Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences

Dušan Okanović
Novatec Consulting GmbH, Germany

Samuel Beck
Lasse Merz
Christoph Zorn
Leonel Merino
André van Hoorn
University of Stuttgart, Germany

Fabian Beck
University of Duisburg-Essen, Germany

ABSTRACT

Even though load testing is an established technique to assess load-related quality properties of software systems, it is applied only seldom and with questionable results. Indeed, configuring, executing, and interpreting results of a load test require high effort and expertise. Since chatbots have shown promising results for interactively supporting complex tasks in various domains (including software engineering), we hypothesize that chatbots can provide developers suitable support for load testing.

In this paper, we present PerformoBot, our chatbot for configuring and running load tests. In a natural language conversation, PerformoBot guides developers through the process of properly specifying the parameters of a load test, which is then automatically executed by PerformoBot using a state-of-the-art load testing tool. After the execution, PerformoBot provides developers a report that answers the respective concern. We report on results of a user study that involved 47 participants, in which we assessed our tool's acceptance and effectiveness. We found that participants in the study, particularly those with a lower level of expertise in performance engineering, had a mostly positive view of PerformoBot.

ACM Reference Format:
Dušan Okanović, Samuel Beck, Lasse Merz, Christoph Zorn, Leonel Merino, André van Hoorn, and Fabian Beck. 2020. Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences. In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3358960.3375792

1 INTRODUCTION

The fulfillment of non-functional requirements is of utmost importance for the success of software and the business that depends on it. The evaluation of non-functional properties of software, such as performance, includes choosing appropriate tools and methods, setting them up, executing them, and collecting performance data. The next step is to make sense of this data, i.e., to analyze it and to find if and where there are performance problems. All of these steps require a significant level of expertise and can successfully be executed usually only with a significant effort. However, performance experts come in short supply and even if they are available, the task of performance analysis can be time-consuming and error prone [13]. Existing approaches that aim to simplify this process [34, 35] are limited to either choosing the underlying approach or result interpretation.

One interesting trend that can be observed in recent years is the increasing use of chatbots, which simulate human-like conversations in order to answer questions, provide information, and trigger other services [11]. Chatbots are becoming increasingly popular also in the domain of software development, where chatbots provide support to developers in their everyday work [2].

The overarching goal of our work is to provide support, especially to those with little to no expert knowledge of the required methods and tools, in executing performance evaluation tasks. We see chatbots as a promising tool for approaching this goal. The particular contribution of this paper is to investigate this hypothesis by presenting our PerformoBot chatbot that supports in a particular performance evaluation task — namely load testing — and by evaluating PerformoBot's effectiveness and usability in a user study.

By interacting with our PerformoBot in natural language, users can specify load test parameters, such as domain, workload, duration, and metric of interest. They receive execution reports tailored exactly to those concerns, while the chatbot does background tasks of configuring the appropriate tools and analyzing the data to extract the information needed to answer the stated concern. To run the performance analysis, PerformoBot creates an actual configuration for a specific (load testing) tool. For generating the reports based on the collected data, we use the Vizard framework [23]. This way, the user receives a clear and concrete answer to the stated concern, together with appropriate visualizations and generated textual explanations.

The user study aimed to investigate (i) how PerformoBot helps developers to create and execute a load test, (ii) how developers perceive PerformoBot and the interaction with it, (iii) what educational effect PerformoBot has on developers, and (iv) how the interaction with PerformoBot depends on the knowledge and experience of developers. In the study, each participant was asked to solve a load testing task using PerformoBot and to answer a questionnaire on the task and the experiences with PerformoBot. The study was conducted as an online study and answered by 47 participants with different levels of expertise. The results suggest

ICPE '20, April 20–24, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-6991-6/20/04…$15.00
https://doi.org/10.1145/3358960.3375792

that PerformoBot supports especially novice users in conducting load tests. Moreover, the survey results guide future directions with respect to extensions.

The rest of the paper is structured as follows. Section 2 provides the background for our work. In Section 3 we present our approach. In Sections 4 and 5 we present the results of the conducted user study. Section 6 presents an overview of related works in this field and in Section 7 we draw conclusions and outline future work. The supplementary material, containing the artifacts from the paper and the results of the survey, is available online [21].

2 BACKGROUND

When analyzing user inputs, chatbots use the concepts of *intents* and *entities* [1]. Within user inputs, chatbots identify intents, i.e., the keywords and phrases a chatbot reacts to, which represent what the user of a chatbot wants (to do, to know, etc). In the training phase, developers of the chatbot define expected phrases that should be mapped to one intent. The phrases are used to train a machine learning model for natural language in order to recognize similar phrases. Entities can be considered parameters for intents and can be mandatory or optional. The difference is that, if any of the mandatory entities is missing, the chatbot will ask the user to specify it. In general, anything can be defined as an entity, be it something simple or a combination of previously defined entities. Afterwards, these entities are manually marked in the training phrases of corresponding intents.

An example of intents and corresponding entities is shown in Figure 1 as an excerpt of a conversation between a user and a chatbot

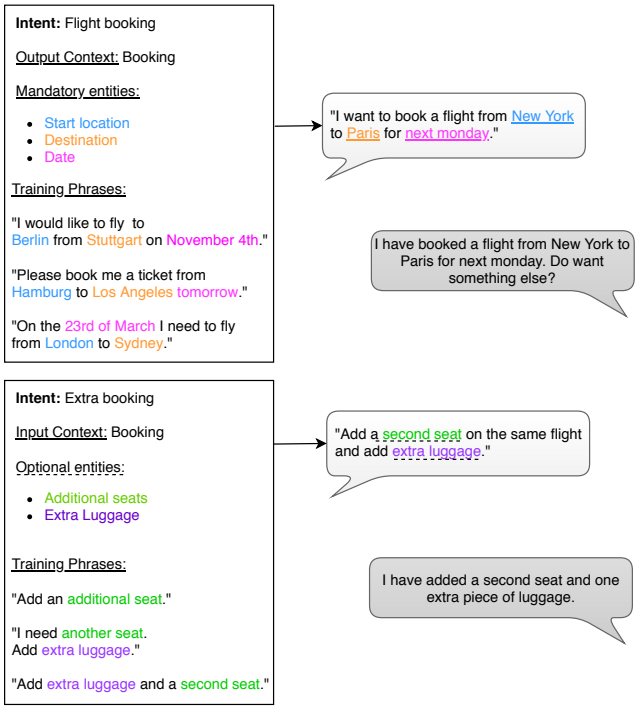


Figure 1: An example of intents and corresponding entities for a chatbot that helps with flight bookings.

that provides help with booking a flight.[1] The sentence "I would like to fly to Berlin from Stuttgart on November 4th" represents one of the training phrases. Based on these training phrases, the chatbot is also able to recognize a similar phrase ("I want to book a flight from New York to Paris for next Monday.") in the actual conversation. Typically, this is an iterative process, where developers add new training phrases as the chatbot is being used, because people always find new ways to interact with the chatbot.

Chatbot frameworks only provide means to implement logic. An additional conversational UI can be developed or they can be integrated with popular communication platforms like Slack, Telegramm, Facebook Messenger, or Skype. Currently, there is a multitude of chatbot frameworks, e.g., Microsoft Bot Framework,[2] Amazon Lex,[3] IBM Watson Assistant,[4] and Dialogflow.[5]

3 PERFORMOBOT

In this section we describe PerformoBot. The modeling of the interaction between PerformoBot and its users is described in Section 3.1. In Section 3.2 we provide details on the implementation.

3.1 Modeling Conversations

PerformoBot is designed as a bot in a *pull mode* [33], i.e., the user is the one who initiates the interaction. The bot has two main tasks: First, it interacts with a user with the goal to specify the load test. Second, it processes the collected data and generates a report to provide explanations, with the help of the Vizard framework [23].

The model shown in Figure 2 presents possible flows of conversation between PerformoBot and different types of users (we are using the BPMN 2.0[6] notation). A typical conversation starts with an exchange of greetings between the user and the chatbot, where the chatbot also provides a brief explanation of what it can do. During the conversation, PerformoBot recognizes the level of experience of the user and guides the conversation further. This was implemented by specifying different training phrases and intents for different types of users.

In the example in Figure 3a, we defined an intent for users without knowledge about load tests using training phrases like "I want to create a load test" or "Help me define a load test". As a result, the phrase "Please perform a load test for me" was recognized by PerformoBot. The chatbot then explains to the user what is needed to set up a load test, i.e., workload, domain, and the metric that will be collected. The collection of the parameters is done using another intent. The parameters are defined as entities and were chosen according to typical load test scenarios, namely: workload, target domain (URL), duration of the test, and metric that will be observed. If some entity is missing from the user input, PerfomoBot will ask the user for a value of that entity. The process is repeated until all entities have their respective value. Users with limited knowledge in the area of performance engineering may need more explanations about what PerformoBot can do and more information on performance evaluation in general. Users with some knowledge

[1] The example is based on the ideas in: https://blog.aimultiple.com/business-chatbot/
[2] https://dev.botframework.com/
[3] https://aws.amazon.com/lex/
[4] https://www.ibm.com/cloud/watson-assistant/
[5] https://dialogflow.com
[6] Business Process Model And Notation, www.omg.org/spec/BPMN/2.0/

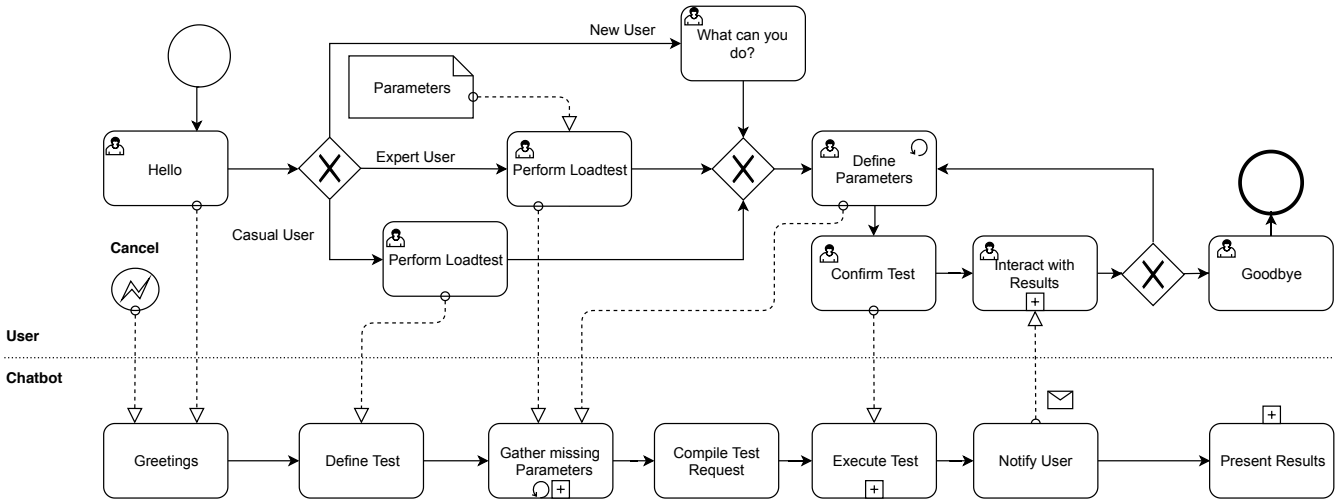


Figure 2: Overview of the interaction between PerformoBot and a user.

(Figure 3b) in the field might be able to directly follow the flow of setting up a load test, probably without requiring explanations on the topic. Lastly, expert users are probably able to skip most parts of the conversation and directly set up a load test with preexisting knowledge of typical parameters (Figure 3c).

At any point in this interaction, the user can ask about explanations of specific keywords, parameters, and metrics before continuing with the task specification, e.g., "What is a load test?", "What is a domain?", "What is latency?". After defining parameters with the help of PerformoBot, users are presented with the current setup of their load test. They can change any of these parameters, reset the whole setup, or let PerformoBot execute the load test. Possible conversation flows for users with different levels of experience are shown in Figure 3. When the user confirms the setup of the test, PerformoBot passes this information to the Vizard framework [23], which executes the test and generates a report that PerformoBot shows to the user.

The user can also interact with the obtained results, setup more load tests, or end the conversation. If anything goes wrong, the user is always able to cancel the conversation, which clears specified parameters, and the conversation can start from the beginning.

To implement PerformoBot we chose Google's chatbot framework Dialogflow. We defined every activity from Figure 2 as an intent in PerformoBot. These intents are specified in the web interface of Dialogflow using the training phrases related to that intent. Using this mechanism, we also configure the chatbot responses that are triggered by a specific user message. This allowed us to adapt responses dependent on what load test parameters the user already defined and to store these parameters over the duration of the conversation. Another possibility would have been to guide the user through a specific order of intents and to not allow her to continue until a specific parameter is defined. However, our design decision allowed for more open interactions, letting the users jump between load test specification and explanations, as well as specify load tests faster by skipping unnecessary questions.

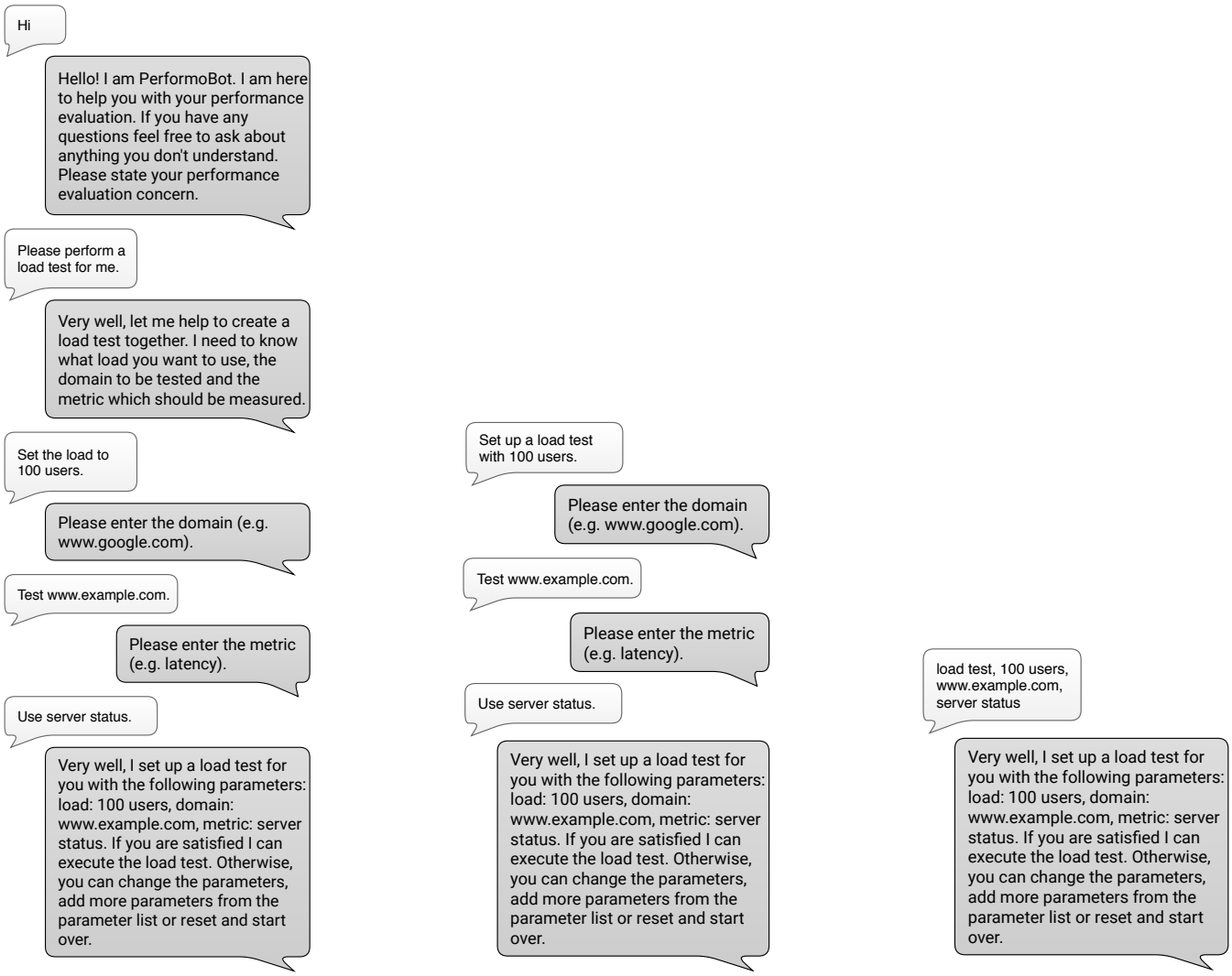
3.2 Implementation of PerformoBot

The main components required for the processing of user requests are shown in Figure 4.

In Dialogflow, we specify the language model used by PerformoBot. Users interact with PerformoBot through the interface of Dialogflow to configure a load test. When a conversation reaches the point where the configuration is finished, i.e., when the bot has collected from the user all the parameters required for the test, the performance experiment is executed, and the user has to wait for the results of the analysis tool. Because Dialogflow is limited to conversations, we had to implement the fulfillment functionality [1] for Dialogflow to connect it to a web service that actually executes load tests. We send the load test configuration to the Vizard framework [22], which executes load tests and sends back execution reports. For the load test execution itself, Vizard uses Apache JMeter.[7] A web service is implemented as a wrapper around Vizard. The service analyzes the data and extracts the information needed to create a response, which is then sent back to Dialogflow to be shown to the user.

In order to avoid that users have to wait for their results, we use the event technology of Dialogflow, allowing Vizard to notify PerformoBot once the performance analysis is finished. This way, the generated report can be shown to the user via a link. An example performance report is depicted in Figure 5. It consist of three different parts. First, the performance concern is shown on the top, followed by a diagram showing the requested metric, a general background description, and a specific answer to the concern. The second part of the report describes the details of the performance experiment. This description contains the used tool, duration, number of samples, and the collected metrics. A description of the analysis tool and further references to it are also provided. The third part of the report covers additional metrics collected during the experiment. Each section of the other metrics contain also a diagram visualizing further data.

[7] https://jmeter.apache.org/



(a) Novice users. **(b) Medium users.** **(c) Advanced users.**

Figure 3: Different flows of conversation between a novice (left), a user with medium experience (middle), and a user with an advanced knowledge with respect to load testing (right).

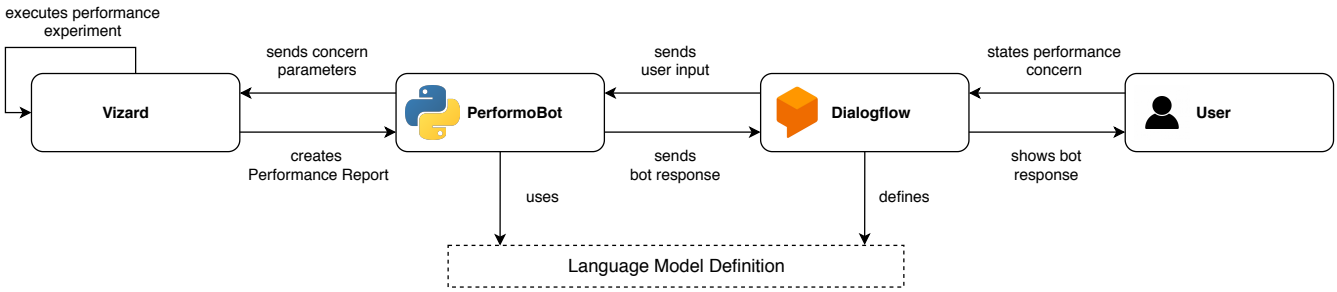


Figure 4: Main components in the system.

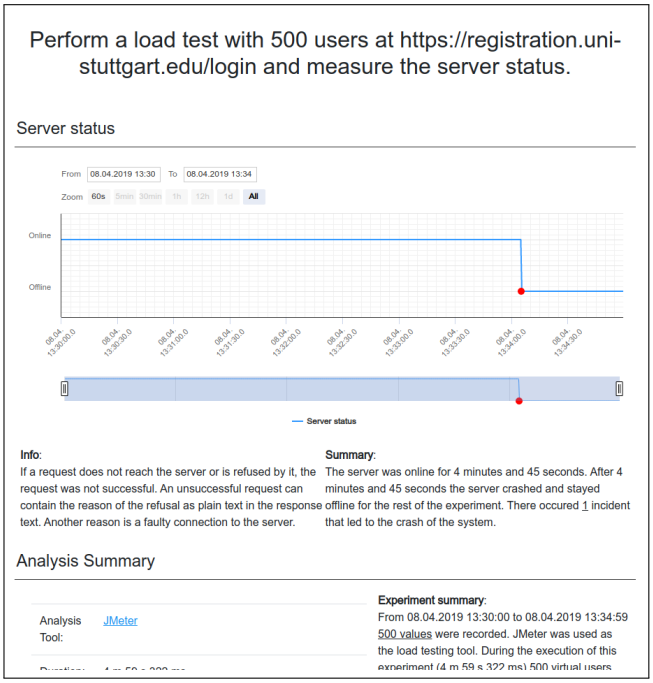


Figure 5: Performance report (excerpt) with visualizations.

4 USER STUDY

To analyze how PerformoBot can support developers in creating and executing a load testing task, we designed a user study. We describe its design, research questions, characteristics of participants, procedure, and data collection.

4.1 Study Design

To investigate the effects of using PerformoBot on user performance, we assess the participants' responses to a task with respect to correctness and completion time. We also measure learning aspects to analyze whether the explanations of PerformoBot have an educational effect on participants. We examine user experience in terms of perceived helpfulness. To analyze how the expertise of participants affects their perception of PerformoBot's support, we invited participants with various levels of expertise in software (performance) engineering. We summarize the goals of our experiment using the template proposed by Wohlin *et al.* [36]:

> Analyze *a chatbot-based approach* for the purpose of *supporting software developers in executing software performance engineering activities* with respect to *acceptance and effectiveness* from the point of view of the *developers*.

We address our goals through the following research questions:

RQ1: How does PerformoBot help participants to create and execute a load test?

RQ2: How do participants perceive PerformoBot and interact with it?

RQ3: What educational effect does PerformoBot have on the participants?

RQ4: How does the interaction with PerformoBot depend on the knowledge and experience of the participants?

To answer these research questions, we designed and conducted an online user study. An online user study allowed us to reach a larger and more heterogeneous set of participants.

Pilot: We ran a pilot study with three participants to detect potential issues and estimate the duration required to complete the task. We invited to the pilot study participants with little experience in performance evaluation. We used the slowest execution time of participants in the pilot to define the duration of the study. We did not find crucial issues with the design of the user study, and therefore, we started the actual user study shortly afterwards.

Tools: We used Google Forms to implement the questionnaire. To link the data of each user from the form and Dialogflow, we created a unique identification number for each participant that was required to start the conversation with the chatbot and to fill in the questionnaire.

4.2 Participants

We designed the study to involve participants of a wide range of levels of experience in software performance engineering, both from academia and industry. To further extend the number of potential participants, we promoted the user study using social media as well as through invitations sent by email to a list of selected software developers.

4.3 Procedure

In the study, participants were asked to create and execute a load testing task. Afterwards, the participants had to fill in a questionnaire to collect their feedback.

Load testing task. We presented a scenario that involves a web system that expects a heavy load in the near future. The participants were asked to investigate whether the system would be able to handle this load. The parameters necessary to complete the task were provided in a description of the scenario that included: *(i)* the expected number of users, *(ii)* the system's domain, and *(iii)* the metric that was to be analyzed. After introducing the scenario, the participants were asked to start a conversation with the chatbot to solve the task. The participants were expected to interact with the chatbot to identify the parameters needed to create and configure a load test. After specifying the load test, the participants could ask the chatbot to execute the load test or modify it further. We limited the time that participants could spend in the task to eight minutes to ensure that the participants concentrated on the task. Once the time limit was reached, the chatbot provided the correct load test configuration, and showed a pre-generated report. Finally, the participants moved on to the study's questionnaire. In the interest of time, we did not run the test, rather we were displaying a pre-generated report of a load test execution for the system in question.

Questionnaire. The participants were asked, based on the given result in the report, to answer the scenario's question, i.e., whether

the system would be able to handle the load. The questionnaire consisted of three sections (as can be seen in [21]).

Firstly, we asked the participants about their background in computer science and their familiarity and knowledge of performance evaluation. Additionally, we asked them whether they have previous experience with chatbots, i.e., how often they have used one and what kinds of chatbots they have interacted with before.

Secondly, we asked the participants whether they were able to complete the task and, if not, to explain why. Next, the participants were asked to answer the task question, namely whether the servers are able to handle the increased load in the described scenario. We opted to provide all participants with the same information, even if they created a different load test. We also asked the participants how much support PerformoBot gave them to solve the task. Specifically, we asked which features were particularly helpful and which ones they felt were missing to complete the task. Additionally, the participants were asked to summarize their impressions using only one adjective. The participants were asked to rate, using a 5-step Likert scale, how natural the conversation felt to them, how understandable the explanations of PerformoBot were, and if they felt they learned something new about performance evaluation based on the conversation with the chatbot.

Lastly, the participants were asked about PerformoBot in general, i.e., whether they would recommend PerformoBot for performance evaluation, and if so, why.

4.4 Data Collection

We collected the following data from the sessions: *(i)* the number of participants who successfully complete the task, which illustrates the overall support that our chatbot can give to developers when performing analysis experiments; *(ii)* the correctness and the time needed by the participants to complete the task, which relate to the effectiveness of PerformoBot on user performance; *(iii)* the interactions with the chatbot, which support a quantitative analysis (e.g., number of sentences written by the participants) as well as a qualitative analysis (e.g., implicit intent of written sentences); *(iv)* time spent with the chatbot, during and after the study; *(v)* the answers from the questionnaire, which enable, for instance, to group expert and beginner participants' responses; *(vi)* the distance of participants' interactions to their actual intent, which measures a potential learning effect of PerformoBot on the participants.

5 RESULTS

The total number of participants was 47, and all of them answered to have a background in computer science. 41 of them (87.2%) stated that they were able to complete the task. 38 participants correctly answered the scenario question in the questionnaire, i.e., whether the service will be able to handle the load or not, three answered wrongly and six answered that they do not know. Of the six participants who were not able to complete the task, three were still able to answer the scenario question correctly based on the displayed report and the other three answered that they do not know. This also means that out of the participants who have answered that they successfully completed the task, three answered wrongly to the scenario question and another three participants answered that they did not know.

The reasons for not completing the task were two-fold. For some participants the available time was too short to learn and to set up the load test, or there was too little guidance about how to correctly use PerformoBot in terms of keywords and parameters. Other participants, who answered that they did not complete the task, reported that this kind of load test is too simple and incomplete to actually answer the scenario.

To execute the load testing task, participants on average used 3.64 minutes of the available eight minutes, with a standard deviation of 1.86 minutes. It has to be noted, however, that these values are not particularly precise, since Dialogflow rounds the recorded times down to minutes, which means if someone took 3 minutes and 40 seconds it got rounded to 3 minutes. The participants sent on average 11.53 messages when interacting with the bot (the median was 10), with a standard deviation of 5.84 coming mainly from 5 participants with over 20 interactions and 4 participants who needed less than 5 messages.

In the following sections we discuss the results of the questionnaire in regard to our research questions.

5.1 RQ1: How does PerformoBot help participants to create and execute a load test?

In order to evaluate in what ways PerformoBot was able to help participants to set up, execute, and analyze a load test, we asked them about their experiences completing the given task.

> *Participants find PerformoBot helpful, but raise potentials for improvement.*

The most helpful reasons were the automatic execution of the load test (10), report creation (3), and the explanations (5). As visualized in Figure 6, the participants rated the helpfulness of PerformoBot on a scale from 1 to 5 with 3.33 on average. The answers were relatively evenly distributed with 4 being the most frequent answer given by 13 participants.

Next, we asked the participants about their experiences with PerformoBot and what features where helpful or missing in the context of completing the task. The feature mentioned as the most helpful was PerformoBot's ability to automatically create and execute the load test at the end. Facilitating this process automatically

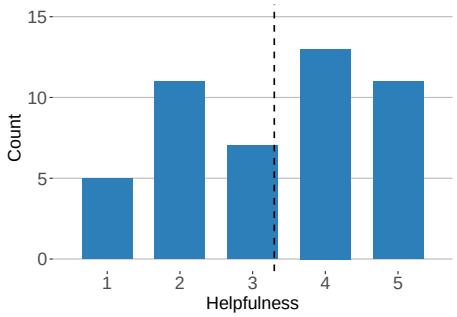


Figure 6: Perceived helpfulness of PerformoBot on a scale from one (useless) to five (very helpful).

was seen as a great step to ease the process of load testing. Furthermore, seven participants specifically mentioned the explanations PerformoBot gave them as a helpful feature. Another five participants commented on the list of available metrics to be a valuable function. Two participants referenced the load test report, the feedback PerformoBot gave them, and the hints about the expected input as beneficial features. One person specifically described not needing any deep knowledge about load testing to use PerformoBot as very helpful. Other participants shared a similar positive notion about the clear description for the conversation flow, the question sequence, and the general ease of use of PerformoBot.

When asked about missing features, the most common group of comments was referring to missing help features. The participants would also have liked to have more advice on available commands, keywords, what a query should look like, and which questions can be asked. Participants suggested examples that could eliminate these problems: emphasizing the keywords the chatbot will react to, suggesting how to correct the wrong input, and clearer instructions. The second most frequent comments were related to the functionality of PerformoBot. Some participants tried to ask more detailed questions about the given scenario and to create more complex load test configurations, to which they did not get satisfying answers. Some examples are questions about the infrastructure, load balancing, and error handling. Additionally, participants missed the possibility of specifying multiple load tests with different parameters. Two participants even suggested to replace PerformoBot with a window with input fields and a simple button. Some individual problems were mentioned, like missing synonyms, a missing list of metrics, and missing load test context.

We also asked the participants to describe their interaction with PerformoBot in one adjective. 26 participants used a positive adjective, 16 a negative and five a neutral one. The most used one was "annoying" (four times), while "fast", "curious", "helpful" and "intuitive" followed with two mentions each.

At the end, we asked the participants for reasons to recommend or not recommend PerformoBot. They would recommend PerformoBot mainly for novice, non-technical users, since they do not need to know deeply about performance evaluation and could learn more in the process. Additionally, most participants would use PerformoBot for its support in setting up, executing, and analyzing load tests. Especially for small and simple problems, it is considered a quick and simple way to perform load testing. One participant suggested to use PerformoBot in other situations, such as in teaching or in games. Some propose to use PerformoBot, together with a built-in interactive documentation, as a 24/7 online remote assistant with which one can communicate even on mobile devices for fast performance evaluation.

On the other side, participants would not recommend PerformoBot for complex performance evaluations. An in-depth analysis of performance metrics and issues like bottlenecks, root cause analysis or stack tracing as well as more complicated use case scenarios are currently not supported by PerformoBot. Some experts stated that they would prefer to analyze these manually with their known tools. Furthermore, participants described the current interaction as too complicated, confusing, and frustrating. One participant expressed this as spending more time trying to learn how to use PerformoBot than actually solving the task. Instead, they would suggest a user interface with form fields and explanatory pictures or guides, as an alternative to PerfomoBot. For the future, participants suggested to provide examples, a list of keywords, and the use of commands inside the chatbot. Additionally, they would like to see features for more performance concerns, like identifying the maximum capacity of a web service or information about the outcomes of different parameter combinations.

5.2 RQ2: How do participants perceive PerformoBot and interact with it?

The Dialogflow history of each participant was analyzed to find common problems regarding their interaction with PerformoBot. In the following, the most frequent answers are highlighted.

> *PerformoBot is lacking explanations and fallback mechanisms to prevent unmatched intents and does not provide enough insight into performance evaluation for expert users.*

Expert users tried to get more information around the scenario, i.e., about the infrastructure, usual workload, and the balancing methods of the observed software system. Example questions of the users were *What is the network bandwidth?* and *How many servers do we have?* Since the bot did not provide intents for these kind of questions, the participants were only given a fallback intent to try another question. The absence of responses to these questions were also prominent negative responses in the questionnaire.

Participants also asked the chatbot about more complex things that were not related to the given scenario, or load testing in general. For example, participants who had more than 15 interactions with the bot challenged it with small talk and questions about computer science, e.g., *Can you calculate 3 + 1?*, *What is your name?*, and *What is a computer?*.

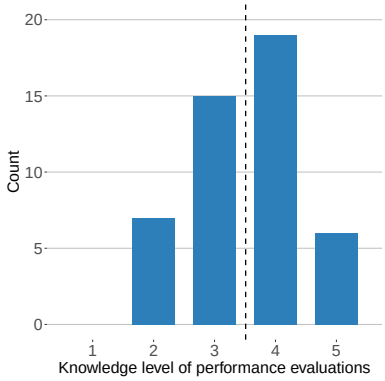
We also discovered that participants whose input could not be matched to an intent tried to modify it repeatedly but used the same input strategy. The most common example of this was when the bot asked for a metric for the analysis: *Please enter a metric (e.g. latency).* Users misunderstood this request and tried to answer with an actual value for the latency and did not try to put in the name of the metric at hand. We also discovered this occurrence in the questionnaire responses of the participants who answered that they got stuck in a conversation with the bot.

5.3 RQ3: What educational effect does PerformoBot have on the participants?

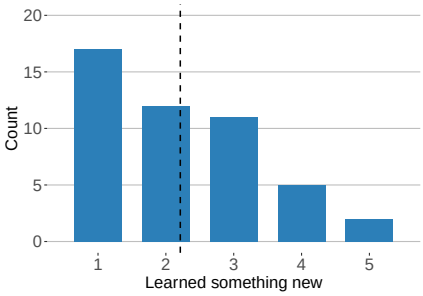
To evaluate educational effects of PerformoBot and whether it allows the users to learn more about performance evaluation, we asked the participants to assess their knowledge of the field, before and after the interaction with the bot.

> *Some educational effect of PerformoBot on the participants could be observed.*

When asked about their knowledge on the topic of the performance evaluation, only two participants stated that they did know what performance evaluation was before the study. Everyone else already had some knowledge and rated it above average on a scale



(a) Self-assessed knowledge of the field of performance evaluation on a scale from one (none) to five (expert).



(b) Perceived educational effect of PerformoBot on a scale from one (learned nothing) to five (learned a lot).

Figure 7: The self-assessed knowledge level and gain of the participants in the field of software performance evaluation.

from 1 to 5, as illustrated in Figure 7a. We asked the participants if they learned something new about performance evaluation through PerformoBot and most of them did not, as shown in Figure 7b. Even the two participants stating they did not know anything beforehand, answered 2 and 3 respectively to this question.

5.4 RQ4: How does the interaction with PerformoBot depend on the knowledge and experience of the participants?

At the beginning of the questionnaire, the participants were asked to estimate their knowledge in the field of software performance evaluation on a scale from one (never heard about it before) to five (expert). Figure 7a illustrates the answers to that question. It can be seen that the majority of participants was quite knowledgeable about software performance evaluations as the most common answer is level four. All participants stated that they at least have heard about the topic. For comparison between inexperienced or novice users and expert users, the participants were split into two categories depending on their knowledge about software performance evaluations. The participants who estimated their expertise below the average of 3.5 were assigned to the novice class (n=22), while the participants whose expertise estimation was above this average were assigned to the expert class (n=25).

> *Participants with less experience in performance evaluation find PerformoBot to be more helpful and have a more positive sentiment towards it than those with more experience.*

Of the 22 participants that were categorized as novice users, 20 (90.9%) were able to complete the task that was given during the study and 18 (81.8%) answered the scenario question correctly. In comparison, 21 of the 25 expert users (84%) were able to complete the task and 20 (80%) answered the scenario question correctly.

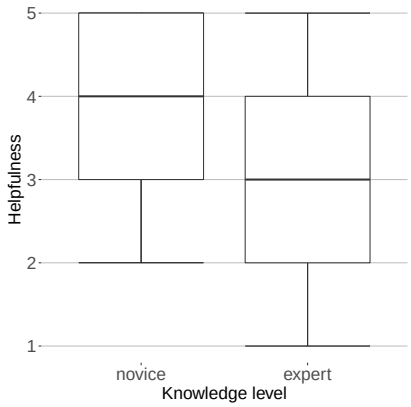
The novice participants who did not finish the task stated that the given time was not enough to get familiar with the bot and that it was unclear which phrases and metrics the bot expects as input. The reasons why expert users did not finish the task are, e.g., that the given time was too short for such a complex task, they expected it to be more complicated than it actually was, or they thought the task was not sufficient for answering the scenario question.

Figure 8a shows how different categories of participants rated the helpfulness of PerformoBot in solving the given task. It can be noted that the novice users perceive the bot as more helpful.

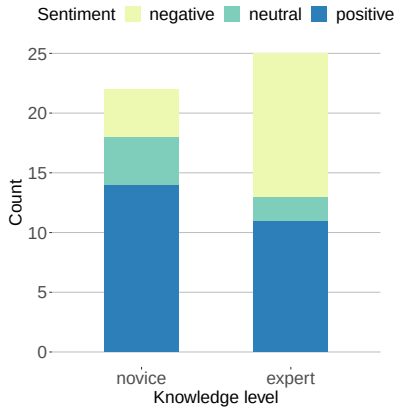
An analysis of the sentiment using the adjectives describing their interaction with PerformoBot as seen in Figure 8b shows that 63.6% of novice participants perceived their interaction as positive, 18.2% perceived it as neutral, and another 18.2% as negative. On the other hand, the sentiment of expert participants on the interaction was divided: 44% of expert users perceived the interaction as positive while 48% as negative, with 8% perceived the interaction as neutral.

5.5 Threats to Validity

In the selection of participants for the user study, we complemented the invitations that we sent to participants highly experienced in performance engineering with additional targeted invitations to less experienced computer scientists and an open invitation on social media. The resulting participants are not a representative sample of all software developers, but the process ensured a desired diversity of expertise levels regarding the subject of the study (see Figure 7a). Hence, we assume a certain generalizability of the results beyond the involved participants. We included in the study a task that is relevant and included in other studies [26]. However, we were not able to cover a wider variety of tasks due to length restrictions of the online experiment (i.e., drop out rates would likely increase when the study takes longer than 20–30 minutes). Also due to the logistics of the study, we had to compromise on the realism of the task introducing some simplifications and working with only few parameters that users had to configure, which some of the expert participants criticized. The anonymity of the online experiment might have also invited participants to test the limits of the system instead of following our instructions (i.e., taking the task not too seriously), but in general, our impression from the analyzed conversation protocols is that this affected only a small number of cases. The study is not comparing our solution against a baseline, which obviously limits the conclusiveness of our findings. However, it was not possible to select a meaningful baseline from



(a) Perceived helpfulness of PerformoBot on a scale from one (useless) to five (very helpful) for different user types.



(b) Sentiment about the interaction with PerformoBot for different user types.

Figure 8: Helpfulness and sentiment of PerformoBot.

existing tools because only expert-oriented tools are available—letting non-expert users try such a tool without extensive prior training would only predictably result in failed tasks, and hence, an unfair comparison.

6 RELATED WORK

The problem of reducing the complexity of performance analysis has been tackled previously. Solutions have been proposed for abstracting the process of problem stating from choosing, configuring, and running an appropriate performance analysis approach [28, 34]. Especially in the field of application performance management [12], automatic analysis of the results is becoming an important requirement for tools that aim to be market leaders. The issue here is that these approaches still require significant prior knowledge in the field, as well as experience to analyze the results. Some works have been proposed for integrating performance-awareness into development environments, e.g., to assess and display the performance (impact) on code changes using model-based and measurement-based approaches (e.g., performance tests and production monitoring) [3, 7, 9, 14]. However, these approaches do not focus on configuring the performance evaluation and do not include the use of conversational interfaces such as chatbots. Generally, also visualizations make performance analysis more accessible; various specialized approaches have been suggested [15], but none for visual reporting of load tests or similar scenarios.

Chatbots (also called *conversational interfaces* or *natural language interfaces*) are gaining popularity in many applications and have already been applied for data analysis. Not a full-fledged conversational interface, but a popular example that answers data-related questions phrased as keywords or a sentence is *Wolfram|Alpha*; it often provides a mix of visualizations, tables, and lists as a reply, but actual conversations are not possible. Also, general-purpose search engines like Google answer more and more queries directly or by providing visualized data-snippets (e.g., for "How many people live in California?"). Srinivasan and Stasko [32] provide a short overview of natural-language interfaces that are combined with visualizations for data analysis—systems mostly focus on specifying a visualization and trigger visualization-related interactions. A recent example that covers both aspects is *FlowSense* [37] where users can both generate visualizations (e.g., "Show a scatterplot of 'mpg' and 'horsepower'.") and control interactions (e.g., "Highlight the selected cars in a parallel coordinates plot."). Such systems are different from our approach that specifies a data-generating test scenario but does not immediately control the visualizations.

Also, software engineering research has started to investigate the application of conversational interfaces in the context of the development process. The *BotSE 2019* workshop (co-locacted with ICSE 2019) [30] discussed the usage of bots (chatbots being a subtype of these) in software engineering. Chatbots have been tested or discussed, for instance, to find experts for a certain code artefact [6], to help avoid potentially conflicting code changes [24], or to support project meetings with background information on the development [16]. Regarding the visualization of software dependencies, chatbots can help to select and filter elements [5]; a natural-language interface (here, also involving speech recognition) is particularly useful when software visualizations are presented in virtual reality, and hence, text input gets harder [29].

In addition to the chat interface, our approach provides the analysis results as a detailed interactive report. It is inspired by previous work that uses *natural language generation* (NLG) [10, 27] to generate documentation of software or to report data analysis results. For instance, there are approaches that textually summarize source code [17, 18, 31], commits and releases [8, 19], or characteristics of executed tests [25]. Some approaches combine the generated texts with interactive visualizations similar to our reports, for instance, to describe runtime information of a method [4] or to summarize different aspects of code quality [20]. In an earlier work-in-progress publication [23], we already sketched a reporting framework (Vizard), which we now integrate with the chatbot interface.

7 CONCLUSION

We investigated whether and how a chatbot-based system, called PerformoBot, can support developers in setting up, running, and

interpreting a load test. Through different flows of conversations, the system allows asking questions and indirectly adapts to the level of expertise of the user regarding performance engineering. After the parameters of the test have been established through the conversation, the system executes the test and presents the results in the form of an interactive report featuring textual and visual descriptions. In a user study with 47 computer scientists having a diverse range of expertise regarding performance evaluation, we could confirm that PerformoBot supports configuring and analyzing a load test: participants generally were able to solve the task, they mostly found the system helpful, and some participants agreed that the system educated them on the topic. Furthermore, we found that such a system might best support less experienced users. For a future wider applicability, the system still needs to be extended with more explanations, fallback mechanisms for unmatched intents, as well as more advanced performance evaluation methods such as different tests or additional configurable parameters.

ACKNOWLEDGMENTS

We would like to thank all participants of the user study. This project is part of the project "Visual Reporting of Performance and Resilience Flaws in Software Systems", supported by Baden-Württemberg Stiftung, and was partially funded by the Deutsche Forschungsgemeinschaft (German Research Foundation; grants Declare—HO 5721/1-1, SoftwareDynamics^2—288909335, and TRR 161—251654672) and by the German Federal Ministry of Education and Research (ContinuITy—01IS17010).

REFERENCES

[1] Dialogflow Concepts. https://cloud.google.com/dialogflow/docs/concepts.
[2] K. B. Ahmad Abdellatif and E. Shihab. A repository of research articles on software bots. http://papers.botse.org.
[3] F. Beck, O. Moseler, S. Diehl, and G. D. Rey. In situ understanding of performance bottlenecks through visually augmented code. In *Proc. International Conference on Program Comprehension*, ICPC, pages 63–72, 2013.
[4] F. Beck, H. A. Siddiqui, A. Bergel, and D. Weiskopf. Method Execution Reports: Generating text and visualization to describe program behavior. In *Proc. IEEE Working Conf. on Soft. Visualization*, VISSOFT, pages 1–10, 2017.
[5] S. Bieliauskas and A. Schreiber. A conversational user interface for software visualization. In *Proc. IEEE Working Conference on Software Visualization*, VISSOFT, pages 139–143. IEEE, 2017.
[6] J. Cerezo, J. Kubelka, R. Robbes, and A. Bergel. Building an expert recommender chatbot. In *Proc. 1st International Workshop on Bots in Software Engineering*, BotSE, pages 59–63. IEEE, 2019.
[7] J. Cito, P. Leitner, M. Rinard, and H. C. Gall. Interactive production performance feedback in the IDE. In *Proc. 41st International Conference on Software Engineering*, ICSE, pages 971–981, 2019.
[8] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk. On automatically generating commit messages via summarization of source code changes. In *Proc. Int. Working Conf. on Source Code Analysis and Manipulation*, SCAM, pages 275–284, 2014.
[9] A. Danciu and H. Krcmar. To what extent does performance awareness support developers in fixing performance bugs? In *Proc. 15th European Workshop on Performance Evaluation*, EPEW, pages 14–29, 2018.
[10] A. Gatt and E. Krahmer. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170, 2018.
[11] L. Goasduff. Chatbots will appeal to modern workers. https://www.gartner.com/smarterwithgartner/chatbots-will-appeal-to-modern-workers/, 2019.
[12] C. Heger, A. van Hoorn, M. Mann, and D. Okanovic. Application performance management: State of the art and challenges for the future. In *Proc. ACM/SPEC International Conference on Performance Engineering*, ICPE, pages 429–432. ACM, 2017.
[13] C. Heger, A. van Hoorn, D. Okanovic, S. Siegl, and A. Wert. Expert-guided automatic diagnosis of performance problems in enterprise applications. In *Proc. 12th European Dependable Computing Conference*, EDCC, pages 185–188. IEEE, 2016.
[14] V. Horký, P. Libic, L. Marek, A. Steinhauser, and P. Tuma. Utilizing performance unit tests to increase performance awareness. In *Proc. 6th ACM/SPEC International Conference on Performance Engineering*, ICPE, pages 289–300, 2015.
[15] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer. State of the Art of Performance Visualization. In *EuroVis - STARs*, 2014.
[16] C. Matthies, F. Dobrigkeit, and G. Hesse. An additional set of (automated) eyes: chatbots for agile retrospectives. In *Proc. 1st International Workshop on Bots in Software Engineering*, BotSE, pages 34–37. IEEE, 2019.
[17] P. W. McBurney and C. McMillan. Automatic documentation generation via source code summarization of method context. In *Proc. Int. Conf. on Program Comprehension*, ICPC, pages 279–290. ACM, 2014.
[18] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker. Automatic generation of natural language summaries for Java classes. In *Proc. IEEE International Conference on Program Comprehension*, ICPC, pages 23–32. IEEE, 2013.
[19] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. Automatic generation of release notes. In *Proc. International Symposium on Foundations of Software Engineering*, FSE, pages 484–495. ACM, 2014.
[20] H. Mumtaz, S. Latif, F. Beck, and D. Weiskopf. Exploranative code quality documents. *IEEE Transactions on Visualization and Computer Graphics*, 26(1), 2020.
[21] D. Okanović, S. Beck, L. Merz, C. Zorn, L. Merino, A. van Hoorn, and F. Beck. Can a Chatbot Support Software Engineers with Load Testing? Approach and Experiences — supplementary material. https://dx.doi.org/10.5281/zenodo.3662711, 2020.
[22] D. Okanović, A. van Hoorn, C. Heger, A. Wert, and S. Siegl. Towards performance tooling interoperability: An open format for representing execution traces. In *Proc. European Workshop on Performance Engineering*, EPEW, pages 94–108, 2016.
[23] D. Okanović, A. van Hoorn, C. Zorn, F. Beck, V. Ferme, and J. Walter. Concern-driven reporting of software performance analysis results. In *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering, ICPE 2019*, pages 1–4, 2019.
[24] 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 *Proc. 1st International Workshop on Bots in Software Engineering*, BotSE, pages 29–33. IEEE, 2019.
[25] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. C. Gall. The impact of test case summaries on bug fixing performance: An empirical investigation. In *Proc. IEEE/ACM 38th International Conference on Software Engineering*, ICSE, pages 547–558. IEEE, 2016.
[26] E. Proko and I. Ninka. Analyzing and testing web application performance. *International Journal of Engineering and Science*, 3(10):47–50, 2013.
[27] E. Reiter, R. Dale, and Z. Feng. *Building natural language generation systems*. MIT Press, 2000.
[28] H. Schulz, D. Okanović, A. van Hoorn, V. Ferme, and C. Pautasso. Behavior-driven load testing using contextual knowledge—approach and experiences. In *Proc. ACM/SPEC International Conference on Performance Engineering*, ICPE, 2019.
[29] P. Seipel, A. Stock, S. Santhanam, A. Baranowski, N. Hochgeschwender, and A. Schreiber. Speak to your software visualization—exploring component-based software architectures in augmented reality with a conversational interface. In *Proc. IEEE Working Conference on Software Visualization*, VISSOFT. IEEE, 2019.
[30] E. Shihab and S. Wagner, editors. *International Workshop on Bots in Software Engineering*, BotSE@ICSE 2019, 2019.
[31] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for Java methods. In *Proc. IEEE/ACM International Conference on Automated Software Engineering*, ASE, pages 43–52, 2010.
[32] A. Srinivasan and J. Stasko. Natural language interfaces for data analysis with visualization: Considering what has and could be asked. In *Proc. Eurographics/IEEE VGTC Conference on Visualization*, EuroVis, pages 55–59. Eurographics, 2017.
[33] M.-A. Storey and A. Zagalsky. Disrupting developer productivity one bot at a time. In *Proc. 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 928–931, 2016.
[34] J. Walter, S. Eismann, J. Grohmann, D. Okanović, and S. Kounev. Tools for declarative performance engineering. In *Proc. ACM/SPEC International Conference on Performance Engineering*, ICPE, pages 53–56, 2018.
[35] J. Walter, A. van Hoorn, and S. Kounev. Automated and adaptable decision support for software performance engineering. In *Proc. 11th EAI International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS*, pages 66–73, 2017.
[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
[37] B. Yu and C. T. Silva. FlowSense: A natural language interface for visual data exploration within a dataflow system. *IEEE Transactions on Visualization and Computer Graphics*, 26(1), 2020.