Experiences Building an Answer Bot for Gitter

Ricardo Romero
rromero@cs.fsu.edu
Florida State University
Tallahassee, Florida

Esteban Parra
parrarod@cs.fsu.edu
Florida State University
Tallahassee, Florida

Sonia Haiduc
shaiduc@cs.fsu.edu
Florida State University
Tallahassee, Florida

Abstract

Software developers use modern chat platforms to communicate about the status of a project and to coordinate development and release efforts, among other things. Developers also use chat platforms to ask technical questions to other developers. While some questions are project-specific and require an experienced developer familiar with the system to answer, many questions are rather general and may have been already answered by other developers on platforms such as the Q&A site StackOverflow.

In this paper, we present GitterAns, a bot that can automatically detect when a developer asks a technical question in a chat and leverages the information present in Q&A forums to provide the developer with possible answers to their question. The results of a preliminary study indicate promising results, with GitterAns achieving an accuracy of 0.78 in identifying technical questions.

CCS Concepts

• **Software and its engineering → Collaboration in software development**; **Documentation**;

Keywords

communication, chat, social media, team communication platforms, bot, Q&A, recommendation

ACM Reference Format:
Ricardo Romero, Esteban Parra, and Sonia Haiduc. 2020. Experiences Building an Answer Bot for Gitter. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387940.3391505

1 Introduction

Developing complex software systems requires large teams of developers to collaborate, communicate, and coordinate their efforts. Recently, modern messaging and collaboration platforms such as Gitter[1] and Slack[2] have revolutionized team communications and project coordination by providing a user-friendly way of managing and organizing conversations, facilitating knowledge sharing, and by integrating with external software development tools such as GitHub, Travis CI, and Jira[11, 14, 23, 25]. Developers are nowadays preferring these messaging platforms, which allow them to receive real-time responses from other developers, as opposed to more traditional, asynchronous communication like mailing lists [25].

Developers often use modern messaging platforms to ask technical questions to other developers[10]. A recent study of the messages exchanged in the instant messaging platform used by one large software development company found that about half of the messages were related to problem-solving (*i.e.*, questions and answers) [25]. Although some of these questions can be very specific and require a wealth of experience and knowledge of a system to answer, other questions, which are generally asked by beginners, may have been already answered on other platforms, such as Q&A forums. In Q&A forums like StackOverflow, developers interact by posting questions and answers related to different programming languages, technologies, and software development topics [3, 16]. Therefore, given that there are already millions of technical questions answered on StackOverflow, there are high chances that at least some of the troubleshooting questions asked by developers on chat platforms have already been answered on StackOverflow.

In this paper, we introduce **GitterAns**, a bot that automatically detects when a troubleshooting question is asked in an online Gitter chat and then provides the user with possible answers, based on querying StackOverflow for posts similar to the question. Automatically answering these questions could lead to a decrease in the response time, as well as the effort that developers in an online community have to put into answering these questions. A preliminary evaluation shows that GitterAns is currently able to detect troubleshooting questions with 78% accuracy. When answering the questions, however, we found that, in its current implementation, GitterAns is able to find the correct answers only in about half of the cases. Our future work will focus on improving both the question identification and question answering components of GitterAns, as well as on performing a large-scale evaluation.

2 The GitterAns Framework

In this section, we present an overview of GitterAns and its main components, shown in Figure 1. GitterAns has three main parts: question detection, searching for answers on StackOverflow, and answer processing.

2.1 Question Detection

To detect a troubleshooting question, GitterAns performs the following steps for any incoming message to the chat room: 1) read the incoming message, 2) pass the message through a preprocessing and feature extraction procedure (described below) and 3) based on the extracted features, predict whether the message is a troubleshooting question or not using a machine learning classifier.

[1] https://gitter.im/
[2] https://slack.com/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3391505

If the message is determined to be a troubleshooting question, it is then passed on to the *Search for Answers* step. Otherwise, the message is ignored.

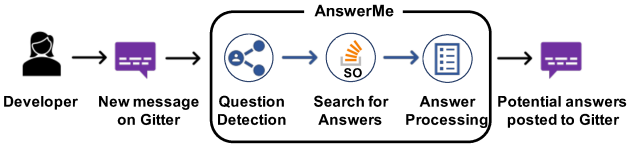

Figure 1: Overview of GitterAns

2.1.1 *Read Incoming Messages*

Using a Gitter room's URI, (*e.g.*, nodejs/node), a curl request using pycurl[3] is performed to obtain the room's unique ID. Once the room ID is obtained, a message stream to obtain the messages posted is initialized using the Python Requests library and the Streaming API from Gitter. Then, any incoming message can be read by the approach.

2.1.2 *Preprocessing and Feature Extraction*

In order to represent messages in a format that is appropriate for the next steps in our approach, we first preprocess them by using lemmatization and stop words removal and then extract their representative lexical features using tf-idf.

Lemmatization is the process of transforming words into their root based on their part-of-speech tag within a sentence. We use the NLTK[4] *pos_tag* function for determining a word's part of speech. Each word in a sentence then becomes a tuple of the form <word, postag> which is passed to the NLTK function WordNetLemmatizer to correctly produce the lemmatized word. The entire text of the message is processed this way, resulting in a tokenized list of lemmatized words.

Next, stop words (English common words and programming keywords that have been lemmatized) are removed from the message. We then use Scikit-learn[5] to represent the remaining words using a Term Frequency-Inverse Document Frequency(tf-Idf) vector.

2.1.3 *Classification*

After a message has been preprocessed and its features extracted, GitterAns uses a machine learning classifier trained on a set of previously labeled messages in order to predict whether the text is a troubleshooting question or not. Any machine learning classifier can be used for this step. In our preliminary study, described in Section 3, we evaluated three machine learning algorithms, namely Naïve Bayes, Random Forrest, and Stochastic Gradient Descent. In that same section we also describe the data and the procedure we used in our preliminary study to train and test these classifiers.

If the message is determined to be a troubleshooting question, it is then passed on to the *Search for Answers* step or ignored otherwise.

2.2 Search For Answers

To search for potential answers to a detected troubleshooting question, we use the Google CustomSearch API[6] to search StackOverflow for semantically similar posts to the question. More specifically, we use the troubleshooting question as a query and then use it to run a custom Google search, restricting the scope to only include pages hosted on StackOverflow.com. This allows us to search StackOverflow, while also leveraging the power of the Google search engine, which is known to perform much better than the native StackOverflow search engine. If the query returns search results, they are saved as a JSON object.

2.3 Answer Processing

The JSON object with the potential answers retrieved in the previous step is parsed and reduced to a dictionary of links and post titles. Afterwards, the top 3 results are posted to the Gitter channel in response to the developer's question.

3 Preliminary Study

To evaluate the performance of GitterAns, we performed a preliminary study on the nodejs/node Gitter chat room. In this preliminary study, we evaluated the ability of GitterAns to correctly identify troubleshooting questions in the nodejs/node Gitter channel in terms of classification accuracy, as well as its ability to recommend answers to troubleshooting questions.

3.1 Nodejs Gitter Data

Gitter provides access to its data through a REST API[7]. Using the Gitter API we extracted all of the messages from the nodejs/node chat room[8] and stored them as a JSON file. The resulting dataset contains 109,189 messages.

Since the messages obtained from Gitter do not contain information about whether they are troubleshooting questions or not, manual labeling is needed in order to determine the ground truth for our classification step. First, one of the authors manually labeled 1,700 randomly selected Gitter messages from the dataset. As a result of this initial manual classification, only 54 messages were labeled as troubleshooting questions. This revealed a large imbalance in our data, which could hinder the training of the machine learning component in GitterAns. Therefore, we needed a different approach in order to ensure we obtain more troubleshooting questions for the classifier to learn from.

When inspecting more closely the 1,700 manually labeled messages, we identified a set of unigrams, bigrams, and trigrams that were mostly common in the identified troubleshooting questions: "Is it possible", "Can someone help", "I need help", "I don't understand", "Can someone explain", "How do you", "Question:", "Error", "Error:", "How can I", "Is there any way", "Does anyone know", "How do I", and "?". Therefore, in order to increase our chances of identifying more troubleshooting questions, we filtered the remaining unclassified messages into a smaller subset of possible troubleshooting questions by eliminating all the messages that did not contain any of the previously mentioned substrings. By applying this filter, we reduced the remaining set of messages to 6,437 potential troubleshooting question messages. Afterwards, we randomly sampled 1,000 of these messages, which the first author then manually classified. This round of manual labeling resulted in 641 of the messages

[3] http://pycurl.io/
[4] https://www.nltk.org/
[5] https://scikit-learn.org/stable/
[6] https://developers.google.com/custom-search/v1/overview
[7] https://developer.gitter.im
[8] https://gitter.im/nodejs/node

being classified as non-technical text and 359 messages being classified as troubleshooting questions. We used this last set of 1,000 manually labeled messages for training and testing the machine learning classifier in GitterAns.

3.2 Classifier

In this preliminary study, we explore three machine learning classifiers for classifying messages as troubleshooting questions or not: Multinomial Naïve Bayes (NB)[17], Random Forest (RF)[6], and Stochastic Gradient Descent (SGD)[5]. We also performed hyperparameter tuning for these classifiers.

3.2.1 ***Naïve Bayes (NB)*** is an efficient linear probabilistic classifier that uses Bayes' theorem to identify strong (naive) assumptions between features. NB assumes that all of the features in a given class are conditionally independent of each other [22]. The multinomial Naïve Bayes model captures word frequency information in the documents using a unigram language model with integer word counts. Each document is then typically represented as a vector of integer or real number attributes, which indicate the importance of words in the document[17].

3.2.2 ***Random Forest (RF)*** is a combination of multiple independent decision trees, where each tree is built from a sample drawn with replacement from the training set. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree). However, due to averaging, its variance also decreases. The amount decreased is usually more than enough to compensate for the increase in bias, hence yielding an overall better model[6].

3.2.3 ***Stochastic Gradient Descent (SGD)*** is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention in the context of large-scale learning as it has shown to have high performance for large-scale problems and sparse data[5].

3.2.4 ***Hyperparameter Tuning*** was performed in two stages. First, we created a list of dictionaries that hold the different parameters for each classifier, namely a parameter grid. Second, this list of dictionaries was then passed into a grid search that iterates over every possible combination of parameters in the parameter grid and runs a stratified 10-fold cross-validation on each combination of parameters and returns the best performing parameters for the given data.

We used Scikit-learn to perform the hyperparameter tuning. For the NB model, a total of 96 parameter combinations were created. The RF classifier had 288 parameter combinations. SGD had a total of 4,608 parameter combinations. The parameters we found to work the best for NB were: alpha = 1.0, fit prior = false, max df = 0.4, n-gram range = 1,1. The parameters we found to work the best for RF were: max depth = 2, n estimators = 25, max df = 0.7, n-gram range = 1,2. The parameters we found to work the best for SGD were: alpha = .001, l1 ratio = 0.6, loss = hinge, max iterations = 50, penalty = l2, shuffle = true, tol = 0.1, max df = 0.7, n-gram range = 1,3.

3.3 Answering Questions

In order to determine whether GitterAns is able to find the answers to troubleshooting questions, we implemented a prototype (see Figure 2) and used it to retrieve and post the top 3 potential answers for a random sample of 20 troubleshooting questions extracted from the 359 troubleshooting questions we identified during our final manual labeling process (see Section 3.1). We created a new Gitter channel which we used for testing our bot GitterAns. We then manually posted each of the 20 troubleshooting questions in the testing channel and inspected the answers provided by GitterAns, determining if they actually answered the question or not.

3.4 Results

First, we present the results of the individual machine learning models after hyperparameter tuning and 10-fold cross-validation in terms of classification accuracy. RF showed the poorest performance with an accuracy of 0.64, followed by NB with an accuracy of 0.71, while the SGD model performed the best with an accuracy of 0.78. However, one disadvantage of SGD is the fact that hyperparameter tuning for this model took over 24 hours.

The results of the preliminary study on the potential answers identified by GitterAns for a random sample of troubleshooting questions showed mixed results. On the one hand, for 11 of the 20 technical questions, GitterAns did not provide a StackOverflow post that properly answered the question. On the other hand, for the remaining nine questions, GitterAns was able to recommend a StackOverflow post answering the question as the *first recommendation.*

Previous work has shown the importance of query choice for information retrieval applications on software engineering data [18] and has unveiled the fact that noise can negatively impact the performance of these approaches [19]. We therefore hypothesize that using the whole troubleshooting question as a query may inject too much noise in the search, leading to GitterAns not being able to return proper answers for half of the questions. For example, the following is one of the messages containing a troubleshooting question posted to the nodejs/node Gitter channel: "*@analog-nico I have followed your blog post steps and am able to install all my npm packages for iojs; however when I run the application it throws "Module did not self-register on bcrypt module". A little google search reveals this could be because it can detect multiple node installations. Any suggestions.*". When using the whole text of the message as a query, GitterAns is not able to produce any relevant results. However, we can easily tell that most of the text included in the message (*e.g.*, "I have followed your blog post steps and am able to install all my npm packages for iojs; however when I run the application it throws", "A little google search reveals this could be because", "Any suggestions"), while useful to give context to a reader, it is not necessarily relevant to a search for potential answers. To see if our hypothesis regarding noise in the query holds in this case, we tried removing all the noise and using only the error message the user encountered (*"Module did not self-register on bcrypt module"*) as a query. The results in response to this trimmed query showed that

GitterAns is able to retrieve a valid StackOverflow post addressing the question as the first recommendation. Given this insight, our future work will therefore focus on ways to reduce query noise.

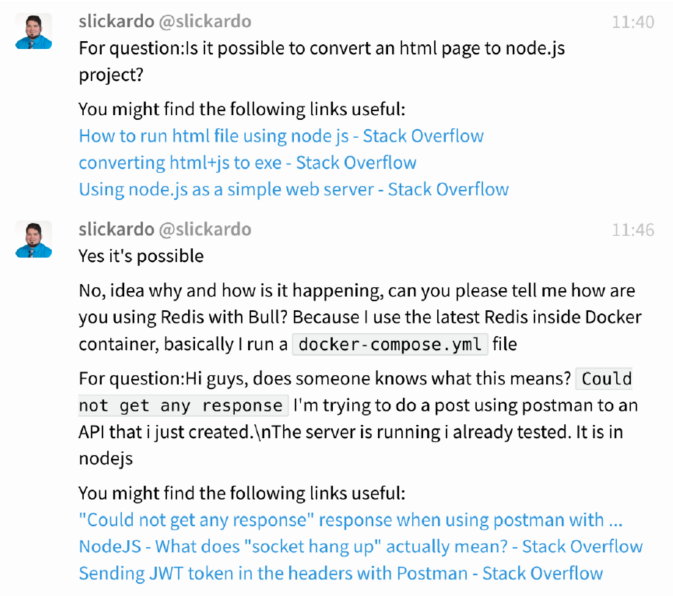

Figure 2: GitterAns in action

4 Related Work

We divide the related work into two subsections. First, we introduce work on the analysis of instant messaging communication tools such as Gitter and Slack in software engineering. Second, we present an overview of work related to bots used by developers.

4.1 Modern Chat Platforms in Software Engineering

With the rise of modern chat platforms such as Slack and Gitter, recent work focused on exploring the developer communities using these tools [1, 2, 4, 10, 15, 20, 21, 24].

Studies with developers have found that developers self-reported using Slack for multiple purposes and to support various activities[15]. Moreover, a study on the communication tools used in 400 open-source software GitHub repositories found that GitHub Issues, personal e-mail, Gitter, Twitter, and mailing lists are the five most popular communication channels currently used in open source development. Slack was found to be eighth in terms of popularity among all the communication means observed [14].

Alkadhi *et al.* [2] presents an exploratory study on the presence of rationale messages that contain information justifying the decisions made throughout the software life cycle (*i.e.*, Rationale messages) in the chat messages of three development teams made up of students working on a multi-project capstone course. Their findings show the presence of rationale, as well as the usefulness of SVM and Naïve Bayes towards the automatic identification of such messages.

Recent work by Stray *et al.* [25] studied a group of 30 developers and their communication through Slack channels at a large software development company. The analysis identified that the messages exchanged were related to the following purposes: general information/coordination, general discussions, problem-focused communication, technical communication, and socializing. Moreover, their results show that in this company, about half of the messages are related to problem-solving (*i.e.*, questions and answers), with very little social talk among the team members.

4.2 Bots in Software Engineering

Developers use bots for several purposes and research efforts have been made toward automating a variety of tasks and improving the efficiency and effectiveness of developers by using bots. We present some examples of bots in software engineering below. For a more detailed overview of this topic, we refer the reader to [12].

Bots like Wunderlist[9] are used for team and task management, distributing tasks among team members, and setting reminders [15]. Other bots are integrated with modern chat and social media platforms, allowing communication between developers and stakeholders. Software bots also integrate with GitHub or assist developers to perform customer support by capturing customer feedback and providing answers to clients [15].

Other recent bots to support software development include: tool-recommender-bot [7], a bot designed to recommend software engineering tools to developers on GitHub, as well as a bot to recommend experts that can potentially answer developers' questions in StackOverflow and Discord [9, 13]. AnswerBot [8, 26] is a similar approach to our work and is a web-based bot that uses Information Retrieval and StackOverflow to generate a set of paragraphs as an answer summary for a given technical problem. Unlike GitterAns, AnswerBot is a stand-alone search engine website and it expects to receive a technical problem formulated as a query.

5 Acknowledgments

Sonia Haiduc and Esteban Parra were supported in part by the National Science Foundation grants CCF-1846142 and CCF-1644285.

6 Conclusion and Future Work

We introduced a new bot called GitterAns, which automatically classifies new messages in Gitter as being troubleshooting questions or not. Then, for each detected troubleshooting question, GitterAns automatically provides potential answers by querying for StackOverflow posts that are semantically similar to the question. We performed a preliminary study on a set of 1,000 messages from a Gitter channel. Our results show promise, as GitterAns is able to classify messages with an accuracy of 0.78. However, when providing answers to the troubleshooting questions, we found that GitterAns was able to suggest correct answers for only 9 of 20 randomly selected questions. We hypothesize this is due to the large amount of noise present in the messages that were used as queries.

Our future work will focus on ways to remove the noise from messages containing troubleshooting questions, as well as on further improving the classification of messages. We also aim to perform a large-scale evaluation of GitterAns and plan to involve real developers in the evaluation.

[9] https://www.wunderlist.com/

References

[1] Rana Alkadhi, Jan Ole Johanssen, Emitza Guzman, and Bernd Bruegge. 2017. REACT: An Approach for Capturing Rationale in Chat Messages. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'17)*. IEEE, Toronto, ON, Canada, 175–180.

[2] R. Alkadhi, T. Lata, E. Guzmany, and B. Bruegge. 2017. Rationale in Development Chat Messages: An Exploratory Study. In *Proceedings of the 14th IEEE/ACM International Conference on Mining Software Repositories (MSR'17)*. 436–446.

[3] Miltiadis Allamanis and Charles Sutton. 2013. Why, When, and What: Analyzing Stack Overflow Questions by Topic, Type, and Code. In *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR'13)*. IEEE, San Francisco, CA, USA, 53–56.

[4] Abram Anders. 2016. Team Communication Platforms and Emergent Social Collaboration Practices. *International Journal of Business Communication* 53, 2 (April 2016), 224–261.

[5] Léon Bottou. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'10)*, Yves Lechevallier and Gilbert Saporta (Eds.). Physica-Verlag HD, 177–186.

[6] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (Oct. 2001), 5–32.

[7] Chris Brown and Chris Parnin. 2019. Sorry to Bother You: Designing Bots for Effective Recommendations. In *Proceedings of the 1st IEEE/ACM International Workshop on Bots in Software Engineering (BotSE)*. IEEE, Montreal, QC, Canada, 54–58. https://doi.org/10.1109/BotSE.2019.00021

[8] Liang Cai, Haoye Wang, Bowen Xu, Qiao Huang, Xin Xia, David Lo, and Zhenchang Xing. 2019. AnswerBot: An Answer Summary Generation Tool Based on Stack Overflow. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'19)*. Association for Computing Machinery, New York, NY, USA, 1134âĂŞ1138. https://doi.org/10.1145/3338906.3341186

[9] Jhonny Cerezo, Juraj Kubelka, Romain Robbes, and Alexandre Bergel. 2019. Building an Expert Recommender Chatbot. In *Proceedings of the 1st IEEE/ACM International Workshop on Bots in Software Engineering (BotSE)*. IEEE, Montreal, QC, Canada, 59–63. https://doi.org/10.1109/BotSE.2019.00022

[10] Preetha Chatterjee, Kostadin Damevski, Lori Pollock, Vinay Augustine, and Nicholas A Kraft. 2019. Exploratory Study of Slack Q&A Chats as a Mining Source for Software Engineering Tools. In *Proceedings of the 16th IEEE International Conference on Mining Software Repositories (MSR'19)*. IEEE, Montreal, Canada, 490–501.

[11] Shaiful Alam Chowdhury and Abram Hindle. 2015. Mining StackOverflow to Filter out Off-topic IRC Discussion. In *Proceedings of the 12th IEEE Working Conference on Mining Software Repositories (MSR'15)*. IEEE, Florence, Italy, 422–425.

[12] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. 2019. Current and Future Bots in Software Development. In *Proceedings of the 1st IEEE/ACM International Workshop on Bots in Software Engineering (BotSE)*. IEEE, Montreal, QC, Canada, 7–11. https://doi.org/10.1109/BotSE.2019.00009

[13] Katsunori Fukui, Tomoki Miyazaki, and Masao Ohira. 2019. A Bot for Suggesting Questions That Match Each User's Expertise. In *Proceedings of the 1st IEEE/ACM International Workshop on Bots in Software Engineering (BotSE)*. IEEE, Montreal, QC, Canada, 18–19. https://doi.org/10.1109/BotSE.2019.00012

[14] Verena Käfer, Daniel Graziotin, Ivan Bogicevic, Stefan Wagner, and Jasmin Ramadani. 2018. Communication in Open-Source Projects-End of the E-mail Era?. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering(ICSE'18)*. IEEE, Gothenburg, Sweden, 242–243.

[15] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW'16)*. ACM, 333–336.

[16] Mario Linares-Vasquez, Bogdan Dit, and Denys Poshyvanyk. 2013. An Exploratory Analysis of Mobile Development Issues Using Stack Overflow. In *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories (MSR'13)*. IEEE, San Francisco, CA, USA, 93–96.

[17] Andrew McCallum and Kamal Nigam. 1998. A Comparison of Event Models for Naïve Bayes Text Classification. In *Proceedings of the 1st AAAI Workshop on Learning for Text Categorization (ICML/AAAI'98)*. AAAI, Madison, WI, USA, 41–48.

[18] Chris Mills, Gabriele Bavota, Sonia Haiduc, Rocco Oliveto, Andrian Marcus, and Andrea De Lucia. 2017. Predicting Query Quality for Applications of Text Retrieval to Software Engineering Tasks. *ACM Trans. Softw. Eng. Methodol.* 26, 1 (2017), 3:1–3:45. https://doi.org/10.1145/3078841

[19] C. Mills, J. Pantiuchina, E. Parra, G. Bavota, and S. Haiduc. 2018. Are Bug Reports Enough for Text Retrieval-Based Bug Localization?. In *Proceedings of the 34th IEEE International Conference on Software Maintenance and Evolution (ICSME'18)*. 381–392. https://doi.org/10.1109/ICSME.2018.00046

[20] Alessandro Murgia, Daan Janssens, Serge Demeyer, and Bogdan Vasilescu. 2016. Among the Machines: Human-Bot Interaction on Social Q&A Websites. In *Proceedings of the 2016 Conference Extended Abstracts on Human Factors in Computing Systems (CHI/EA'16)*. ACM, San Jose, CA, USA, 1272–1279.

[21] Elahe Paikari and André van der Hoek. 2018. A Framework for Understanding Chatbots and Their Future. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'18)*. ACM, Gothenburg, Sweden, 13–16.

[22] Stuart Jonathan Russell and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

[23] M. Storey, A. Zagalsky, F. F. Filho, L. Singer, and D. M. German. 2017. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering* 43, 2 (Feb. 2017), 185–204.

[24] Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *Proceedings of the 24th ACM/SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16)*. ACM, Seattle, WA, USA, 928–931.

[25] Viktoria Stray, Nils Brede Moe, and Mehdi Noroozi. 2019. Slack Me if You Can!: Using Enterprise Social Networking Tools in Virtual Agile Teams. In *Proceedings of the 14th International Conference on Global Software Engineering (ICGSE'19)*. IEEE, Montreal, Quebec, Canada, 101–111.

[26] B. Xu, Z. Xing, X. Xia, and D. Lo. 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 706–716. https://doi.org/10.1109/ASE.2017.8115681