Building a Bot for Automatic Expert Retrieval on Discord. In *Proceedings of the 5th International Workshop on Machine Learning Techniques for Software Quality Evolution (MaLTESQuE '21), August 23, 2021, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3472674.3473982

1 INTRODUCTION

Identifying expertise within the members of an organization is important and can be crucial when building an agile team in software development [11]. This is especially true in presence of multiple online and continuous communication channels.

One popular way to get in contact with experts is to use an online chat platform. Discord is a digital platform structured in terms of communities, typically called "servers", created by users. Members of these communities can interact through text, voice, and video channels. Discord is now highly popular and it has more than 150 million monthly active users [5].

Discord does not naturally propose or enforce a structure in the way interaction happens among users. Discord users can write and share documents in an unconstrained fashion. To use some particular actions, such as information finding, bots are becoming very popular [7].

Expert retrieval. Finding experts in online open-source communities is beneficial to guide new members and indicate them who is the right person to ask questions or discuss a particular topic [3]. This effort is particularly relevant in the presence of a large volume of communications.

One of the main characteristics of expert retrieval systems is to analyze a large dataset and retrieve a rank of people sorted by their level of expertise, itself based on arbitrary criteria. Our effort can be classified as *expert finding in social networks* [1] since our data is made of messages collected from an online platform.

Expert retrieval is a broad area in which two types of retrieval systems may be distinguished: *expert finding* systems and *expert profiling* systems. The first one is about searching for a person with some level of expertise for a given topic, while the second one is about retrieving a list of topics in which a person is considered an expert [1]. Our work focuses on expert finding, but because of the close relation between these two kinds of systems, we expect our result to be also useful for expert profiling.

Word embedding. It is often difficult to relate questions raised by users to the profiles of other people. To tackle this problem word embeddings are a valuable resource to associate terms according to its meaning since words are modeled as a vector in a high-dimensional space and it is possible to calculate how close or far away they are on it.

The advantage of word embedding is that it is self-supervised, as such, there is no need to have tagged data to train models and it learns the word meanings from their distributions in textual documents [6].

ExpertFinder. We have designed a bot called ExpertFinder for Discord servers that helps users to identify experts for a particular topic. The notion of expertise is based on the frequency of some words in affirmative sentence. For example, if a user is using the word `seaside` (a popular web framework library) in several sentences that are not questions (i.e., it does not contain a question mark), then the user is assumed to be an expert in `seaside`.

Any users can query ExpertFinder with a given list of words – supposedly relevant topics of interest. ExpertFinder privately replies to the user with the list of experts according to the number of occurrences each expert mentioned.

We have conducted a preliminary evaluation of ExpertFinder, in particular to evaluate the simple heuristics we have employed.

Paper outline. This paper is structured as follows. Section 2 gives a brief overview of the work related to this paper. Section 3 explains how the bot was implemented. Section 4 talks about the methodology followed to assess the results of the bot. The models defined for expertise retrieval are shown in Section 5. Results are analyzed in Section 6. The limitations of the approaches used in this work are discussed in Section 7. Finally, Section 8 presents our conclusions and future work.

2 RELATED WORK

Several efforts have been made to develop expert retrieval systems related to software mainly based on Communities of Question Answering (CQA) websites. For instance, Gharebagh et al. [4] uses data from the popular website StackOverflow and makes an expertise classification for each topic according to the accepted answers users give to questions related to it.

On a different approach, Liu et al. [8] uses data from *Sun forums (now Oracle forums)* and *Apple Discussions* to create expertise profiles based on a complex three-stage framework.

An important aspect highlighted in Niemann [9] is the duality of expertise: the candidate has to be an expert and be considered one by her/his community. To address this duality he proposes a model that classifies the posts from CQA forums into dialogue acts to determine the expertise of people based on the kind of interaction that users have and term usage.

Canfora et al. [2] proposed a technique to find mentors from analyzing a mailing list of a software project. Their work highlights the importance of mentors having "good instruct attitude" which sometimes is not held by the people who know the most about a topic. ExpertFinder Bot also captures this characteristic because counting term frequency is also a measure of the number of interactions a person has had, which in most cases consists of resolving questions to other members of the community. However, on one hand, the bot is an effort to take advantage of new platforms like Discord that have became frequently used for software development and, in the other hand, the bot is an attempt to create a tool as simple as it can be by not depending on any other source and applying a simple algorithm to retrieve experts.

A similar work by Cerezo et al. [3] builds a discord bot for expert retrieval based on sentence classification using a term frequency-inverse document frequency technique. It also highlights the importance of considering the "uncanny valley" effect when thinking about the interaction between the bot and the user.

The last decade has resulted in tremendous progresses in Natural Language Processing (NLP) and in many applications. Rampisela et al. [10] make an attempt to classify expertise in an academia context by using word embeddings in a very similar way to what is done in this work. According to the authors, this method has a performance comparable to standard expert retrieval models.

Despite the fact expert finding systems is a well studied problem, new platforms like Discord have very different characteristics. This work proposes an easy-to-implement alternative to take advantage of the information present in online communities that can be useful to identify expertise.

3 BOT FOR EXPERT RETRIEVAL

We have built *ExpertFinder*, a bot for Discord to help a user finding some experts for a given set of topics. ExpertFinder is implemented in Python, using the `discord.py` API. This section highlights some of the decisions we have taken when designing ExpertFinder.

Data acquisition. When deployed, ExpertFinder scans the Discord server it is installed on and fetches all historical messages, and so, for all the channels of the server. Fetching historical messages can also be carried out in an incremental fashion, up to the last message recorded by the previous fetch.

Each fetched message has information about (i) the content (ii) the Discord ID of the author (iii) the channel ID where the message was sent through and (iv) the date and time the message was sent. From this information it is also possible to determine the Discord username and nickname of the author.

Preprocessing. After the data of the server is collected, our bot cleans the data by removing stop words and URLs since these components of messages do not directly contribute to the task of expert finding.

The goal of our bot is to identify potential experts for a number of keywords. Our preprocessing cleaning phase removes questions since it is unlikely that a user asking a question can be considered an expert. Users who answer this question are more likely to have a greater level of expertise than the asking person. As such, we clean out questions since focusing on affirmative statements are more likely to contribute to increasing a level of expertise in our model.

Our bot removes content of a message that precedes a question mark. After a question mark, the remaining message content is used to retrieve expertise retrieval. For instance if someone message was *"I don't get what this method does. Can anyone help me? I couldn't find it in the documentation of that library"*, our bot just considers the part after the question mark (*"I couldn't find it in the documentation of that library"*).

Retrieval. Once the chat content is cleaned, our model considers each word as a potential topic of expertise. A pair *(word, occurence)* is associated to its author in our expertise model. Our model is a large map that associates authors to a list of words associated to their frequencies. Roughly, our model knows what the words are and their occurrences mentioned by each discord users. We filter out words that are very seldom-used, i.e., our model ignores words that arbitrarily appear less then three times in the whole history.

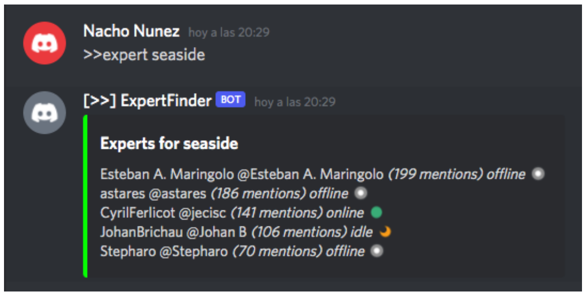
Moreover, they are ranked according to the number of times each person used the concept on the history of messages in an affirmative sentence. Some variations of this approach are developed and are explained in Section 5.

Interaction design. There are essentially two ways for Discord users to interact with ExpertFinder: either by sending a private message to it (the very same way as sending a private message to any other user), or by sending a public message in a channel. In both cases, the instructions must be preceded with a particular prefix to let the bot identify messages for it.

A query is simply formulated by a Discord user as `»expert [topics]`, where `[topics]` is a place holder for one or more topics, i.e., words.

Once a valid query is made by a user the bot processes it and retrieves the top-five users having the most mentions of the topic. The number of results shown by the bot is limited to a maximum of 5, even when the complete list of people could be larger. It was set to that number because it allows one to have several options to contact to and it is not too large to be visually and practically inappropriate for a text conversation.

The rank retrieved by the bot includes the Discord nickname, username, number of mentions of the concept and the status of the user at the moment the query was made (offline, idle or online). There is an example of this in Figure 1.


Nacho Nunez hoy a las 20:29
>>expert seaside

[>>] ExpertFinder BOT hoy a las 20:29
Experts for seaside
Esteban A. Maringolo @Esteban A. Maringolo *(199 mentions) offline* ●
astares @astares *(186 mentions) offline* ●
CyrilFerlicot @jecisc *(141 mentions) online* ●
JohanBrichau @Johan B *(106 mentions) idle* ●
Stepharo @Stepharo *(70 mentions) offline* ●

Figure 1: Example query and reply from ExpertFinder bot

ExpertFinder privately replies to the user who made a query in a public channel. The goal is to avoid publicly sending automatically-generated messages, which could be negatively perceived by the community.

4 METHODOLOGY TO EVALUATE OUR EXPERTFINDER BOT

As far as we are aware of, it is not common for Discord or Slack communities to have bots that help identify experts. As such, no methodologies have been proposed, as far as we know, to evaluate the performance of a such a bot.

4.1 Pharo Discord server

We selected the Pharo Discord server to deploy ExpertFinder, mostly because of the familiarity of the authors with this community. The Pharo Discord server has 2,790 members and on average more than 100 messages have been sent since September 8, 2016. The Pharo Discord server consists of 55 text channels that are grouped in categories like *New Users* (mostly for beginners), *Pharo* (general information and features about the Pharo programming language), *Libraries* (for libraries-specific information), *International* (for non-English conversations) and *Intermittent* (for event such as sprint, job advertisement, and discussion about Google summer of code).

The Pharo community does not use bots on its Discord server. As such, the community is probably unfamiliar with bot interacting.

4.2 Workflow

Step 1 - First release. An initial release of the bot is deployed in the Pharo Discord server accompanied with a brief advertisement made by the authors of this paper. After two weeks, 14 people had in total 84 interactions with the bot. From them, 61 (73%) were valid expert-finding queries and 39 were related to Pharo programming language. Just 8 of them were queries with more than one word.

Help on how to interact with the bot is offered to Discord users who initiate a conversation with ExpertFinder.

Step 2 - Benchmark elaboration.
In order to have a manageable number of queries for making the evaluation, it was decided to choose just 10 of the 1-word queries from the ones correctly formulated. There were preferred concepts about different libraries across Pharo and frequently used along the message history of this Discord community.

Meanwhile, all the 8 multiple-words queries were taking into account. In Table 1 all the queries considered are displayed.

Table 1: Selected queries by category.

Category	Selected Queries
1 - word	Bloc, Calypso, Iceberg, Roassal, Seaside, Spec, Spec2, VM, VR, Woden.
multiple - words	Artificial Intelligence, Beta Arm64 VM, Dynabook Smalltalk, Dynamic Library Windows, Glamorous Toolkit, State Machine, Dolphin Smalltalk, Headless VM.

Step 3 - Bot evaluation - Survey to experts. To assess the performance of the heuristics we employed in ExpertFinder to identify experts, we directly surveyed the 5 most ranked users about their level of expertise for each of the 18 queries (Section 4.3).

Step 4 - Word embeddings. In order to improve the precision of the results, a word embedding model was trained from scratch using the complete message history of Pharo Server. It contains more than 180,000 messages. The purpose of using a model based on word embeddings is to exploit the cosine similarity of vectors to identify topics that are considered as close.

4.3 Who is an expert?

One of the main difficulties we faced when building our expert retrieval system was in determining whether the people listed by the bot were indeed experts in the asked topic or not. This challenge arises from the lack of structured and multiple source of informations about positions and expertise areas of the member of an open-source community.

As a consequence, to evaluate the performance of our expert finding system to designate a user as an expert, we rely on a self-reported expertise of the user through two short and simple questions:

(1) *Do you consider that you have some level of expertise in [topic]?*
(2) *Do you think you are capable of contributing to solve some problems related to [topic]?*

These questions were asked through a private message by showing one button for "yes" (a check mark icon) and one for "no" (an "X" icon). The aim of doing this was to induce asked people to give a binary answer.

In the remainder of this paper, users who answered affirmatively both questions will be considered experts.

4.4 Measures

Following Balog et al. [1], the quality of the rankings given by the expert finding system depends on the capability for our model to list relevant experts for a given arbitrary topic. For that reason the recommended metrics to use in these cases are precision metrics like Mean Average Precision, Precision@N, and Mean Reciprocal Rank. Nevertheless, the first of those metrics would require knowing who are all the experts on each query topic which is hard to achieve in this context. Therefore it does not make sense to use it for truncated lists of results.

Due to the format of the answer of the bot – this is, a short list where the maximum number of results is 5 at most – it is more suitable to have a precision measurement just over the results the user received (i.e., the top 5 most ranked users). Therefore we introduce the Query Precision (QP):

$$QP = \frac{\text{Number of correct results on the list}}{\text{Length of the answers list}} \quad (1)$$

This definition is very close to the Precision@N measure, which refers to the number of correct answers over the first N results, where N is arbitrary. To some extent, QP is similar to Precision@N with a value of N equal to the length of the list. In this case N is a variable because it could be less than five in cases where the bot is not capable of finding that number of users considered experts.

Another measure that is convenient for evaluating the bot is the mean reciprocal rank, which corresponds to the inverse of the first correct answer on the list:

$$RR = \frac{1}{\text{Position of the first correct answer}} \quad (2)$$

For instance, consider that for a particular query ExpertFinder identifies two users as supposedly being experts, in which the first-ranked user *is not* an expert and the second-ranked *is* an expert. The reciprocal rank would then be $1/2 = 0.5$. This also implies that in the cases where the top ranked result is correctly identified as an expert, the value of RR is 1.

Note we do not consider metrics to measure recall since there is no clear way on how to compute it in our scenario.

4.5 Surveying users

Our third step of the methodology is about surveying Discord users reported as our expertise models for the 18 queries. Some of the users did not answer the survey we sent them, and as such, we associate a range for each of the two metrics (Query Precision and Reciprocal Rank). We determine:

- the *lower bound*, which represents the worst scenario where it is assumed that users who did not answer the survey are not experts,
- the *upper bound*, which represents the most favorable cases where non-respondents are experts.

For instance if for some query two persons did not declare their expertise, two of them said they are experts and one answer negatively, the QP range will be $0.4 - 0.8$, because in the worst case there were just two experts out of five and in the best scenario there were four experts out of five in the list retrieved by the bot.

The Mean Query Precision (MQP) and Mean Reciprocal Rank (MRR) range correspond to the average of the lower bound and the average of the upper bound for each measure across all the queries. These are the two metrics we consider to assess the performance of ExpertFinder for each of the results of the benchmark.

Finally, we refer as the *ground truth* to the answers of the surveyed users. We will use it to compare the different expertise models we present in Section 5 by calculating the metrics introduced above.

4.6 Word Embeddings

We employ word embeddings to exploit the similarity between words / topics. The model we trained has 200 dimensions and was built using Python and the Gensim library. A direct application of the Gensim library produces a model that associates words that are close to each other using the cosine similarity measure. An example of this is shown in Table 2, which lists all the words related to the word "database" from a model trained using the history of the Pharo Discord server.

For the purpose of this work, using word embeddings enables us to consider related concepts when deciding whether a user is expert or not. It appears to be very adequate technique to tackle this problem since (i) the query input is just one concept but it could have left out variations of it – e.g. shortening terms like "db" and plural variations like "databases" or "dbs" – and (ii) it makes sense that experts know and talk about close concepts and terms. For instance, experts in databases probably dominate systems like NoSQL and Postgres server.

Table 2: Most similar concepts to "database"

Concept	Cosine Similarity
'databases'	0.74
'db'	0.73
'relational'	0.71
'nosql'	0.71
'postgres'	0.71
'migrations'	0.7
'dbs'	0.69

Another important feature is the aggregation of vectors into one. It is especially relevant for queries containing more than one word as input. Basic operations like the average generally work fine to make a good representation of a multiple-word composed term.

In terms of efficiency, despite the fact using word embeddings is equivalent in time to make several simple queries the bot still replies to the user almost immediately.

5 EXPERTISE MODELS

A number of expertise models can be defined, considering the granularity of the occurrences and multiple topics. Queries with more than one word as input are more complex compared to single-word inputs. To address this difference the models used to determine the expertise of the users can be divided into two categories.

5.1 1-word models

- **Mentions:** This model simply counts the number of times users mention the word along their message history.
- **Messages:** It counts the number of messages the user has sent that includes the word at least one time.

- **Messages using word embeddings:** It does the same as the previous model, but also counts the number of messages users sent that include *similar* words according to the embeddings. Similar words are the first 10 words with a cosine similarity greater than 0.6.

5.2 Multiple-words models

- **History:** It calculates the minimum between the mentions of each word in the message history of the user.
- **Same Message:** It counts the number of messages sent by the user that contain every word in the query.
- **Consecutive:** It counts the number of messages sent by the user where the words are consecutive and in the order they were typed in the query.
- **SameMessageWE:** Firstly, one vector is calculated to represent the whole query concept by simply averaging the vector representation of each individual word. Secondly, it counts the number of messages for each of the first ten words with a cosine similarity greater than 0.6 and the word given in the query. Finally, the ranking list is the sum – by user – of the counters previously calculated.

6 RESULTS AND ANALYSIS

We run the expertise models given above and compare them against the ground truth made of the answers from the surveyed Discord users. Table 3 gives the results of the two metrics listed in Section 4.5. We add the Av.N metric as the average number of experts returned by the expertise model.

Table 3: Average value of measures across queries for different models

	Model	MQP	MRR	Av.N
1-word	Mention	0.5 - 0.7	0.85 - 0.9	5
	Message	0.5 - 0.7	0.85 - 0.9	5
	Message WE	0.58 - 0.78	0.85 - 0.9	5
multiple -words	History	0.46 - 0.66	0.73 - 0.79	3.8
	Same Message	0.55 - 0.75	0.88 - 1	2.5
	Consecutive	0.52 - 0.74	0.58 - 0.92	1.9
	Same Msg WE	0.30 - 0.66	0.47 - 0.84	4.8

Firstly, it is worth noting that most of the models have a MQP greater than 0.5 in the worst-case scenario. This means that on average at least half of the users returned by ExpertFinder are actually experts.

Secondly, in most of the models tried the top-ranked users are indeed experts, which is observable with the high values of the MRR column of the table. This indicates that a user that talks the most about a certain topic is usually an expert for that topic. Based on this, it is possible to deduce that a way to achieve a greater precision is to take just the first-ranked users returned by the bot.

The best 1-word models are the message counter with word embeddings to take into account related words. The higher level of the lower bound – compared to the other models – represents some false positives that appear in the rest of one-word specifications are not considered when using word embeddings.

A different scenario may be observed when considering multiple-words queries because the word embeddings approach seems to be less accurate than the other models for this type of queries. This could be explained by the greater number of results on average it retrieves compared to the rest of the multiple-words models. Because word embeddings consider a group of words, the list of experts is naturally larger and can also lead to false positives that are larger. Presumably, this problem is not present in 1-word models due to the answers of the bot being large enough even when word embeddings are not used.

Looking up the queries one by one it is possible to see that it retrieves experts not considered by the other models, but paying the price of more false positive cases.

7 DISCUSSION AND LIMITATIONS

Experts binary classification. The ranking created by the ExpertFinder bot works better when it is applied to topics where the number of times the concept appears is high. But the results are not very accurate when that number is low enough. This seems that has to do with the lack of a threshold that defines whether someone is an expert or not. For instance, take the case where the list is just made up by one name and the counter number is low. Is this an expert or not? The ranking has proven to be a good relative measure, but its weakness is in terms of absolute results.

Therefore the list is more like a *relative-knowledge* ranking and when the list is large enough the probability of the top-5 users are indeed experts increase. That could explain the good results in the MRR measure, where most times, those on top of the list are experts. This could also lead to a high self-perception of expertise just because someone is the person who *relatively* know more about the topic in the community.

Nevertheless the aim of this work is to give a reference to members who can help and give guidance about a topic, and sometimes it is not necessary to talk to a senior expert to have that. There are also some mechanisms to avoid this, for example, showing a confidence-level measure about the result has been delivered.

Message quantity bias. The algorithms used by the bot tend to be sensitive to community members who send relatively more messages. Moreover, word embedding methods seem to increase the bias, since they consider a wide range of terms and therefore even more messages.

On one hand, it is an advantage that the results include people who interact more, because it is more likely they reply to a question if they have some level of expertise in the topic. On the other hand it may lead to wrong results.

Self-reported expertise. Expert status could have a different interpretation among the candidates asked. A different and more objective measure could be achieved. An example of this is the number of years the user has been working on the topic or whether it is used or developed in his main activity. The disadvantage of this approach is that more complex questions can lead to less participation on the survey.

8 CONCLUSION AND FUTURE WORK

In this preliminary work, we have found two results. First, our technique to automatically identify experts has a high precision specially in the top-ranked positions. Second, word embeddings seem to have a positive impact to find experts but it also leads to a higher number of false positives.

The future work concerning to the evaluation has three main dimensions:

i. Extensive dimension, which is referred to encourage more people to use the bot and have more queries to evaluate their precision – as the bot does not need anything more than access to Discord messages, it is quite simple to use it in other software development communities.
ii. Intensive dimension, which has to do with being able to identify the level of expertise of the people retrieved by the bot – e.g., using an Likert-type scale – that can give us more information than the binary classification used in this work.
iii. The user experience dimension, by focusing on the ease of use and how useful are the results retrieved by the bot from the users point of view.

Regarding to the methods used, a possible path is to consider classifying messages according to the act of speech or try to identify some interaction patterns between people in order to distinguish expert-to-expert conversations from expert-to-non expert or non experts dialogues.

Finally, the word embedding approach showed improvement in the performance of the expert finding system for some queries, but gives more false positives in others. We believe the application of this technique to this kind of problems could be enhanced in order to keep just the positive side of it.

ACKNOWLEDGMENTS

We thank Lam Research and the ANID FONDECYT Regular 1200067 for partially sponsoring the work presented in this paper.

REFERENCES

[1] Krisztian Balog, Yi Fang, Maarten de Rijke, Pavel Serdyukov, and Luo Si. 2012. Expertise Retrieval. *Found. Trends Inf. Retr.* 6, 2-3 (Feb. 2012), 127–256. https://doi.org/10.1561/1500000024
[2] Gerardo Canfora, Massimiliano Di Penta, Stefano Giannantonio, Rocco Oliveto, and Sebastiano Panichella. 2013. YODA: Young and newcOmer Developer Assistant. In *2013 35th International Conference on Software Engineering (ICSE)*. 1331–1334. https://doi.org/10.1109/ICSE.2013.6606710
[3] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2019. Building an Expert Recommender Chatbot. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 59–63. https://doi.org/10.1109/BotSE.2019.00022
[4] Sajad Sotudeh Gharebagh, Peyman Rostami, and Mahmood Neshati. 2018. T-Shaped Mining: A Novel Approach to Talent Finding for Agile Software Teams. In *Advances in Information Retrieval*, Gabriella Pasi, Benjamin Piwowarski, Leif Azzopardi, and Allan Hanbury (Eds.). Springer International Publishing, Cham, 411–423.
[5] Discord Inc. 2021. *About us*. https://www.discord.com/company
[6] Daniel Jurafsky and James H. Martin. 2020. *Speech and Language Processing (3nd Edition Draft)*. https://web.stanford.edu/~jurafsky/slp3/
[7] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. 2018. Software Bots. *IEEE Software* 35, 1 (2018), 18–23. https://doi.org/10.1109/MS.2017.4541027
[8] Xiaomo Liu, G. Wang, Aditya Johri, Mi Zhou, and Weiguo Fan. 2012. Harnessing global expertise: A comparative study of expertise profiling methods for online communities. *Information Systems Frontiers* 16 (09 2012).
[9] Michael Niemann. 2014. Finding expertise using online community dialogue and the Duality of Expertise. In *Proceedings of the Australasian Language Technology Association Workshop 2014*. Melbourne, Australia, 69–78. https://www.aclweb.org/anthology/U14-1009
[10] Theresia V. Rampisela and Evi Yulianti. 2020. Academic Expert Finding in Indonesia using Word Embedding and Document Embedding: A Case Study of Fasilkom UI. In *2020 8th International Conference on Information and Communication Technology (ICoICT)*. 1–6. https://doi.org/10.1109/ICoICT49345.2020.9166249
[11] Peyman Rostami and Mahmood Neshati. 2019. T-shaped grouping: Expert finding models to agile software teams retrieval. *Expert Systems with Applications* 118 (2019), 231–245. https://doi.org/10.1016/j.eswa.2018.10.015