Noname manuscript No.
(will be inserted by the editor)

MSRBot: Using Bots to Answer Questions from Software Repositories

Ahmad Abdellatif · Khaled Badran · Emad Shihab

the date of receipt and acceptance should be inserted later

Abstract Software repositories contain a plethora of useful information that can be used to enhance software projects. Prior work has leveraged repository data to improve many aspects of the software development process, such as, help extract requirement decisions, identify potentially defective code and improve maintenance and evolution. However, in many cases, project stakeholders are not able to fully benefit from their software repositories due to the fact that they need special expertise to mine their repositories. Also, extracting and linking data from different types of repositories (e.g., source code control and bug repositories) requires dedicated effort and time, even if the stakeholder has the expertise to perform such a task.

Therefore, in this paper, we use bots to automate and ease the process of extracting useful information from software repositories. Particularly, we lay out an approach of how bots, layered on top of software repositories, can be used to answer some of the most common software development/maintenance questions facing developers. We perform a preliminary study with 12 participants to validate the effectiveness of the bot. Our findings indicate that using bots achieves very promising results compared to not using the bot (baseline). Most of the participants (90.0%) find the bot to be either useful or very useful. Also, they completed 90.8% of the tasks correctly using the bot with a median time of 40 seconds per task. On the other hand, without the bot, the participants completed 25.2% of the tasks with a median time of 240 seconds per task. Our work has the potential to transform the MSR field by significantly lowering the barrier to entry, making the extraction of useful information from software repositories as easy as chatting with a bot.

Ahmad Abdellatif, Khaled Badran and Emad Shihab
Data-Driven Analysis of Software (DAS) Lab
Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada
E-mail: {a_bdella, k_badran, eshihab}@encs.concordia.ca

Keywords Software Bots · Mining Software Repositories · Conversational Development Assistant

1 Introduction

Software repositories contain an enormous amount of software development data. This repository data is very beneficial, and has been mined to help extracts requirements (e.g., [10,46]), guides process improvements (e.g., [28, 55]) and improves quality (e.g., [29,35]). However, we argue that even with all of its success, the full potential of software repositories remains largely untapped. For example, recent studies presented some of the most frequent and urgent questions (e.g., "Where do developers make the most mistakes?" and "Which mistakes are the most common?") that software teams struggle to answer [16]. Many of the answers to such questions can be easily mined from repository data.

Although software repositories contain a plethora of data, extracting useful information from these repositories remains to be a tedious and difficult task [12,13]. Software practitioners (including developers, project managers, QA analysts, etc.) and companies need to invest significant time and resources, both in terms of personnel and infrastructure, to make use of their repository data. Even getting answers to simple questions may require significant effort.

More recently, bots were proposed as means to help automate redundant development tasks and lower the barrier to entry for information extraction [58]. Hence, recent work laid out a vision for how bots can be used to help in testing, coding, documenting, and releasing software [17,68,59,66]. To bridge the gap between the visionary works and practicality, and to bring those visionary works to life, we devise a framework of using bots over software repositories. Although different bots have been developed for the software engineering domain, no prior work has applied bots to answer the developers questions using the stored data from software repositories.

Although it might seem like applying bots on software repositories is the same as using them to answer questions based on Stack Overflow posts, the reality is there is a big difference between the two. One fundamental difference is the fact that bots that are trained on Stack Overflow data can provide general answers, and will never be able to answer project-specific questions such as "how many bugs were opened against my project today?". Also, we would like to better understand how bots can be applied on software repository data and highlight what is and what is not achievable using bots on top of software repositories.

Therefore, our goal is to design and build a bot framework for software repositories and perform a case study to examine its efficiency and highlight the challenges facing our framework. The approach contains five main components, a **user interaction** component, meant to interact with the user; **entity recognizer** and **intent extractor** components, meant to process and analyze the user's natural language input; a **knowledge base** component, that contains all of the data and information to be queried; and a **response generator** component, meant to generate a reply message that contains the query's answer and return it to the user interaction component. To evaluate our bot approach, we add support for 15 of the most commonly asked questions by software practitioners mentioned in prior work [56,14,54,16,24]. To evaluate our framework, we perform a case study with 12 participants using the Hibernate and Kafka projects. In particular, we asked those participants to perform a set of tasks using the bot then evaluate it based on its replies. We examine the bot in terms of its effectiveness, efficiency, and accuracy and compare it to a baseline where the survey participants are asked to do the same tasks without using the bot. We also perform a post-survey interview with a subset of the survey participants to better understand the strengths and areas of improvements of the bot approach.

Our results indicate that bots are useful (as indicated by 90.0% of answers), efficient (as indicated by 84.17% of answers) and accurate (as indicated by 90.8% of tasks) in providing answers to some of the most common questions. In comparison to the baseline, the bots significantly outperform the manual process of finding answers for their questions (the survey participants were able to only answer 25.2% of the questions correctly and took much longer to find their answers). Based on our post-survey interviews with the participants, we find that bots can be improved if they enable users to perform deep-dive analysis and help compensate for user errors, e.g., typos. Based on our results, we believe that applying bots on software repositories has the potential to transform the MSR field by significantly lowering the barrier to entry, making the extraction of useful information from software repositories as easy as chatting with a bot.

In addition to our findings, the paper provides the following contributions:

- To the best of our knowledge, this is the first study to use bots on software repositories. Also, our framework allows project stakeholders to extract repository information easily using natural language.
- We perform an empirical study to evaluate our bot framework and compare it to a baseline. Also, we provide insights on areas where bot technology/frameworks still face challenges in being applied to software repositories.
- We make our framework implementation [6] and datasets [7] publicly available in an effort to accelerate future research in the area.

Paper Organization. The rest of the paper is organized as follows. Section 2 provides background on the bot and discusses the related work to our study. We detail our framework and its components in Section 3. We explain the questions supported by the bot and questionnaire survey to evaluate our framework in Section 4. In section 5, we report our findings, detailing the usefulness, speed, and accuracy of the bot. Section 6 discusses the bot evaluation and the implications of our results. Section 7 discusses the threats to validity, and section 8 concludes the paper.

2 Background

In this section, we provide a brief background of bots and their roles in the software development process. Also, we discuss the work that is most related to ours. We divide the prior work into two main areas; work related to the visions for the future use of bots, and answering developers' questions.

Software Bots. Storey and Zagalsky defined bots as tools that perform repetitive predefined tasks to save developer's time and increase their productivity [58]. They outlined five areas where they see bots as being helpful: code, test, DevOps, support, and documentation bots. In fact, there exist a number of bots, mostly enabled by the easy integration in Slack that fit into each of the aforementioned categories, for example, Jirafe [22], a code bot that allows developers to report bugs easily. Similarly, Dr. Code is a test bot that tracks technical debt in software projects and many others that notify developers whenever a special action happens, e.g., Pagerbot. One key characteristic of these bots is that they simply automate a task, and do not allow developers or users to extract information (i.e., ask questions, etc.) that they need answers too. In our work, we design and evaluate a bot framework that is able to intelligently answer questions based on the repository data of a specific project.

Recently, software bots are getting more attention from Software Engineering researchers [58]. For example, Wyrich and Bogner [67] implemented a bot that performs code refactoring and creates a pull request with the new changes. Paikari *et al.* [48] developed Sayme, a bot to help developers address code conflicts through its user interface. Moreover, bots are used to detect bugs and generate fixes for them [64,45]. For example, Urli *et al.* [63] developed the Repairnator which is a repair bot for Java programs. The Repairnator keeps monitoring the CI of the project and in the case of a test failure, the bot reproduces the bug and generates a fix. On the other hand, developers can use recommendation bots to find reviewers for a pull request [36], static analysis tools to prevent bugs in their repositories [19], and experts on a source code artifact [20].

Visions for the Future Use of Bots. In addition to the visionary work by Storey and Zagalsky [58], which presented a cognitive support framework in the bots landscape, a number of other researchers proposed work that laid out the vision for the integration of bots in the software engineering domain. In many ways, this visionary work motivates our bot framework. Acharya *et al.* [8] proposed the idea of code drones, a new paradigm where each software artifact represents an intelligent entity. The authors outline how these code drones interact with each other, updating and extending themselves to simplify the developer's life in the future. They see the use of bots as key to bringing their vision to life. Similarly, Matthies *et al.* [40] envisioned a bot that analyzes and measures the software project's data to help development teams track their project progress. On the other hand, researchers envisioned bots that generate fixing patches and validate the refactoring and bug fixes [64], and explain those fixes to the developers [44].

Beschastnikh *et al.* [17] presented their vision of an analysis bot platform, called Mediam. The idea of Mediam is that developers can upload their project to GitHub and allow multiple bots to run on them, which will generate reports that provide feedback and recommendations to developers. The key idea of the vision is that bots can be easily developed and deployed, allowing developers quick access to new methods developed by researchers. Robillard *et al.* [50] envisioned a future system (OD3) that produces documentation to answer user queries. The proposed documentation is generated from different artifacts i.e. source code, Q&A forums, etc.

The gap between these visionary works and industrial use, motivates our work in order to bridge this gap by bringing this visionary work to life. And, to support software developers that have different levels of technical knowledge in their daily tasks. However, our work differs in that we build a bot framework that extracts data from software repositories to allow developers to answer some of the most common questions they have - and our focus is more on how to build and evaluate such a framework.

Answering Developer Questions. The work most related to ours is the work that built various approaches to help developers answer questions they may have. For example, Gottipati *et al.* [27] proposed a semantic search engine framework that retrieves relevant answers to user's queries from software threads. Bradley *et al.* [18] developed a conversational developer assistant using the Amazon Alexa platform to reduce the amount of manual work done by developers (e.g., create a new pull request). Hattori *et al.* [30] proposed a Replay Eclipse plugin, which captures the fine-grained changes and views them in chronological order in the IDE. Replay helps developers answer questions during the development and maintenance tasks. Treude *et al.* [62] proposed a technique that extracts the development tasks from documentation artifacts to answer developers' search queries.

Perhaps the work closest to ours, is the work that applied bots in the software engineering domain. Murgia *et al.* [47], built a Stack Overflow answering bot to better understand the human-bot interaction. They deployed a bot, that impersonated a human and answered simple questions on Stack Overflow. Although their bot performed well, it faced some adoption challenges after it was discovered that it was a bot. Similar to Murgia, Xu *et al.* [68] developed AnswerBot, a bot that can summarize answers (extracted from Stack Overflow) related to developers' questions in order to save the developer time. Tian *et al.* [59] developed APIBot, a framework that is able to answer developers' questions on a specific API using the API's documentation. APIBot is built on the SiriusQA assistant, however the authors' main contribution is the "Domain Adaption" component that produces the questions patterns and their answers. The authors ran APIBot on the Java 8 documentation and performed a study with 92 questions that developers and students asked about Java 8. The results showed that APIBot achieved 0.706 Hit@5 score (i.e., provided an answer in the top 5 answers) based on the questions and answers it was tested with.

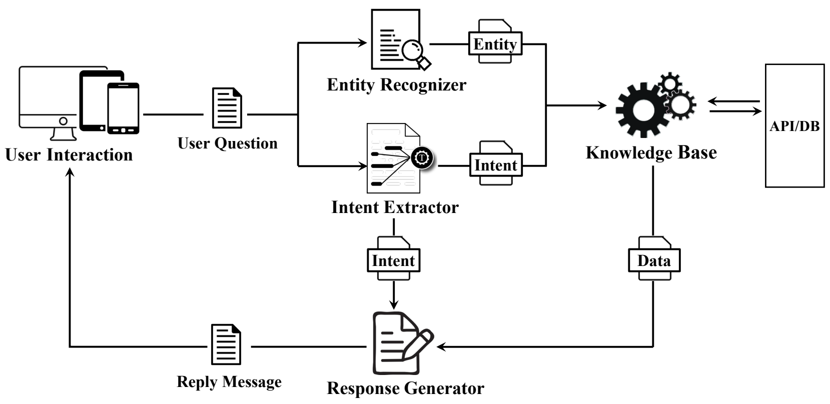


Fig. 1 Overview of the MSRBot Framework and its Components' Interactions

Although our work is similar to the work mentioned above and shares a common goal to answer developers' questions, our work differs from the prior work in several ways. First, our work differs in that we apply bots on software repositories, which brings different challenges (e.g., having to process the repos and deal with various numerical and natural text data) than those compared to bots trained on natural language from Stack Overflow. However, we believe that our work complements the work that supports developer questions from Stack Overflow. Second, our work is fundamentally different, since our goal is to help developers interact and get information about their project from internal resources (i.e., their repository data, enabling them to ask questions such as "who touched file x?"), rather than external sources such as Stack Overflow or API documentation that do not provide detailed project-specific information.
We believe that our work complements the previous work since, instead of using external sources of information to support developers, we focus on using an internal source of information (repositories data). Moreover, our work contributes to the MSR community by laying out how bots can be used to support software practitioners, allowing them to easily extract useful information from their software repositories. And, our findings shed light on issues that need to be addressed by the research community.

3 MSRBot Framework

Our goal is to build a bot that users can interact with to ask questions (such as questions presented in Table 1) to their software repositories. To enable this, our framework is divided into five main parts (as shown in Figure 1), namely 1) user interaction 2) entity recognizer 3) intent extractor 4) knowledge base and 5) response generator. In the following subsections, we detail each of these parts and showcase a working example of our framework.

3.1 User Interaction

Users of bot frameworks need to be able to effectively interact with their information. This can be done in different ways, e.g., through natural language text, through voice and/or visualizations. In addition to handling user input, the user interaction component also presents the output of the question to the user. This is done in the same window and appears as a reply to the user's question. Users are expected to pose their questions in their own words, which can be complicated to handle, especially since different people can pose the same question in many different ways. To help us handle this diversity in the natural language questions, we devise entity recognizer and intent extractor components, which extract structured information from unstructured language input (question) posted by the user. We detail those components in the next subsections.

3.2 Entity Recognizer

The entity recognizer component identifies and extracts a useful information (entity) that a user mentioned in the question using Named Entity Recognition (NER) [60]. Also, it categorizes the extracted entities under certain types (e.g. city name, date, and time). There are two main NER categories: Rule-Based and Statistical NER. Prior work showed that statistical NER is more robust than the rule-based in extracting entities [49, 38, 43]. In the rule-based NER, the user should come up with different rules to extract the entities while in the statistical NER the user trains a machine learning model on an annotated data with the named entities and their types in order to allow the model to extract and classify the entities. The extracted entities help the knowledge base component in answering the user's question. For example, in the question: "Who modified Utilities.java?", the entity is "Utilities.java" which is of type "File Name". Having the file name is necessary to know which file the user is asking about in order to answer the question correctly (i.e. bring the information of the specified file). However, knowing the file name (entity) is not enough to answer the user's question. Therefore, we also need an intent extractor component, which extracts the user's intention from the posed question. We detail this component in the next subsection.

3.3 Intent Extractor

The intent extractor component extracts the user's purpose/motivation (intent) of the question. In the last example, "Who modified Utilities.java?", the intent is to know the commits authors that modified the Utilities file. One of the approaches (e.g., [69]) to extract the intents is to use Word Embeddings, more precisely the Word2Vec model [42]. The model takes a text corpus as input and outputs a vector space where each word in the corpus is represented by a vector. In this approach, the developer needs to train the model with a set of sentences for each intent (training set). Where those sentences express the different ways that the user could ask about the same intent (same semantic). After that, each sentence in the training set is represented as a vector using the following equation:

$$\mathbf{Q} = \sum_{j=1}^{n} \mathbf{Q}_{wj} \quad \text{Where } \mathbf{Q}_{wj} \ \epsilon \ VS \tag{1}$$

where $\mathbf{Q}$ and $\mathbf{Q}_{wj}$ represent the word vector of a sentence and vector of each word in that sentence in the vector space VS, respectively. Afterwards, the cosine similarity metric [33] is used to find the semantic similarity between the user's question vector (after representing it as a vector using Equation 1) and each sentence's vector in the training set. The intent of the user's question will be the same as the intent of the sentence in the training set that has the highest score of similarity. The extracted intent is forwarded to the response generator component in order to generate a response based on the identified intent. Also, the intent is forwarded to knowledge base component in order to answer the question based on its intent. We explain this component in the next subsection.

If the intent extractor is unable to identify the intent (low cosine similarity with the training set), it notifies the knowledge base and the response generator components, which respond with some default reply.

3.4 Knowledge Base

The knowledge base component is responsible for answering the user's questions (e.g. making an API call or querying a DB). It uses the passed intent from the previous component to map it with an API call or DB query that needs to be executed in order to get the answer of the question. And, it uses the extracted entities from the entity recognizer component as parameters for the query or call. For example, if a user asks the question "Which commits fixed the bug ticket HHH-11965?", then the intent is to get the fixing commits and the issue key "HHH-11965" is the entity. So, the knowledge base component uses the identified intent to retrieve the mapped query to the extracted intent and the entity as a parameter to that query. Therefore, the knowledge base component queries the database on the fixing commits (using SZZ [57]) that are linked to Jira ticket "HHH-11965". The component forwards the query's results to the response generator component to generate a reply message on the user's question. In case the intent extractor was unable to identify the intent, the knowledge base will do nothing and wait for a new intent and entities. Furthermore, the knowledge base component verifies the presence of the entities associated with the extracted intent and notifies the response generator in case of missing entities or unable to retrieve the data from the API. We describe the response generator component in the next subsection.

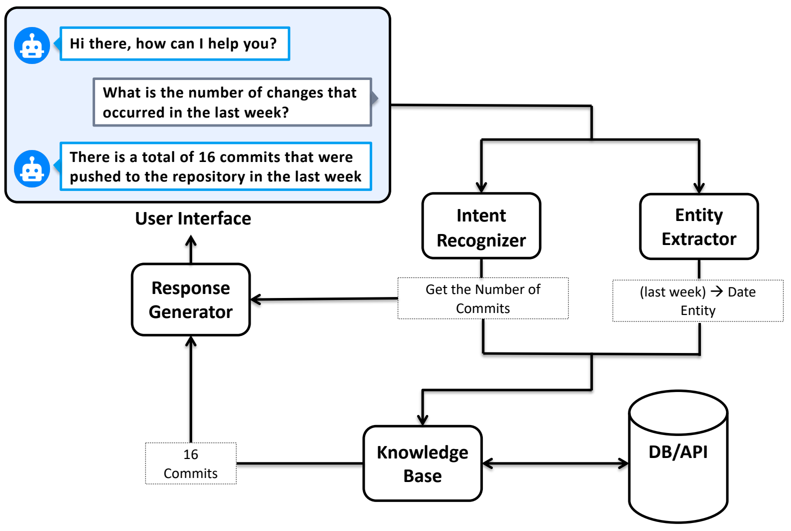

Hi there, how can I help you?
What is the number of changes that occurred in the last week?
There is a total of 16 commits that were pushed to the repository in the last week
User Interface
Intent Recognizer
Entity Extractor
Response Generator
Get the Number of Commits
(last week) → Date Entity
16 Commits
Knowledge Base
DB/API

Fig. 2 Example of MSRBot Framework Components Interactions to Answer User's Question

3.5 Response Generator

The response generator component generates a reply message that contains the answer to the user's question and sends it to the user interaction component to be viewed by the user. The response is generated based on the question asked, and more specifically, the extracted intent of the question. Finally, it sends the generated message to the user interaction component.

In some cases, the bot may not be able to respond to a question due to lack of information (i.e., intents and entities). For example, if it is not possible to extract the intent, the response generator returns a default response: "Sorry, I did not understand your question, could you please ask a different question?". And, in case of a missing entity, the response is "Could you please specify the *entity*?" and it mentions the entity in the message (e.g., file name).

It is worth mentioning that there are certain components that need to be customized to suit our approach. We customize the entity recognizer and intent extractor components to make our bot applicable on software repositories data. For example, we need to train the entity recognizer on the software engineering entities that are specific to software repositories (e.g., Jira tickets and commit hashes) to be able to identify those entities in the posed query. Also, our knowledge base component is specific to software repositories in a number of ways. First, it extracts and links the data from the source code and issue tracking repositories. Second, the knowledge base is customized to retrieve the answer to the user's queries based on the defined intents and entities. On the other hand, the remaining components are applicable to any type of software bots (e.g., user interface).

3.6 Explanatory Example

Putting everything together, we showcase an end-to-end example of the interactions between the components of our proposed framework to answer the user's question. In Figure 2, the user interacts with the bot through the user interface component to inquire about the number of changes that happened in the last week. Next, the user interface component forwards the question to the entity recognizer and intent extractor components to parse the user's query. The entity recognizer component identifies the entity 'last week' (i.e., 21/01/2019 27/01/2019) of type 'Date' in the query. On the other hand, the intent extractor component classifies the intent of the posed query as "Get the Number of Commits". Then, the results of the entity recognizer and intent extractor are sent to the knowledge base component to be processed. Also, the extracted intent is forwarded to the response generator component to generate the reply message. The knowledge base component uses the forwarded intent to retrieve the SQL query that is mapped to it while the entity is used as the parameter for the query. Next, the knowledge base executes the SQL query and retrieves the data from the database. Then, the knowledge base sends the query execution result (i.e., 16 commits) to the response generator. Finally, the response generator uses the results from the knowledge base and the forwarded intent from the intent extractor to generate the reply message ("There is a total of 16 commits that were pushed to the repository in the last week"), and sends it back to the user interface component to present the answer of the question to the user.

4 Case Study Setup

To determine whether using bots really helps answer tasks based on repository data, we perform an experiment with 12 participants using Hibernate-ORM and Kafka repositories. We built a web-based bot application that implemented our framework and had users directly interact with the bot through this web-application. A screenshot of our bot's interface is shown in Figure 3. We divided the participants into two groups (each of 6 participants). Then, we sent emails that include links to the bot and online survey to both groups' members, asking each group to perform a set of tasks using the bot. Finally, we examine the bot in terms of its effectiveness, efficiency and accuracy and compare it to a baseline where both groups' members are asked to perform the same tasks without the bot. It is important to emphasize that each group performs the same set of tasks related to a specific repository. In other words, all members of group 1 performed the tasks using Hibernate-ORM, while the members of the group 2 used Kafka project.

To extract the intents and entities, we leveraged Google's Dialogflow engine [25]. Dialogflow has a powerful natural language understanding (NLU) engine that extracts the intents and entities from a user's question based on a custom NLP model. Our choice to use Dialogflow was motivated by the fact that it can be integrated easily with 14 different platforms and supports more than 20 languages. Furthermore, it provides speech support with third-party integration and the provided service is free. These features make it easier to enhance our framework with more features in the future.

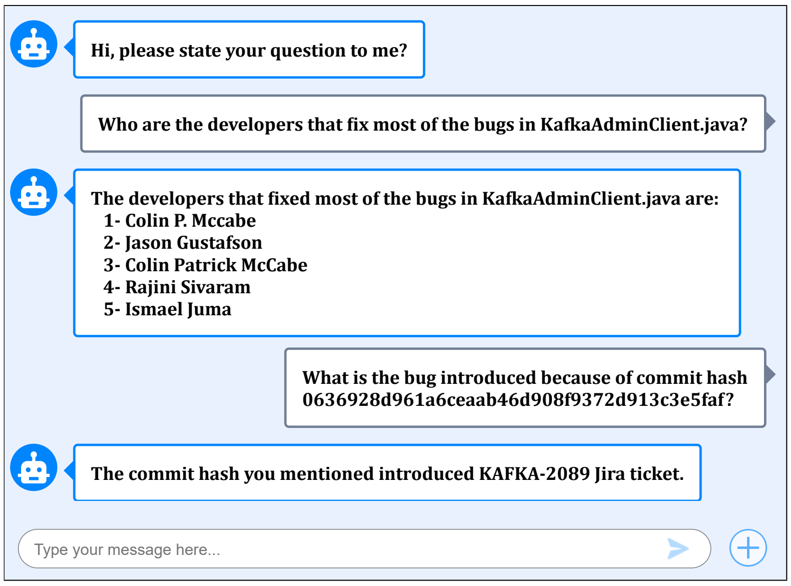
Any NLU model needs to be trained. Therefore, to train the NLU, we followed the same approach as Toxtli et al. [61]. Typically, the more examples we train the NLU on, the more accurate the NLU model can extract the intents and entities from the users questions [26]. As a first step, we conducted a brainstorm session to create the initial training set which represents the different ways that the developers could ask for each intent in Table 1. Moreover, our bot can handle basic questions such as greeting users and asking general questions about the bot such as: "How are you?". Then, we used the initial set to train the NLU and asked two developers (each has more than 7 years of industrial experience) to test the bot for one week. During this testing period, we used the questions that the developers posed to the bot to further improve the training of the NLU. The training data is publicly available on [7].

Although we use Dialogflow in our implementation, it is important to note that there exist other tools/engines that one can use such as Gensim [2], Stanford CoreNLP [39], Microsoft's LUIS [41], IBM Watson [32], or Amazon Lex [1].

To ensure that the usage scenario of the Bot is as realistic as possible, we asked the participants to perform a set of tasks using the bot. We used the questions that have been identified in the literature as being of importance to developers [16,15,24,54,14,56] to formulate the tasks in our experiment. The participants use the bot by posing any number of natural language questions needed to complete the task. Next, the participants evaluate the bot by answering a set of questions related to the task and bot's reply. To compare, we also ask the participants to perform the same tasks without the bot, which we call the baseline comparison. We detail all the steps of our case study in this section.

4.1 Questions Supported by the Bot

To perform our study, we needed to determine a list of questions that our bot should support. To do so, we surveyed work that investigated the most commonly asked questions of software practitioners (mostly developers and managers). We found a number of notable and highly cited studies, such as the study by Begel and Zimmermann [16,15], which reported questions commonly asked by practitioners at Microsoft, the study by Fritz and Murphy [24] that


Hi, please state your question to me?

Who are the developers that fix most of the bugs in KafkaAdminClient.java?

The developers that fixed most of the bugs in KafkaAdminClient.java are:
1- Colin P. Mccabe
2- Jason Gustafson
3- Colin Patrick McCabe
4- Rajini Sivaram
5- Ismael Juma

What is the bug introduced because of commit hash 0636928d961a6ceaab46d908f9372d913c3e5faf?

The commit hash you mentioned introduced KAFKA-2089 Jira ticket.

Fig. 3 An Example User Conversation with MSRBot

conducted interviews with software developers to determine questions that developers ask and the study by Sharma *et. al* [54], which prioritized the importance of questions that developers ask. We also surveyed a number of other studies whose goal was not directly related to questions that developers ask, but which we found to be relevant for us to understand which questions we should ask (e.g., [14,56]). In most of these studies, *the authors reported that software practitioners are lacking support in answering these questions, which is exactly what our bot can help provide.* After going through the literature, we selected arbitrarily 15 questions that our bot can support and *can be answered using repository data.*

Table 1 presents the questions we use in the case study, the rationale for supporting the question and the study where the question was mentioned/motivated from. Each question represents an intent and the bold words represent the entities in the question. For example, the user could ask Q11 as: "What are the buggy commits that happened last week?", then the intent is "Determine Buggy Commits" and the entity is "last week". It is important to emphasize that the bot's users can ask the questions in different ways other than what is mentioned in Table 1. In the last example the user can ask the bot "What are the changes that introduced bugs on Dec 27, 2018" where the intent remains the same although the question is asked in a different way and the entity is changed to a specific date (Dec 27, 2018).

Although we support 15 questions in our prototype at this time, it is important to note that the bot framework can support many more questions and we are extending it to do so now. We opted to focus on these 15 questions

Table 1 List of Questions Examined by MSRBot and the Rationale for Their Inclusion

Question	**Rationale**	**Ref.**
Q1. Which commits fixed the **bug id**?	To refer the commit to a developer who is working on similar bug.	[16]
Q2. Which developer(s) fixes the most bugs related to **File Name**?	To know which developer(s) have experience in fixing bugs related to a specific component or file.	[16]
Q3. Which are the most bug introducing files?	To refactor the buggy files in the repository.	[16]
Q4. Who modified **File Name**?	To know which developer(s) worked on the file in order to ask them about a piece of code.	[54,24]
Q5. Which are the bugs introduced by commit **Commit Hash**?	To study the type of bugs introduced because of certain commit.	[16]
Q6. What is the number of commits in/on **Date**?	To track the progress of the team members at particular time (e.g., in the last day, week, month).	[24]
Q7. What commits were submitted on **Date**?	To know exactly what commits were done, to flag for review, testing, integration, etc.	[24]
Q8. What is/are the latest commit(s) to **File Name**?	Developers may want to know what are the last things that changed in a file/class to be up to date before they make modifications.	[54,24,15]
Q9. What are the commits related to **File Name**?	To track changes which are happening on the file which the developer is currently working on. Or to know the changes happening to a file that was abandoned by a developer.	[14]
Q10. What is/are the most common bug(s)?	The team wants the most important/common bug (registered as watchers) so it can be addressed.	[16]
Q11. What are the buggy/fixing commits that happened in-/on **Date**?	To know the buggy or fixing commits happened at a particular time e.g. before release date.	[16]
Q12. How many bugs have the status/priority?	Quickly view the number of bug reports with a specific status (e.g., open) or priority (e.g., blocker) plan a fix for it.	[16]
Q13. Who is the author of **File Name**?	Developers who have questions about a specific file or class may want to speak to the person who created it.	[56,54]
Q14. Which developer(s) have the most unfixed bugs?	To determine the overloaded developers and possibly re-assign bugs to others on the team.	[16]
Q15. What is the percentage of bug fixing commits that introduced bugs in/on **Date**?	To study the characteristics of the fixing commits which happened at a certain time and induced bugs.	[16]

since our goal is to evaluate the bot in this research context and wanted to keep the evaluation manageable.

Table 2 Participants' Knowledge on Version Control Repositories and Issue Tracking System

Likert Scale	**Number of Participants**	
	Version Control Repositories (Git)	**Issue Tracking System (Jira)**
1 (No Knowledge)	0	1
2 (Entry Level)	2	4
3 (Intermediate)	3	4
4 (Competent)	5	1
5 (Expert)	2	1

4.2 Study Participants

Once we decided on the 15 questions that we are able to support, we want to evaluate how useful the bot is. Since bots are meant to be used by real software practitioners, we decided to evaluate our bot through a user study.

Our user study involved 12 participants. For each participant, we asked them about their main occupation, background, software development experience and their knowledge of software repositories. All participants were graduate students (4 Ph.D. and 8 master students). Of the 12 participants, 75% have more than 3 years of professional software development experience and 25% have between 1-3 years of development experience. The participants' experience using Version Control Repositories (e.g., Git) and Issue Tracking System (e.g., Jira) are shown in Table 2.

We deliberately reached out to graduate students to conduct our user study for a few specific reasons. First, we knew that most of these graduate students (80%) have worked in a professional software development environment in the past. Second, since this is one of the first studies using bots, we wanted to interview some of the participants in person, which provides us with invaluable feedback about aspects of using bots that may not come out in the user study. Expecting developers from industry, who are already busy and overloaded, dedicate this much time to our study would be difficult and if they did, we would not be able to go as deep in our study with them. Also, as prior work has shown, students can be a good proxy for what developers do in professional environments, especially if the participants are experienced and the technology under study is new, which is our case [51,31]. Lastly, the main goal of our case study is to evaluate the proposed framework, i.e., this is a judgment study rather than a sample study, using students is completely valid. The real important factors for us is not the characteristics of the participants (students), rather the evaluation of the framework. Once we recruited our participants, we devised a questionnaire survey to evaluate the bot and baseline approaches. We detail our survey next.

4.3 Questionnaire Survey

We devised a survey that participants answer to help us understand the usefulness of the bot. To make the situation realistic, we mined the data from the Git and Jira repositories of the Hibernate-ORM and Kafka projects. Hibernate is a Java library that provides Object/Relational Mapping (ORM) support. And, Kafka is a Java platform that supports streaming applications. We setup our bot framework to be able to answer all the supported questions on Hibernate's and Kafka's repository data. There was no specific reason for choosing those projects as our case study, however, they did meet some of the most common criteria - they are large open source projects (Hibernate with 177 releases and Kafka with 97 releases) that uses Git and Jira, they have rich history (each has more than 6,500 commits, 8,800 bug reports), are popular amongst developers (each has more than 340 unique contributors) and have been studied by prior MSR-type studies (e.g., [9,53,34,21]).

Our survey was divided into three parts. The first section gathered information about the participants and asked questions related to experience, current role, and knowledge in mining software repositories, which is the information we presented in the section above. The second part of the survey was composed of 10 tasks that the participants are asked to perform. The task statements we gave the users all the needed information to complete the task. For example, the given task statement would say "ask about the commits that fix HHH-6574", and a user might use the bot to perform the task by asking: "which commits fixed the bug id?" In this case, the user is free to ask the question in any way they prefer, e.g., what are the fixing commits of HHH-6574 ticket? We provide a Jira ticket number that exists in Hibernate and Kafka, since we do not expect the user/participant to know this, however, someone related to any project is asking this question, s/he would indeed have such information. The remainder of the second part contained questions for the participants to evaluate the bot based on the answer they receive. We discuss the questions used to evaluate the bot in the next section.

In addition to asking the participants to complete the tasks using the bot, we also got them to perform the same tasks manually (without using the bot) to have a baseline comparison. For the baseline evaluation, we gave the exact tasks formulated from the questions shown in Table 1 to the participants, so they know exactly what to answer to. The participants were free to use any technique they prefer such as writing a script, performing web searches, using tools (e.g., gitkraken [3] and Jira Client [5]), executing Git/Jira commands, or searching manually for the answer in order to complete the tasks. Our goal was to resemble as close to a realistic situation as possible.

4.4 Evaluating the Bot

Bots are typically evaluated using factors that are related to both, accuracy and usability [65]. Particularly, this work suggested two main criteria when evaluating bots:

- **Usefulness:** which states that the answer (provided by the bot) should include all the information that answers the question clearly and concisely [70,52].
- **Speed:** which states that the answer should be returned in a time that is faster than the traditional way that a developer retrieves information [70].

In essence, bot should help developers in performing their tasks and do this in a way that is faster than if you were not using the bot. In addition to the two above evaluation criteria, we added another criteria, related to the accuracy of the answers that the bot provides. In our case, we define **accuracy** as the number of correctly completed tasks performed by the bot, where the task is marked as correct if the returned answer by the bot matches the actual answer the actual answer to the question [65]. We formalize our case study with three research questions that are related to the three evaluation measures used, in particular we ask:

RQ1: How useful are the bot's answers to users' questions?
RQ2: How quickly can users complete their tasks using the bot?
RQ3: How accurate are the bot's answers?

To address **RQ1**, we ask two sub-questions. First, we ask whether the bot was able to return an answer in the first place (in some cases it cannot) and we ask the participants whether they consider that the answer returned is useful in answering the question posed on a five point Likert's scale (from very useless to very useful).

For **RQ2**, we recorded the actual time the participants need to perform each of the tasks while using the bot through an online survey tool. For each task, we measure the time starting from the moment the participant is given the task until s/he submits the task. We use this time to quantitatively compare the time savings to the baseline, i.e., accomplishing the same tasks without using the bot. In addition, we ask the participants to indicate how fast the bot replies to their questions on a five point Likert's scale from very slow to very fast (to measure perceived speed). For the case of the baseline, we asked the participants to measure and report the time it took them to complete each task. We limited the maximum time to finish each task to 30 minutes when using the bot and in the baseline, i.e., in case they did not manage to get an answer within 30 minutes, we considered the task to be incomplete.

To address **RQ3**, we recorded all the interactions performed by the participants and, more importantly, the output of each bot's components including its responses. Then, we analyzed the tasks' results manually to determine if a task is completed correctly or not (by cloning the repositories and writing the scripts to answer the questions). For example, if the participant asks for the number of commits on a particular day, we would check if the returned answer was actually correct or not. This enables us to ensure that the entity recognizer, intent extractor, mapping process in the knowledge base, and the response generator components are working correctly. In the case of the baseline questionnaire, we asked the participants to provide us the answer of each task in the survey. This allowed us to determine the accuracy of the manually determined tasks. To ensure that the participants actually searched the answer for the asked task, we required that the participants briefly explain how they performed the task. Having this information also provided us with insight into how much work and what tools/techniques/commands practitioners typically use to perform such tasks. Finally, at the end of the survey, we added an optional field to allow the participants to write their comments or suggestions, if they have any.

To avoid overburdening the participants, we divided them into two groups, where each group has 6 participants and is given 10 tasks to perform on a certain repository. The tasks were formulated from 10 randomly selected questions from the list of questions supported by the bot. The questions that were selected to formulate the tasks are Q1, Q2, Q3, Q5, Q6, Q7, Q9, Q11, Q14, and Q15 in Table 1. The first group performed the tasks on the Hibernate project while the members of the second group are instructed to do the same tasks on the Kafka project. Both groups (the Hibernate and Kafa groups) were asked to perform the tasks twice, once using the bot and another time without the bot (which we call the baseline). None of the participants knew the questions that the bot was trained on. This is to ensure they will use their own words when they are interacting with the bot and to monitor the questions that the participants ask the bot about. It is important to emphasize that each group received the same tasks in both, the baseline questionnaire and the bot-related questionnaire.

5 Case Study Results

In total, the 12 participants asked the bot 165 questions (some developers asked more than 10 questions) to perform the assigned tasks [7]. Of the 165 questions, we excluded 9 questions from our analysis because they were out of scope (e.g., “What's your name?”, “What language are you written in?”), as the main focus of this work is to study bots on software repositories. Therefore, all of the presented results are based on the remaining 156 questions that are relevant.

5.1 RQ1: How useful are the bot's answers to users' questions?

As mentioned earlier, one of the first criteria for an effective bot is to provide its users with useful answers to their questions. Evaluating a bot by asking how useful its answers were commonly used in most bot-related research (e.g. [68, 70, 23]).

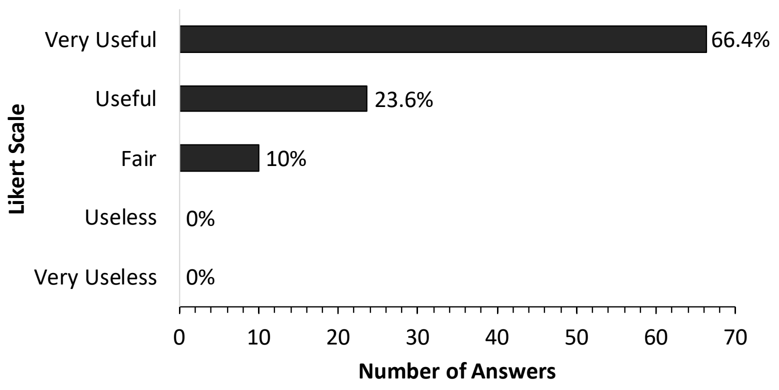


Fig. 4 Usefulness of the Bot's Answers

Participants were asked to indicate the usefulness of the answer provided by the bot after each question they asked. The choice was on a five-point Likert's scale from very useful (meaning, the bot provided an answer they could actually act on) to very useless (meaning, the answer provided does not help answer the question at all). The participants also had other choices within the range, which were: useful (meaning, the answer was helpful but could be enhanced), fair (meaning, the answer gave some information that provided some context, but did not help the answer fully) and useless (meaning, the reply did not help with the question, but a reply was made).

Figure 4 shows the usefulness results in case they were correct. Overall, **90.0% of the participants indicated that the results returned by the bot were considered to be either useful or very useful**. Another 10.0% indicated that the bot provided answers that were fair, meaning the answers helped, but were not particularly helpful in answering their question. It is important to emphasize that when considering usefulness, we considered all tasks that were performed correctly.

Upon closer examination of the fair results, we found a few interesting reasons that lead users to be partially dissatisfied with the answers. First, in some cases, the users found that the information returned by the bot to not be easily understandable. For example, if a user asks for all the commit logs of commits that occurred in the last year, then the returned answer will be long and terse. In such cases, the users find the answers to be difficult to sift through, and accordingly indicate that the results are not useful. Such cases showed us that perhaps we need to pay attention to the way that answers are presented to the users and how to handle information overloading. We plan to address such issues in future versions of our bot framework. Another case is related to information that the users expected to see. For example, some users indicated that they expect to have the commit hash returned to them for any commit-related questions. Initially, we omitted returning the commit hashes (and generally, identification info) since we felt such information is difficult

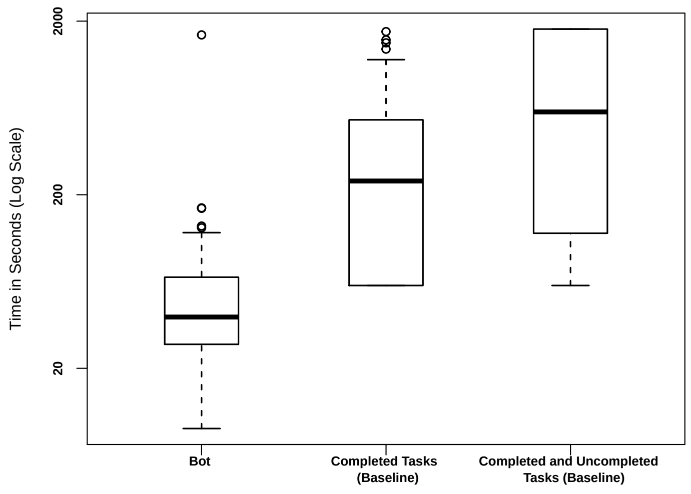


Fig. 5 Time Required to Complete Tasks (bot vs. baseline)

to read by users and envisioned users of the bot to be more interested in summarized data (e.g., the number of commits that were committed today). Clearly, the bot proved to be used for more than just summarized information and in certain cases users were interested in detailed info, such as a commit hash or bug ID. All of these responses provided us with excellent ideas for how we will evolve the bot.

The majority (90.0%) of the bot's users found it to be useful or very useful. Areas for improvement include figuring out how to effectively present the bot's answers to users.

5.2 RQ2: How quickly can users complete their tasks using the bot?

Since bots are meant to answer questions in a chat-like forum, speed is of the essence. Therefore, our second RQ aims to shed light on how fast can users perform tasks related to their repositories using the bot and compare that to the time they need to complete the same tasks without the bot (i.e., the baseline). We also ask the users to indicate their perceived speed of the bot.
Measured speed. We measure the exact time that the participants needed to complete each of the given tasks, once using the bot and another without the bot, which we call the baseline. This gives us better insights about the actual time savings when using the bot.

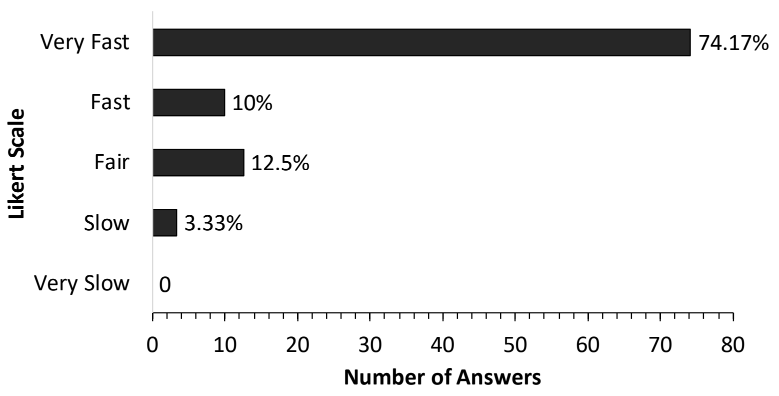


Fig. 6 Speed of the Bot's Reply

Figure 5 shows boxplots of the distribution of the time it took for the participants to perform the tasks, with and without the bot (note that the y-axis is log-scaled to improve readability). As evident from Figure 5, using the bot (the left most box plot) significantly outperforms the baseline approach, achieving a median task completion time of 40 seconds and a maximum of 1666 seconds. On the other hand, for the baseline approach, we have two results - one that considers all tasks that users were able to complete (labeled "Completed Tasks (Baseline)" in Figure 5) and the other considering all tasks, i.e., completed and uncompleted[1] (labeled "Completed and Uncompleted Tasks (Baseline)" in Figure 5). The median task completion time for the tasks in the baseline approach is 240 seconds with a maximum of 1,740 seconds. While, if all the completed and uncompleted tasks are considered, the time to perform a task is even higher, with a median of 600 seconds and a maximum of 1,800 seconds. To ensure that the difference between the bot and the two cases of the baseline is statistically significant, we performed a Wilcox test, and the difference in both cases (i.e., using the bot vs. completed tasks in the baseline and using the bot vs. all tasks in the baseline), and find that the difference is statistically significant (i.e., p-value$\leq$ 0.01). It is obvious that using the bot to perform tasks is faster than the baseline approach. However, this research question investigates the amount of time that the bot saves compared to users manually doing the tasks, which in our case is more than 3 minutes/task, on median.

Perceived speed. The other side of the coin is to determine how users perceive the speed of the bot to be. To accomplish this, we asked users to indicate how fast they received the answer to their question from the bot. Once again, the choices for the users were given on a five point Likert's scale, from very fast (approx. 0 - 3 seconds) to very slow ($\geq$ 30 seconds). The participants also

[1] Since we gave a maximum of 30 minutes for participants to complete a task, tasks that were not answered after 30 minutes were considered to be incomplete and also to have taken 30 minutes.

had other choices within the range, which were: fast (4 - 10 seconds), fair (11 - 20 seconds) and slow (21 - 30 seconds).

Figure 6 shows the results of the survey participants. The majority of the responses (84.17%) indicated that the bot's responses were either, fast or very fast. The remaining 15.83% of the replies indicated that the bot's response was either fair or slow. Clearly, our answers show that the bot provides a significant speed up to users.

Deep-dive analysis. To better understand why some of the tasks took longer to perform using the bot, we looked into the logged data and noted 4 cases that may have impacted the response speed of the bot. We found that in those cases, Dialogflow took more than 5 seconds to extract intents and entities from the user's question. We searched for the reasons behind Dialogflow's delay and found that the way users ask questions can make it difficult for Dialogflow's algorithms to extract the entities and intents. On the other hand, there is one case where the participant required more than 30 minutes to complete their task using the bot. We followed up with the participant afterwards to determine the reason and were told that the participants simply "forgot" to complete the task since they were distracted.

As for the case where users took a long time to find that answers in the baseline case, we found that the main reason for such delays is that some tasks were more difficult to answer. Hence, users needed to conduct online searches (e.g., using Stack Overflow) of ways/techniques that they can use to obtain the answer.

That said, overall, the participants were fast in completing tasks using the bot. It is important to keep in perspective how much time using the bot saves. As we learned from the feedback of our baseline experiments, in many cases, and depending on the task being performed, a developer may need to clone the repository, write a short script, and process/clean-up the extracted data to ensure that they complete the tasks correctly - and that might be a best case scenario. If the person looking for the information is not very technical (e.g., a manager), they may need to spend time to learn what commands they need to run or tools to use, etc., which may require several hours or days.

The participants take a median time of 40 seconds to perform a task using the bot. Moreover, the majority (84.17%) of the bot's users perceived the bot's responses to be fast or very fast. However, the way that the user frames the question may impact the speed of the bot's reply.

5.3 RQ3: How accurate are the bot's answers?

In addition to using the typical measures to evaluate bots, i.e., usefulness and speed, it is critical that the bot returns accurate results. This is of particular importance in our case, since software practitioners generally act on this information, sometimes to drive major tasks.

Table 3 Reasons for Uncompleted Tasks by the Bot

Reason	**Number of Questions**
Extract Intent	5
Recognize Entity	5
Developer's Distraction	1
Out of scope	9

Bot's performance. We measure accuracy by checking the tasks' results performed by the users using the bot and comparing it with the actual answer to the task if it was queried manually by cloning the repositories then writing a script to find the answer or executing git/Jira commands. For example, to get the developers who touched the "KafkaAdminClient" file, we ran the following git command: "git log –pretty=format:%cn – clients/src/main/java/org/apache/kafka/clients/admin/KafkaAdminClient.java". This RQ checks each component's functionality in the framework. Particularly, it checks whether the extraction of the intents and entities is done correctly from the natural language question posed by the users. Moreover, we check whether our knowledge base component queries the correct data and if the response generator produces the correct reply based on the intent and knowledge base, respectively. In total, the first two authors manually checked all the 120 completed tasks by the participants using the bot.

Our results showed that the users correctly completed 90.8% (109 of 120) of the tasks using the bot. Manual investigation of the correct tasks showed that the bot is versatile and was able to handle different user questions related to the tasks. For example, the bot was able to handle the questions *"tell me the number of commits last month"* asked by participant 1 vs. *"determine the number of commits that happened in last month."* asked by participant 2 vs. *"how many commits happened in the last month"* from participant 3, which clearly have the same semantics but different syntax.

Our findings indicate that the 10 tasks that the users fail to complete correctly were due to the incorrect extraction of intents or entities by our trained NLU model as shown in Table 3. For example, in one scenario the user asks "tell me the commit info between 27th May 2018 till the end of that month?" and our NLU model was unable to identify the entity (because it was not trained on the date format mentioned in the participant's question). Consequently, the knowledge base and the response generator components mapped the wrong entity and returned an incorrect result. Interestingly, in such cases, some of the participants had to ask the bot more than once to complete the task correctly. For example, P1 posed the following question "how many of June 2018 bugs are fixed" to perform task 10, the bot fails to extract the correct intent (the fixing commits that induce bugs) from the question, which lead to an incorrect reply (returned the developers that are expert in fixing bugs). Consequently, the participants rephrased the query to the bot as follows: "show me the percentage of bugs fixes that introduced bugs in June 2018" which allowed the bot to return the correct answer. Overall, the

participants asked 1.3 question per task, on average. It is important to note that we consider the task where the participant is distracted as incomplete since s/he took more than 30 minutes.
Baseline performance. As mentioned earlier, we also conducted a baseline comparison where we asked users to perform the same tasks without the bot. Figure 7 shows a break down of 1) the number of completed tasks and 2) the number of *correct* tasks per completed tasks. On the positive side, we can see that the survey participants were able to provide some sort of answer for all tasks, albeit some of the tasks (e.g., T3, T6, T11 and T15) had less answers from participants. Across all tasks, the participants provided some sort of answer in 62.6% of the cases.

However, what is most interesting is that the number of correct answers is much lower. Across all tasks, the survey participants provided the correct answer in 25.2% of the cases. For example, for T3, T11 and T15, all of the provided answers were incorrect. On the other hand, T9's answers were all correct.
This outcome highlights another (in addition to saving time) key advantage of using the bot framework, which is that reduction of human error. When examining the results of the baseline experiments, we noticed that in many cases participants would use a wrong command or a slightly wrong date. In other cases where they were not able to provide any answer, they simply did not have the know how or failed to find the resources to answer their question within a manageable time frame.

Overall, the bot achieves an accuracy of 90.8% in answering user's questions, which is much higher than the baseline's accuracy of 25.2%. Techniques to make the NLU training more effective can help further improve the bot's accuracy.

5.4 Follow-up Interviews

The survey results provided us with an excellent way to quantify the usefulness, efficiency, and accuracy of the bot. However, we wanted to obtain deeper insights, particularly related to what the participants felt were the strengths and areas for improvement for the MSR-related bot framework. Therefore, we conducted semi-structured interviews, where we sat with 5 of the 12 survey participants and asked them: 1) What they believe are the strengths of using a bot framework? 2) What they believe can be improved? and 3) any general comments or feedback they had for us? We asked for permission to record the results of the interview, to enable us to perform deeper offline analysis of the results.

In terms of strengths, most of the points mentioned during the interview surrounded the benefits and applicability of the bot on top of software repositories in industry/practical settings. We elaborate on some of the points pointed by the interviewees below:

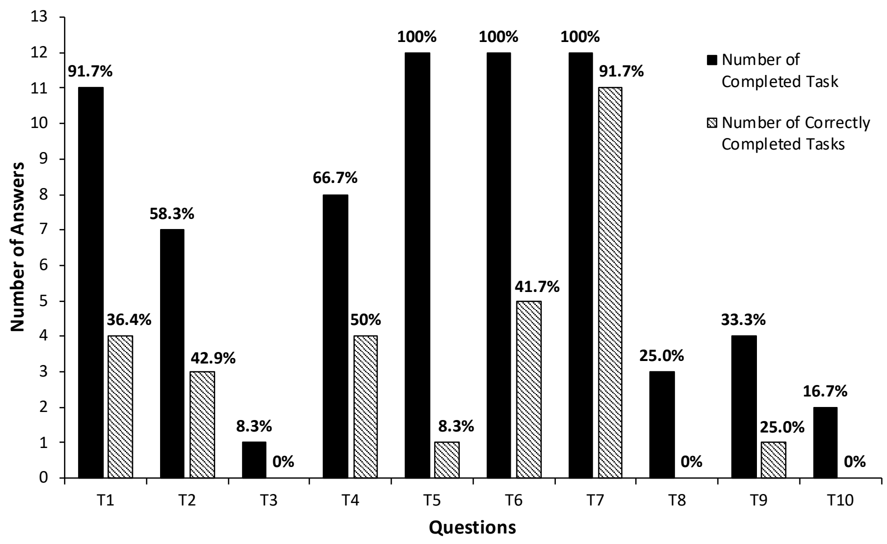


Fig. 7 Number of Answers for Each Task in the Baseline

- **Bots are useful in software projects where personnel with many roles are involved.** Although we knew from the RQs that bots will be useful, however, our interview revealed to us that bots are probably useful to more than just developers. For example, P1 and P2, two of the most experienced participants indicated that usually large projects involve personnel with varying technical backgrounds. And it is usually the less technical personnel (e.g., project managers) that can significantly benefit from repository information, but often lack the know-how to extract such information.
- **Bots are very efficient, even when large and long-lived projects are being analyzed.** Another major strength pointed out is the ability of the bot to provide answers from a very rich history very quickly. For example, P4 mentions the fact that some repos "have more than 20,000 [or more] commits and if you are going through like 20,000 bugs or specific commits it will take so much time and probably you may even miss important data easily".
  Another participant, P5 had a different perspective and saw the potential for bots to help researchers that study software projects. Particularly, he pointed out that our bot framework can be used by researchers who may want to get a few quick answers about a project or do some deep-dive analysis based on some of their findings. For example, one can ask the bot how many changes or how many bugs were reported against a file that they found to yield interesting results in their analysis.
- **Bot's have a very low barrier-to-entry, especially since they can understand natural language.** The participants pointed out that a major advantage is that they did not need to 'learn' any specific technology

or language to interact with the bot. This significantly lowers the barrier to entry for adopters of our framework. For example, P3 says “I like that you type in natural language without thinking about building a query to do the thing [answer the question at hand]”

Other points mentioned reaffirmed our quantitative findings. For example, participants P2, P3 and P5 all mentioned the bot’s speed in replying to questions and the fact that the bot helps complete the tasks at hand faster as a major benefit. We do not discuss this in detail here, since we believe our results already discussed such point and there is no point in repeating the same points again.

We also gained some valuable feedback about the potential areas of improvement for the bot framework. We elaborate on some of the points mentioned by the interviewees below:

– **Support deep-dive of answers provided.** The participants mentioned that although they appreciated the simplicity and clarity of the bot’s answers, you see adding the ability of allowing the user to dive more into the results as a potential future feature. For example, participant P1 mentioned that it would be great to add hyperlinks for commit hashes or bug IDs that are returned. We believe that such a feature is indeed warranted and plan to (and believe we could easily) implement such a feature.
– **Make the bot more resilient and modifiable.** One clear limitation of our bot framework is that it supports a limited number of questions, even though we did that since our goal is to evaluate the viability of using a bot on top of software repositories. In any case, the participants P2 and P3 suggested that a clear improvement is to support more questions. We certainly plan to look into ways that can make our bot learn effectively from questions that are not yet supported, based on the questions users ask. Another participant, P4 suggested that we try and add a recommendation system to the bot so that typos are fixed with a “did you mean ...” type of messages. Also, questions that might be mistyped, e.g., how vs. who, can be provided with a suggested fix.

That said, all participants strongly showed support for the idea and mentioned they see a lot of potential for the combination of bots and software repositories. Our work is a step in the right direction, showing the value and applicability of using bots to effectively extract useful information easily from software repositories.

6 Discussion

In this section, we present other perspectives in evaluating the software bots, and discuss the future and practical implications of our work.

6.1 Bots Evaluation

One of the main challenges in our work is the evaluation part since there is not much prior work that evaluates the use of bots on software repositories. One of the goals of using bots in the software engineering domain is to improve developers' productivity [58]. Therefore, we believe that any proposed bot should be evaluated against the methods that developers usually use to perform their tasks.

As discussed in section 4, we evaluate the proposed framework according to its effectiveness and usefulness for developers in performing their tasks. However, there may be other ways to assess software bots. For example, usage, which can be measured by the number of times that a user uses the bot in a certain duration (e.g., a week). Other measures for speed can be related to the number of interactions needed for a user to complete a task, i.e., asking more questions to perform one task increases the time required to complete that task.

In our case study, there are 11 cases where the participants reworded the question because the bot failed to return the correct answer because of 1) incorrect extraction of intents and entities 2) missing entities in the posed questions 3) typos in the users questions (e.g., What commits happened between 27/5/208 - 31/5/2018) 4) the bot encountered connection issues to the internet. For example, one of the participants asked the bot "What the details of the commits between May 27th 2018 and May 31st 2018", the bot failed to extract the entity (May 27th 2018 and May 31st 2018) because it was not trained on date format that is specified in the posed question. Then, the user rephrased the question to the bot as follows: "What are the details of the commit between 27/5/2018 - 31/5/2018", which allowed the bot to extract the entity correctly.

That said, measuring the total time needed to perform a certain task may cover the number of interactions metric. One can also argue that the number of interactions is considered as a measure for users satisfaction rather than a speed. Because a large number of interactions to perform a task impacts the satisfactions of the bot users negatively [4,11,37].

In general, the evaluation of software bots varies based on their characteristics and the tasks they can perform. For example, if a bot is proactive (e.g., reminds the user to execute a certain task), then the number of interactions may be invalid as a speed metric since the conversation always comes from the bot. Storey and Zagalsky [58] suggested to measure the bot's efficiency and effectiveness in completing developers' tasks. However, we believe that there are different dimensions that bots' developers and researchers can use to assess the proposed bots regardless of their types such as accuracy and user satisfaction. For example, to measure user satisfaction, the bot can monitor and track the sentiment of a user through the conversation and take different actions based on the current user sentiment. On the other hand, bots can be evaluated based on their intelligence, such as how easily they can adapt to new contexts, their ability to explain the reasoning behind their behavior, and the degree of smoothness of the conversation flow [37]. We believe that bots evaluation is still an active area and we encourage researchers to investigate the various dimensions and measures that the bots community can use to assessdifferent bots.

6.2 Study Implications

Our framework has a number of implications on both, software engineering research and practice.

Implications for Future Research: Our study mainly focused on proposing and evaluating a framework to answer some of the most common developers' questions using software repositories. Based on our results, we find a number of implications for future research. First, there is a need for the MSRBot to support more complex queries, as indicated by the participants' comments. For example, one of the participants stated that the bot is going to help a lot in answering more complex queries using the repositories data. Hence, our study motivates the need to examine more complex questions using data from different types of software repositories. Also, our study shows that there is a need to develop approaches that answers questions dynamically (i.e., removing the need to have predefined questions) using repositories to overcome the limited number of questions supported by the bot.
Second, the retrieved data in the bot's answers and the way it is displayed to the user can be improved. For example, one of the participants recommends providing a suggestion to the user in case the bot was unable to identify the user's intent from the posed question. Another participant indicated that the bot's reply should be short and suggested the use of pagination to enable the user to navigate more easily through the bot's reply. Using pagination when displaying the bot's reply (e.g., commits that happened in the last 3 months) needs careful consideration since questions such as how many records to display on each page, and how can users know in which page a specific record is, need to be considered. Furthermore, it is unconventional to implement the pagination in a chat interface, which might affect user satisfaction. Although there are many studies on software bots, we are not aware of any studies that discuss the best way to represent the bot's reply and kind of information that bots' users expect to see. Our findings motivate the need for such studies.

Practical Implications: A direct implication of our findings is that using the bot simplifies the extraction of useful information from software repositories in different ways. First, it helps project stakeholders that do not have technical skills to easily extract the information from different repositories using the natural language. And, although some developers have the skills to perform such tasks (i.e., mining and analyzing data from software repositories), our framework reduces the time to complete those tasks compared to the baseline (i.e., any tool other than MSRBot). Overall, we believe that our framework supports practitioners at different levels in performing their tasks by lowering the barrier to entry for extracting useful data from software repositories. For example, it helps project managers to track the project progress (based on the closed tickets) and assists developers in their daily tasks.

7 Threats to validity

In this section, we discuss the threats to construct, internal and external validity of our study.

7.1 Construct Validity:

The 12 participants used to evaluate the bot framework may have reported incorrect results, which would impact our findings. However, we are quite confident in the results returned since 1) most of these students have professional software development experience and 2) in most cases, there was a clearly popular answer (i.e., very few outliers). We also interviewed a subset of the participants, and based on our discussions, all participants seemed very competent in evaluating the output of the bot.

We selected different questions from the literature to evaluate the proposed framework which might bias our evaluation by adding the questions that the bot can better answer. However, we mitigate this threat to validity in different ways. First, we selected the list of questions from different studies arbitrarily. Second, in the given tasks, the participants are free to ask any type of questions to the bot (that is related to the task). Lastly, the participants are unaware of the list of questions that the bot can answer or is trained on.

7.2 Internal Validity:

We used Google's Dialogflow to extract the intents and entities from the posed questions. Hence, our results might be impacted by Dialogflow's ability to translate the user's questions. That said, RQ3, which examines the accuracy of the bot showed high accuracy, which makes us confident in the use of Dialogflow. However, using another framework, might lead to different results. In the future, we plan to examine the impact of such frameworks on our bot. Also, Dialogflow's NLU model is trained on examples that were provided and manually labeled (i.e., the intents and entities) by us. The quality and quantity of training data may impact the effectiveness of the NLU. That said, in our evaluation, Dialogflow was able to handle the majority of questions from our users. In the future, we plan to investigate ways that our bot can learn from user's input and automate the intent and entity extraction/training phase.

Another threat to internal validity is that we used the questions identified in the literature to formulate the tasks in our case study. In some cases, the statement provided in the task might bias the participants on how to pose questions to the bot. The reason for providing the statements to the participants is that we want to keep our study manageable and to evaluate the developed bot using the supported types of questions. However, we mitigate this threat by providing the question as a statement. Also, we did not reveal the list of the questions that the bot was trained on to the participants. Albeit anecdotal, we believe that our results show a limited effect of the wording of the tasks on the participants because their questions were syntactically different than the provided statements, although they have similar semantics) (e.g., "how many commits in the last month" and "what are the details of the commits between 27/5/2018 - 31/5/2018?".

7.3 External Validity:

Our study was conducted using Hibernate-ORM and Kafka projects and supported 15 common questions. This means that the study might yield different results when selecting other projects or supporting a different set of questions related to software repositories. These are threats to external validity as they may limit the generalisability of our results. However, we plan to expand our study to support more systems and questions in the future. Also, it is important to note that the point of this paper is to design and evaluate the feasibility of using bots on top of software repositories to automate the answering of commonly asked questions.

Also, we used students to evaluate our framework. Therefore, using different participants (i.e., developers) might affect the generalizability of our results. However, we argue that the main purpose of the case study is to evaluate our approach rather than studying the participants characteristics. Moreover, most of the participants have more than 3 years of industrial experience which reduces the threats to external validity. Furthermore, we plan to conduct a large-scale study in the future by having more developers from the industry.

8 Conclusion & Future Work

Software repositories contain numerous amounts of useful information to enhance software projects. However, not all project stakeholders are able to extract such data from the repositories since it requires technical expertise and time. In this paper, we design and evaluate the feasibility of using bots to support software practitioners in asking questions about their software repositories. Our findings show that the bot provides answers that are considered very useful or useful (as indicated by 90% of the participants), efficient (participants take on median 40 seconds to complete the tasks), and accurately answers questions posed by its users (as indicated by 90.8% of completed tasks). Also, our study highlighted some of the potential pitfalls when applying bots on software repositories. For example, our study finds that more attention is needed regarding the content and length of the information that the bot replies with, how to handle complex user questions, and how to handle user errors such as typos. Overall, our work showed that bots have the potential of playing a critical role by lowering the barrier-to-entry for software stakeholders to extract useful information from their repositories. Also, the bots are efficient when used on large projects with a long history, which saves the developers' time needed to extract the data from the repositories. Finally, we believe that our work encourages other researchers to explore different usages of bots on different types of software repositories, e.g., code review repositories.

In addition to addressing the issues found in the user study (e.g., handling user typos), the results in this paper outline some directions for future work. First, since the entity recognizer and intent extractor components accuracy depend on the training dataset, we will examine different techniques to generate a dataset for those components (e.g., using Stack Overflow topics) to increase their accuracy. We want to compare the performance of different Natural Language Understanding Platforms (e.g., Dialogflow and Rasa) using software engineering datasets to help the bot's community identify the platform that best fits their context. Also, we are planning to support a wider range of repositories such as (e.g., Gerrit Code Review). Moreover, in the current implementation of our framework, we did not provide the users with the ability to configure the bot. However, allowing users to configure the bot may improve the flexibility and adaptability of our framework. In the future, we plan to allow the MSRBot to be configured through users' feedback by asking users for their preferences and suggesting possible configurations. Finally, we plan to evaluate our bot framework using industrial settings to gain more insights into the roles and scenarios in which the bot can be used.

References

1. Amazon lex build conversation bots. `https://aws.amazon.com/lex/`. (Accessed on 03/19/2019).
2. gensim: Topic modelling for humans. `https://radimrehurek.com/gensim/index.html`. (Accessed on 2/13/2019).
3. Git client - glo boards — gitkraken. `https://www.gitkraken.com/`. (Accessed on 03/04/2019).
4. The impact of conversational bots in the customer experience - good rebels. `https://www.goodrebels.com/the-impact-of-conversational-bots-in-the-customer-experience/`. (Accessed on 08/05/2019).
5. Jira client — atlassian marketplace. `https://marketplace.atlassian.com/apps/7070/jira-client?hosting=server&tab=overview`. (Accessed on 03/04/2019).
6. A. Abdellatif, K. Badran, and E. Shihab. ahmad-abdellatif/msrbot: Msrbot framework. `https://github.com/ahmad-abdellatif/MSRBot`. (Accessed on 10/10/2019).
7. A. Abdellatif, K. Badran, and E. Shihab. MSRBot: Using Bots to Answer Questions from Software Repositories. *Empirical Software Engineering*, July 2019.
8. M. P. Acharya, C. Parnin, N. A. Kraft, A. Dagnino, and X. Qu. Code drones. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 785–788, May 2016.
9. T. M. Ahmed, C.-P. Bezemer, T.-H. Chen, A. E. Hassan, and W. Shang. Studying the effectiveness of application performance management (apm) tools for detecting performance regressions for web applications: An experience report. In *Proceedings of the*

13th International Conference on Mining Software Repositories, MSR '16, pages 1–12, New York, NY, USA, 2016. ACM.

10. N. Ali, Y. G. Guhneuc, and G. Antoniol. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *IEEE Transactions on Software Engineering*, 39(5):725–741, May 2013.
11. J. A. Ask, M. Facemire, A. Hogan, and H. B. Conversations. The state of chatbots. *Forrester. com report*, 20, 2016.
12. S. Banerjee and B. Cukic. On the cost of mining very large open source repositories. In *Proceedings of the First International Workshop on BIG Data Software Engineering*, BIGDSE '15, pages 37–43, Piscataway, NJ, USA, 2015. IEEE Press.
13. J. G. Bankier and K. Gleason. Institutional repository software comparison, 2014.
14. A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 125–134, May 2010.
15. A. Begel and T. Zimmermann. Appendix to analyze this! 145 questions for data scientists in software engineering - microsoft research. https://www.microsoft.com/en-us/research/publication/appendix-to-analyze-this-145-questions-for-data-scientists-in-software-engineering. (Accessed on 12/20/2018).
16. A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 12–23, New York, NY, USA, 2014. ACM.
17. I. Beschastnikh, M. F. Lungu, and Y. Zhuang. Accelerating software engineering research adoption with analysis bots. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*, ICSE-NIER '17, pages 35–38, Piscataway, NJ, USA, 2017. IEEE Press.
18. N. C. Bradley, T. Fritz, and R. Holmes. Context-aware conversational developer assistants. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 993–1003, New York, NY, USA, 2018. ACM.
19. C. Brown and C. Parnin. Sorry to bother you: Designing bots for effective recommendations. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 54–58, Piscataway, NJ, USA, 2019. IEEE Press.
20. J. Cerezo, J. Kubelka, R. Robbes, and A. Bergel. Building an expert recommender chatbot. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 59–63, Piscataway, NJ, USA, 2019. IEEE Press.
21. G. Digkas, M. Lungu, A. Chatzigeorgiou, and P. Avgeriou. The evolution of technical debt in the apache ecosystem. In A. Lopes and R. de Lemos, editors, *Software Architecture*, pages 51–66, Cham, 2017. Springer International Publishing.
22. J. documentation. Manage jira cloud issues in the comfort of slack — jirafe. `https://www.jirafe.io/#features`. (Accessed on 07/25/2019).
23. D. Feng, E. Shaw, J. Kim, and E. Hovy. An intelligent discussion-bot for answering student queries in threaded discussions. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, pages 171–177, New York, NY, USA, 2006. ACM.
24. T. Fritz and G. C. Murphy. Using information fragments to answer the questions developers ask. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 175–184, New York, NY, USA, 2010. ACM.
25. Google. Dialogflow. `https://dialogflow.com/`. (Accessed on 01/09/2019).
26. Google. Training — dialogflow. `https://dialogflow.com/docs/training`. (Accessed on 02/16/2019).
27. S. Gottipati, D. Lo, and J. Jiang. Finding relevant answers in software forums. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 323–332, Washington, DC, USA, 2011. IEEE Computer Society.
28. M. Gupta, A. Sureka, and S. Padmanabhuni. Process mining multiple repositories for software defect resolution from control and organizational perspective. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 122–131, New York, NY, USA, 2014. ACM.

29. A. E. Hassan. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*, pages 48–57, Sept 2008.
30. L. Hattori, M. D'Ambros, M. Lanza, and M. Lungu. Answering software evolution questions. *Inf. Softw. Technol.*, 55(4):755–775, Apr. 2013.
31. M. Höst, B. Regnell, and C. Wohlin. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201–214, Nov 2000.
32. IBM. Watson conversation. `https://www.ibm.com/watson/services/conversation/`. (Accessed on 01/09/2019).
33. D. Jurafsky and J. H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.
34. S. Kabinna, C.-P. Bezemer, W. Shang, and A. E. Hassan. Logging library migrations: A case study for the apache software foundation projects. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 154–164, New York, NY, USA, 2016. ACM.
35. F. Khomh, B. Adams, T. Dhaliwal, and Y. Zou. Understanding the impact of rapid releases on software quality. *Empirical Software Engineering*, 20(2):336–373, Apr 2015.
36. R. Kumar, C. Bansal, C. Maddila, N. Sharma, S. Martelock, and R. Bhargava. Building sankie: An ai platform for devops. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 48–53, Piscataway, NJ, USA, 2019. IEEE Press.
37. C. Lebeuf, M. Storey, and A. Zagalsky. Software bots. *IEEE Software*, 35(1):18–23, January/February 2018.
38. X. Liu, S. Zhang, F. Wei, and M. Zhou. Recognizing named entities in tweets. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 359–367, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.
39. C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.
40. C. Matthies, F. Dobrigkeit, and G. Hesse. An additional set of (automated) eyes: Chatbots for agile retrospectives. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 34–37, Piscataway, NJ, USA, 2019. IEEE Press.
41. Microsoft. Luis: Language understanding intelligent service. `https://www.luis.ai/home`. (Accessed on 01/09/2019).
42. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.
43. B. Mohit. Named entity recognition. In I. Zitouni, editor, *Natural Language Processing of Semitic Languages*. Springer, USA, 2014.
44. M. Monperrus. Explainable software bot contributions: Case study of automated bug fixes. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 12–15, Piscataway, NJ, USA, 2019. IEEE Press.
45. M. Monperrus, S. Urli, T. Durieux, M. Martinez, B. Baudry, and L. Seinturier. Repairnator patches programs automatically. *Ubiquity*, 2019(July):2:1–2:12, July 2019.
46. R. Mordinyi and S. Biffl. Exploring traceability links via issues for detailed requirements coverage reports. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 359–366, Sept 2017.
47. A. Murgia, D. Janssens, S. Demeyer, and B. Vasilescu. Among the machines: Human-bot interaction on social q&#38;a websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 1272–1279, New York, NY, USA, 2016. ACM.
48. E. Paikari, J. Choi, S. Kim, S. Baek, M. Kim, S. Lee, C. Han, Y. Kim, K. Ahn, C. Cheong, and A. van der Hoek. A chatbot for conflict detection and resolution. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 29–33. IEEE Press, 2019.

49. L. Ratinov and D. Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
50. M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vsquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong. On-demand developer documentation. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 479–483, Sept 2017.
51. I. Salman, A. T. Misirli, and N. Juristo. Are students representatives of professionals in software engineering experiments? In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 666–676, May 2015.
52. G. R. Sankar, J. Greyling, D. Vogts, and M. C. du Plessis. Models towards a hybrid conversational agent for contact centres. In *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*, SAICSIT '08, pages 200–209, New York, NY, USA, 2008. ACM.
53. A. A. Sawant and A. Bacchelli. fine-grape: fine-grained api usage extractor – an approach and dataset to investigate api usage. *Empirical Software Engineering*, 22(3):1348–1371, Jun 2017.
54. V. S. Sharma, R. Mehra, and V. Kaulgud. What do developers want?: An advisor approach for developer priorities. In *Proceedings of the 10th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '17, pages 78–81, Piscataway, NJ, USA, 2017. IEEE Press.
55. T. Siddiqui and A. Ahmad. Data mining tools and techniques for mining software repositories: A systematic review. In V. B. Aggarwal, V. Bhatnagar, and D. K. Mishra, editors, *Big Data Analytics*, pages 717–726, Singapore, 2018. Springer Singapore.
56. J. Sillito, G. C. Murphy, and K. D. Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4):434–451, July 2008.
57. J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.
58. 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*, FSE 2016, pages 928–931, New York, NY, USA, 2016. ACM.
59. Y. Tian, F. Thung, A. Sharma, and D. Lo. Apibot: Question answering bot for api documentation. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 153–158, Piscataway, NJ, USA, 2017. IEEE Press.
60. E. F. Tjong Kim Sang and F. De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.
61. C. Toxtli, A. Monroy-Hernández, and J. Cranshaw. Understanding chatbot-mediated task management. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 58:1–58:6, New York, NY, USA, 2018. ACM.
62. C. Treude, M. P. Robillard, and B. Dagenais. Extracting development tasks to navigate software documentation. *IEEE Transactions on Software Engineering*, 41(6):565–581, June 2015.
63. 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*, ICSE-SEIP '18, pages 95–104, New York, NY, USA, 2018. ACM.
64. R. van Tonder and C. L. Goues. Towards s/engineer/bot: Principles for program repair bots. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 43–47, Piscataway, NJ, USA, 2019. IEEE Press.

65. M. Vasconcelos, H. Candello, C. Pinhanez, and T. dos Santos. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Brazilian Symposium on Human Factors in Computing Systems*, HFCS '17, 10 2017.
66. M. Wessel, B. M. de Souza, I. Steinmacher, I. S. Wiese, I. Polato, A. P. Chaves, and M. A. Gerosa. The power of bots: Characterizing and understanding bots in oss projects. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW):182:1–182:19, Nov. 2018.
67. M. Wyrich and J. Bogner. Towards an autonomous bot for automatic source code refactoring. In *Proceedings of the 1st International Workshop on Bots in Software Engineering*, BotSE '19, pages 24–28, Piscataway, NJ, USA, 2019. IEEE Press.
68. B. Xu, Z. Xing, X. Xia, and D. Lo. Answerbot: Automated generation of answer summary to developersź technical questions. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 706–716, Piscataway, NJ, USA, 2017. IEEE Press.
69. S. Zamanirad, B. Benatallah, M. Chai Barukh, F. Casati, and C. Rodriguez. Programming bots by synthesizing natural language expressions into api invocations. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 832–837, Piscataway, NJ, USA, 2017. IEEE Press.
70. J. Zamora. I'm sorry, dave, i'm afraid i can't do that: Chatbot perception and expectations. In *Proceedings of the 5th International Conference on Human Agent Interaction*, HAI '17, pages 253–260, New York, NY, USA, 2017. ACM.