# Conversational DevBots for Secure Programming: An Empirical Study on SKF Chatbot

Catherine Tony, Mohana Balasubramanian, Nicolás E. Díaz Ferreyra, and Riccardo Scandariato
Hamburg University of Technology
Hamburg, Germany
{catherine.tony,mohana.balasubramanian,nicolas.diaz-ferreyra,riccardo.scandariato}@tuhh.de

## ABSTRACT

Conversational agents or chatbots are widely investigated and used across different fields including healthcare, education, and marketing. Still, the development of chatbots for assisting secure coding practices is in its infancy. In this paper, we present the results of an empirical study on SKF chatbot, a software-development bot (DevBot) designed to answer queries about software security. To the best of our knowledge, SKF chatbot is one of the very few of its kind, thus a representative instance of conversational DevBots aiding secure software development. In this study, we collect and analyse empirical evidence on the effectiveness of SKF chatbot, while assessing the needs and expectations of its users (i.e., software developers). Furthermore, we explore the factors that may hinder the elaboration of more sophisticated conversational security DevBots and identify features for improving the efficiency of state-of-the-art solutions. All in all, our findings provide valuable insights pointing towards the design of more context-aware and personalized conversational DevBots for security engineering.

## CCS CONCEPTS

• **Human-centered computing** → **User studies**; • **Security and privacy** → **Usability in security and privacy**.

## KEYWORDS

DevBot, Software Chatbot, Empirical study, Secure programming

**ACM Reference Format:**
Catherine Tony, Mohana Balasubramanian, Nicolás E. Díaz Ferreyra, and Riccardo Scandariato. 2022. Conversational DevBots for Secure Programming: An Empirical Study on SKF Chatbot. In *The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022), June 13–15, 2022, Gothenburg, Sweden.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3530019.3535307

## 1 INTRODUCTION

Every year hundreds of organizations and companies around the world are negatively affected by severe data breaches and cyber attacks. To a large extent, this is due to a lack of adequate security measures and controls within information systems. Still, secure coding practices in software projects are far from being the norm as developers often lack of proper training, and security is more of an afterthought than an actual priority [16]. In turn, a large number of threats and vulnerabilities emerge as a result of unsavvy decisions and poor security design practices [14].

Recent advances in Artificial Intelligence (AI) in general and machine learning in particular have fostered the emergence of chatbots assisting people's decisions in different fronts and domains (e.g., healthcare, education, and marketing) [2]. Certainly, the software industry has not been the exception to chatbot applications, where conversational agents have also being proposed to support core software engineering tasks (e.g., testing and requirements analysis) [17]. Moreover, such 'DevBots' are slowly becoming popular as well in the field of cyber-security, helping developers to spot security flaws in their code.

Several DevBots have been proposed within the current literature to help in the identification of security flaws during development [10, 12, 17]. These solutions often employ static or dynamic code analysis features for spotting and warning developers about security vulnerabilities. Still, most of these tools do not yet provide *conversational support* to help fixing nor mitigating the bugs they identify [13]. Hence, there is a call for more supportive DevBot solutions that could provide instructional content on how to improve the security of software projects through adequate coding practices.

In this work, we aim to delve into the extent to which current conversational DevBot applications assist the development of secure software systems. For this, we conducted an empirical study on SKF chatbot (SKF stands for Security Knowledge Framework), a conversational agent designed to address developers' security-related questions. Particularly, we aimed at evaluating the extent to which the assistance of SKF chatbot is deemed adequate and useful by its users. To gain insights on these performance aspects we ran a study with 15 participants in which they had to use SKF for removing code vulnerabilities. Overall, the research questions (RQs) addressed in this work are the following:

***RQ1*: How effective are conversational DevBots for secure programming?** To answer this RQ we analysed the extent to which the DevBot helped participants to remove security vulnerabilities. That is, whether it led to full, partial, or no vulnerability fixes, and compared it against the outcome when no DevBot support was provided.

***RQ2*: How useful are conversational DevBots for secure programming?** In this case we developed a questionnaire for capturing the extent to which participants understood the nature of the vulnerabilities they had to fix, and whether they managed to find the right fix for such vulnerabilities. We also assessed whether


*EASE 2022, June 13–15, 2022, Gothenburg, Sweden*

ACM ISBN 978-1-4503-9613-4/22/06…$15.00
https://doi.org/10.1145/3530019.3535307

the DevBot managed to answer all of their questions along with the perceived usefulness of the tool.

***RQ3*: Do developers prefer to use a conversational DevBot when conducting security-related tasks?** To answer this RQ we included some questionnaire items assessing developers' intentions towards a DevBot-aided approach over other problem-solving strategies (e.g., a manual online search).

***RQ4*: What features do developers expect from a conversational DevBot for secure programming?** This RQ aims to identify features that could improve the effectiveness of current conversational security DevBots. For this we included some open-ended questions by the end of our study to elicit the functionalities of an "ideal" chatbot solution.

Our findings suggest that there is still room for improvement in security DevBots, specially when it comes to creating an extensive knowledge base containing reliable information about (i) security vulnerabilities, (ii) fixes, (iii) correct security API usages, and (iv) correct code integration rules. Furthermore, there is a need for more context-aware solutions capable to (i) provide feedback aligned with the individual characteristics of software projects, and (ii) give personalized assistance and support to the end users.

The remaining of this paper is organizes as follows: Section 2 gives a brief overview of the existing conversational and security DevBots as well as some background on SKF chatbot, Section 3 describes our research methodology, the results and their discussion is done in Sections 4 and 5 respectively. We finally conclude the paper and discuss future research directions in Section 6.

## 2 BACKGROUND AND RELATED WORK

In this section we give an overview of the current state-of-the-art in DevBots development. We also introduce the theoretical foundations for our study, namely SKF chatbot and its main characteristics.

### 2.1 Conversational and Security DevBots

In the recent years, bots and their applications have caught the attention of researchers across different software-related disciplines [17]. Specially, there is an increasing interest towards conversational solutions capable of interacting via natural language commands. MSRBot [1] for instance is a conversational DevBot supporting tasks related to the mining of software repositories that can address project-specific questions like *"who modified <file_name>?"* or *"what commits were submitted on <date>?"*. APIBot [19] is another conversational agent capable of answering questions about the adequate usage of APIs, thus releasing developers from the burden of scrutinizing multiple pages of API documentation. DevBots incorporating voice-recognition features can also be found within the current literature. Such is the case of Devy [3], a voice-activated assistant helping software practitioners to perform high-level workflow tasks (e.g., submitting code changes for review). Still, research on voice-controlled DevBots is at a very early stage when compared to chat-based support [17].

Security engineering has not been the exception to DevBots and its applications. Particularly, bug identification, automated testing, and code repair are some of the tasks supported by current security DevBots [17]. For instance, Wyrich and Bogner [21] proposed a refactoring DevBot capable of spotting code smells and fixing simple warnings. Besides, the bot can also send refactored changes to the developers in the form of pull requests. SAW-BOT [18] is another bot helping developers addressing warnings in their code. Like [21], SAW-BOT generates fixes that are later suggested to developers via pull requests. Other security DevBots like Repairnator [20] have been shaped to support code repair tasks and to monitor test failures in continuous integration. Bots detecting risky commits in open-source projects are also matter of ongoing research efforts [17].

### 2.2 SKF Chatbot

Unlike in other software engineering domains, conversational DevBots seem to be underrepresented in the security field. Overall, very few chat-based solutions can be spotted within the current literature, SKF chatbot[1] being one among the most salient ones. SKF was developed as part of OWASP's Secure Knowledge Framework project[2] to enable quick access to information on security vulnerabilities. It contains a knowledge base of common security weaknesses and vulnerabilities along with a set of code examples. Basically SKF can react to 3 type of inquiries on a particular vulnerability, namely (i) *description*, (ii) *solution*, and (iii) *code snippet*. While the first two provide detailed information in natural language about the vulnerability and its prospective solution, the third one corresponds to a code example either in Django, Java, PHP, Flask or Ruby.

SKF chatbot executes three main steps when answering users' queries: (i) intent classification, (ii) entity recognition, and (iii) response generation. In the first step, SKF seeks to understand whether the goal of the user is to get a description, a solution, or a code snippet. For this, it uses a Multinomial Naive Bayes (MNB) classifier to predict the question's intent. Next, it tries to identify the vulnerability for which the user is requesting information. This is done through a keyword-extraction algorithm named Rapid Automatic Keyword Extraction (RAKE), which is domain-independent and helps separating keywords from other words in the question. Finally, SKF generates a response by querying its knowledge base with the intent and the entities extracted from the user's question. Figure 1 illustrates the output generated for *"What is XSS?"*.

```
Enter Question What is XSS?
Description for XSS injection is : Every time the application gets userinput, whether
 this showing it on screen or processing this data in the application background, the
se parameters should be escaped for malicious code in order to prevent crosssite scri
pting injections. When an attacker gains the possibility to perform an XSS injection,
 he is given the opportunity to inject HTML and JavaScript code directly into the app
lication. This could lead to accounts being compromised by stealing session cookies o
r directly affect the operation of the target application.
```

**Figure 1: Terminal version of SKF chatbot**

## 3 RESEARCH METHODOLOGY

Despite being one of the few (for not saying the only) security DevBots with conversational features, SKF's usability has not been yet thoroughly investigated to the best of our knowledge. Hence, we conducted and empirical study to analyse its performance, efficiency, and overall user acceptance. In the following subsections we describe the proposed experimental setting along with the instruments used for SKF's assessment.

[1] https://github.com/Priya997/SKF-Chatbot
[2] https://www.securityknowledgeframework.org/

### 3.1 Experimental Setup

We conducted the evaluation of SKF chatbot using a Java web application containing 2 code-level security vulnerabilities allowing Cross-Site Scripting (XSS) and SQL Injection (SQLI) attacks. While the former consists of placing of malicious SQL code inside webpage inputs, the latter occurs when malicious code is sent through a web application (usually as browser-side scripts). The participants were given the source code of this web application in which the locations of the vulnerabilities were marked in comments. They were asked to fix the specified vulnerable area of the code with and without SKF's help. For this, we created 4 groups, each receiving the tasks and the DevBot support in a different order to reduce the presence of biases in our analysis (Figure 2). For instance, participants in *Group A* had to fix the XSS vulnerability first and then the one corresponding to SQLI. In this case, the support of SKF chatbot was given for the resolution of the second task (i.e., SQLI), whereas for the first one (i.e., XSS) they were only allowed to use Internet search.

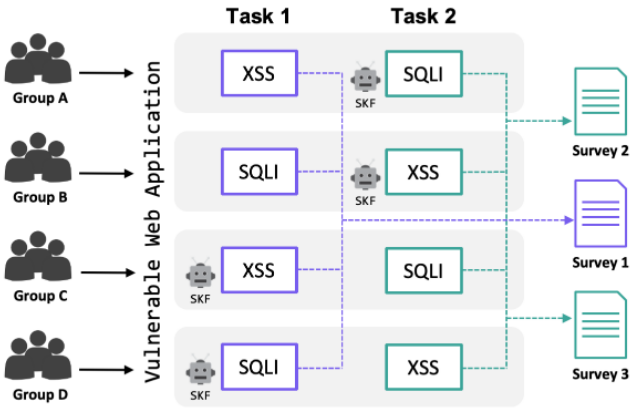


**Figure 2: Workflow of the empirical study**

### 3.2 Survey Instruments

We created 3 survey forms for the experiment containing yes/no, multiple choice, and open-ended questions. Survey 1 was given to the participants after completing the first assignment, whereas Survey 2 and 3 after the second task. Particularly, Surveys 1 and 2 were tailored according to (i) the task the participant had to solve, and (ii) whether she received support of SKF chatbot for its resolution or not. For instance, participants in Group A were asked about the use of online sources for fixing the XSS vulnerability in Survey 1, while in Survey 2 they provided feedback on the use of the conversational DevBot (e.g., their perceived usefulness). Survey 3 remained the same for every group and was given to the participants at the very end of the experiment. Particularly, it contained questions eliciting participants' overall acceptance of the DevBot when solving security vulnerabilities (e.g., if they preferred the DevBot's assistance over a self-conducted Internet search). This third form also included open-ended questions aiming to collect user input that could help shaping additional features for the DevBot. All survey forms and task descriptions were compiled into a **supplementary file** available inside a public repository[3].

## 4 RESULTS

We recruited 15 participants for our experiment (Groups A, B, and C of 4 participants each, and Group D with 3), most of them master students at **Anonymized Institution**. In a preliminary assessment, participants' reported having low to medium knowledge on software security, while their general programming skills averaged in medium to high range which made them suitable candidates for evaluating the performance of SKF chatbot. In the following subsections we report our findings in terms of *Effectiveness*, *Q&A Coverage*, and *Perceived Usefulness*.

### 4.1 Effectiveness

To determine the DevBot's effectiveness, we analysed the number of vulnerabilities successfully fixed by the participants. We considered a vulnerability as "fixed" when the correct patch is applied along with the necessary libraries for executing it. When such libraries are not included, the vulnerability is considered as "partially fixed". A vulnerability is "not fixed" either when no patch is applied or when the wrong solution is implemented.

Based on this criterion, we observed that for both vulnerabilities (i.e., XSS and SQLI) the number of full fixes was higher when no DevBot support was provided (Figure 3). Particularly, 7 out of 15 participants managed to fully solve the given task by conducting an Internet search, but only 3 arrived to the right fix when receiving DevBot support.

When it comes to partial fixes, our results are mixed. As shown in Figure 3-left, there were more participants arriving to a partial solution of the XSS vulnerability with the help of SKF chatbot (4) than using Internet search (1). Conversely, more participants arrived to a partial fix of the SQLI vulnerability without any DevBot support. Particularly, 2 participants arrived to a partial fix with the help of SKF chatbot, whereas 3 did it only by the means of Internet searches (Figure 3-right). The number of unfixed vulnerabilities was for both tasks lower or equal when participants applied a self-conducted Internet search than a DevBot-assisted one.

### 4.2 Q&A Coverage

Another aspect relevant to the DevBot's performance is the amount of questions it manages to answer. Hence, we asked participants whether SKF chatbot responded to all of their questions to understand to which extent it helped in the resolution of the tasks. All in all, 10 out of 15 participants mentioned that SKF chatbot answered all of their questions, while 3 said it only answered them partially. The remaining 2 participants reported not receiving any useful information from the DevBot for addressing the security vulnerabilities.

To further assess SKF's Q&A coverage, we asked participants whether they used other resources (e.g., Internet search) in addition to the DevBot for fixing the vulnerabilities. From 15 participants, 9 reported having used additional resources to complete the given

[3]Repository link: https://collaborating.tuhh.de/e-22/public/skf-chatbot-empirical-study.

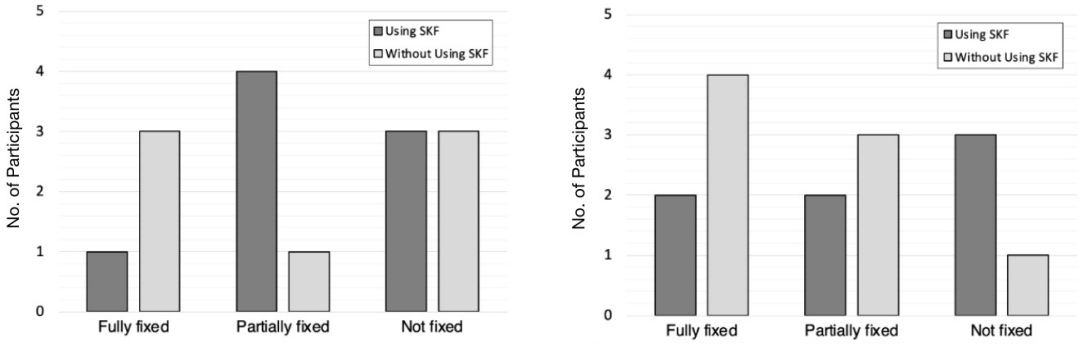


**Figure 3: Participant's performance when fixing the XSS (left) and the SQLI vulnerabilities (right).**

task, whereas 6 mentioned that they did not. Among the 9 participants who used extra resources, 5 of them had reported that the DevBot did answer all of their questions.

### 4.3 Perceived Usefulness

There are two main steps when removing a vulnerability from a software application, namely (i) understanding the nature of the vulnerability, and (ii) finding the correct fix for such a vulnerability. In order to analyse how well SKF chatbot supports this process, we asked participants whether they would prefer a DevBot over an Internet search on each of these stages or not.

As shown in Figure 4, 12 out of 15 participants reported that using SKF chatbot helped them to better understand the vulnerability than simply searching for it on the Internet. On the other hand, opinions were more polarized when it comes to finding the right fix. Particularly, 7 out of 15 respondents said they rather look for the correct fix themselves using the Internet, while 8 opted for a DevBot-supported search. Interestingly, only 3 out of 15 participants managed to find full fixes (i.e., either XSS or SQLI) with the help of SKF chatbot. However, if we consider the the total number of full and partial fixes with DevBot support we obtain 9 out of 15, which is closer to the fixing preferences reported by the participants.

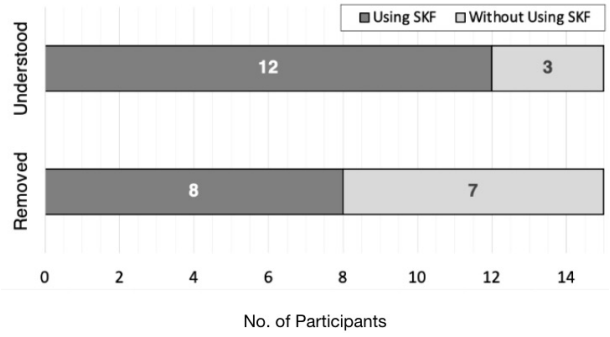


**Figure 4: Participants' preferences regarding understanding and removing security vulnerabilities.**

Last but not least, we asked a subgroup of participants (8) if they would use a conversational DevBot for addressing security vulnerabilities in the future. The available options were *yes*, *no* and *maybe*. Overall, there was only one participant who was willing to receive DevBot support for applying future security fixes, whereas 5 said they might and 2 that they wont.

## 5 DISCUSSION

In this section we discuss the results presented in Section 4 to elaborate on the RQs of our study. This includes aspects related to the performance of SKF chatbot as well as envisaged features for improving chat-based security DevBots. The limitations of our experimental approach are also addressed by the end of this section.

### 5.1 Performance (RQ1, RQ2, and RQ3)

In terms of **effectiveness** (**RQ1**), self-conducted Internet searches seem to outperform the DevBot when it comes to full fixes (i.e., 7/15 against 3/15). Moreover, if we consider full and partial fixes as a whole, the SKF chatbot is still outperformed by the use of the Internet (9/15 against 11/15). After further inspecting the participants' answers and having a follow-up interview with some of them, we identified some drawbacks in the current DevBot approach. Particularly, participants mentioned that SKF chatbot only reacts to specific input, and performs poorly when asked follow-up questions. Such was the case of Participant 12 (P12) who said:

> *P12: to me the bot was not really useful, because I asked different questions and got always the same answer*

Moreover, one participant mentioned it did not provide any information on the external libraries necessary to implement the prescribed fixes. This last point is manifested in the larger number of partial fixes compared to full fixes. Despite of this limitation, some participants considered the DevBot as a good starting point when solving security vulnerabilities. For instance, P4 reported:

> *P4: For someone who was a beginner at such tasks, I needed more examples with explanation. But it was a good point to start. From the points given by the chatbot I understood what exactly I need to search.*

When it comes to **usefulness** (**RQ2**), most of the participants agreed that SKF chatbot helped them to understand the nature of the vulnerabilities better than the sources they found on the Internet (Section 4.3). In part, this could be because the DevBot reduces the time developers spend retrieving information as it provides a full

description of security vulnerabilities, suitable mitigation actions, and code examples. Moreover, unlike code snippets that can be found on the Internet, the ones provided by the chatbot are seen as trustworthy and hence free of additional security vulnerabilities:

> *P9: I was looking into other resources to look for specific Java code examples, i.e. API calls. So for me, it was accompanying material, as the chatbot helped me to nail down the problem itself and give me trustworthy information.*

As mentioned in Section 4.2, 10 out of 15 participants said that the DevBot answered all their questions. However, some limitations in the Natural Language Processing (NLP) features of the bot were spotted by some of them:

> *P9: My question had to match the given question exactly. I ended up with asking for very broad topics and then selecting from the given list of options to find exactly what I was looking for.*

This issue may be associated with the fact that SKF chatbot uses a multinomial naive Bayes algorithm to predict the intent behind the queries. Nowadays there are much more sophisticated and versatile NLP frameworks that can be used for this purpose, and hence to improve the conversational capabilities of security DevBots. For instance, BERT [6] is a novel language representation model which has shown promising results in question-answering and language interface applications. Another suitable approach is the so called Generative Pre-trained Transformer - 3 (GPT-3) [4] which uses deep learning to produce human-like text. Such NLP frameworks combined with a larger knowledge base can help shaping more advanced and efficient conversational security DevBots.

As shown in Section 4.3, 6 out of 8 participants were inclined towards a DevBot-aided approach for addressing future security vulnerabilities. Such **intentions** (**RQ3**) can be associated with the clarity with which information is presented by the DevBot along with its reliability, accessibility, and promptness:

> *P11: The answers (of the DevBot) matched precisely the given problems. In addition the answers were correct, while answers on the Internet are sometimes incoherent and misleading. In a way the Chatbot was a trusted resource, but for explicit code examples I would nevertheless search the internet.*

Nonetheless, the DevBot adoption can be hindered due to conversational limitations (as mentioned earlier) and an unpleasant user interface (P11). Even though participants did not make explicit reference to a lack of adaptability to project-specific scenarios, it can also lessen the DevBot's usability to a large extent.

## 5.2 Envisaged Features (RQ4)

Participants also gave their feedback about which **additional functionalities** (**RQ4**) would be beneficial for a conversational DevBot addressing security flaws in software projects. Besides from improvements in the UI and conversational features of the bot, they suggested adding security indicators to the prescribed solutions:

> *P8: Maybe give recommendations like: this solution helps in 90% of all cases.*
>
> *P11: Maybe giving it some code lines and an estimation of how secure it is? Perhaps even suggesting possible vulnerabilities.*

Sorting and displaying the DevBots' prescriptive solutions according to their security level could improve their navigability, and hence ease their selection. Thereby, developers would become aware of the drawbacks/limitations a particular solution may entail (if any). Supporting examples in other programming languages such as Django and Ruby was another feature pointed by one of the participants (P14). A greater support for integrating solutions into specific code projects was also highlighted as a missing functionality of the DevBot (P12).

Because of our experimental approach, participants did not need to spot the vulnerabilities themselves inside the provided code snippets. Instead, vulnerabilities were given as input so they could conduct the fixes in a controlled experimental fashion. Nevertheless, supporting the detection and identification of security vulnerabilities in software projects should be a key feature of DevBot solutions. Moreover, according to Erlenhov et al. [7], an "ideal DevBot" is defined as:

> *... an artificial software developer which is autonomous, adaptive, and has technical as well as social competence.*

Hence, conversational security DevBots should also be capable of mimicking the social and technical skills of software developers to create a smooth interaction with their users.

In a nutshell, our findings suggest that conversational DevBots for software security should incorporate the following envisaged features to their design:

(1) Strong integration with the development environment.
(2) Automatic code analysis.
(3) Automatic vulnerability detection.
(4) Enhanced natural language interaction.
(5) Context-aware recommendations.
(6) Automatic code repair.

Many of these features require extensive research efforts and interdisciplinary work. Current advances in machine learning, human-computer interaction, and natural language processing can certainly support this quest. Still, a stronger synergy across these disciplines will be necessary to successfully integrate their individual findings into the design of conversational DevBots for security engineering.

## 5.3 Limitations

Although this study has yielded interesting results, there are certain limitations that should be acknowledged. First of all, the size of the sample is relatively small (15) and hence the results may not be generalizable to a larger population. We could also not thoroughly investigate the existence of potential relations between the variables involved in the study (e.g: knowledge level of the users) and the perceived usefulness due to the small sample. Moreover, the study subjects were mostly students (with programming experience) which could also hinder the validity of our findings. Nevertheless, the use of students as participants remains a valid and effective approach for recreating real software engineering settings in laboratory contexts [8]. Therefore, we have analysed and interpreted the results of our study taking into consideration the limitations given by the sample composition and size.

Another limitation point is related to the survey instruments employed throughout the study. Particularly, some of the questions included in the survey forms may not fully capture the aspects they intended to. For instance, capturing participants' perceived usefulness may require scales containing several survey items. Hence, the outcome of this study should be seen as a first attempt in assessing the performance of conversational security DevBots, and not as ultimate conclusions. In the future we plan to extend our analysis by the means of validated scales and constructs like Technology Acceptance Model [5], in order to improve the value of our results.

## 6 CONCLUSIONS AND FUTURE WORK

Supporting developers in their programming practices is of utmost importance for the generation of secure software solutions. Recent advances in AI have contributed to the emergence of DevBots aiding security-related tasks. Still, conversational approaches are in their infancy and thus demand further research and scientific insights. In this work, we have shed some light on the limitations and challenges in the development of conversational DevBots for security engineering. Overall, our findings suggest that these technologies are seen as a reliable source of security knowledge. However, they should incorporate features such as context-aware recommendations and automatic code analysis to increase their levels of user acceptance. Particularly, users would largely benefit from a conversational DevBot capable of providing context-specific answers and code examples that can be easily integrated into existing software projects.

A security DevBot of such characteristics would require an extensive Knowledge Base (KB) comprising data on software security vulnerabilities, mitigation methods, code examples, and library dependencies, among others. Nonetheless, curating such a KB by hand can be a tedious task to carry out. Moreover, technical solutions leveraging manually-curated KBs (e.g., CogniCryptGen [11]) cover just a small number of security use cases. Alternatively, it could be feasible to elaborate richer KBs by leveraging data from open-source projects available online. GitHub Copilot [15] is a commercial tool that employs this approach. However, a recent study shows that 40% of the code generated by GitHub Copilot may contain security vulnerabilities [15]. In the future, we will investigate alternative strategies for constructing KBs suitable for conversational security DevBots, and thereby overcome some of the limitations and challenges presented in this paper. Recently, Fischer et al. [9] released a large dataset consisting of security labelled data-code examples extracted from Stack Overflow[4]. In future publications we will assess whether such a dataset can be leveraged for expanding the knowledge of security DevBot solutions.

## ACKNOWLEDGMENTS

This work was partly funded by the European Union's Horizon 2020 programme under grant agreement No. 952647 (AssureMOSS).

## REFERENCES

[1] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. 2020. MSRBot: Using bots to answer questions from software repositories. *Empir. Softw. Eng.* 25, 3 (2020), 1834–1863.

[2] Eleni Adamopoulou and Lefteris Moussiades. 2020. An overview of chatbot technology. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 373–383.

[3] Nick C. Bradley, Thomas Fritz, and Reid Holmes. 2018. Context-aware conversational developer assistants. In *International Conference on Software Engineering, ICSE*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 993–1003.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). https://arxiv.org/abs/2005.14165

[5] Fred Davis. 1985. A Technology Acceptance Model for Empirically Testing New End-User Information Systems. (01 1985).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186.

[7] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. 2019. Current and future bots in software development. In *BotSE@ICSE 2019 May 28, 2019*, Emad Shihab and Stefan Wagner (Eds.). IEEE / ACM, 7–11.

[8] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. 2018. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering* 23, 1 (2018), 452–489.

[9] Felix Fischer, Huang Xiao, Ching-Yu Kao, Yannick Stachelscheid, Benjamin Johnson, Danial Razar, Paul Fawkesley, Nat Buckley, Konstantin Böttinger, Paul Muntean, et al. 2019. Stack overflow considered helpful! deep learning security nudges towards stronger cryptography. In *28th USENIX Security Symposium (USENIX Security 19)*. 339–356.

[10] Anil Koyuncu, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2020. FlexiRepair: Transparent Program Repair with Generic Patches. *CoRR* abs/2011.13280 (2020).

[11] Stefan Krüger, Karim Ali, and Eric Bodden. 2020. CogniCrypt$_{GEN}$: generating code for the secure usage of crypto APIs. In *CGO '20: 18th ACM/IEEE International Symposium on Code Generation and Optimization*. ACM, 185–198.

[12] Martin Monperrus, Simon Urli, Thomas Durieux, Martin Martinez, Benoit Baudry, and Lionel Seinturier. 2019. Repairnator patches programs automatically. *CoRR* abs/1910.06247 (2019).

[13] Fitzroy Nembhard, Marco M. Carvalho, and Thomas C. Eskridge. 2019. Towards the application of recommender systems to secure coding. *EURASIP J. Inf. Secur.* 2019 (2019), 9.

[14] Lotfi Ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, and Achim D. Brucker. 2017. Time for Addressing Software Security Issues: Prediction Models and Impacting Factors. *Data Sci. Eng.* 2, 2 (2017), 107–124.

[15] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. An Empirical Cybersecurity Evaluation of GitHub Copilot's Code Contributions. *CoRR* abs/2108.09293 (2021).

[16] Kalle Rindell, Karin Bernsmed, and Martin Gilje Jaatun. 2019. Managing security in software: Or: How i learned to stop worrying and manage the security technical debt. In *ARES '19*. ACM, UK, 1–8.

[17] Sivasurya Santhanam, Tobias Hecking, Andreas Schreiber, and Stefan Wagner. 2022. Bots in software engineering: a systematic mapping study. *PeerJ Computer Science* 8 (02 2022), e866.

[18] Dragos Serban, Bart Golsteijn, Ralph Holdorp, and Alexander Serebrenik. 2021. SAW-BOT: Proposing Fixes for Static Analysis Warnings with GitHub Suggestions. In *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 26–30.

[19] Yuan Tian, Ferdian Thung, Abhishek Sharma, and David Lo. 2017. APIBot: question answering bot for API documentation. In *IEEE/ACM International Conference on Automated Software Engineering, ASE*, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 153–158.

[20] Simon Urli, Zhongxing Yu, Lionel Seinturier, and Martin Monperrus. 2018. How to design a program repair bot?: insights from the repairnator project. In *International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2018*, Frances Paulisch and Jan Bosch (Eds.). ACM, 95–104.

[21] Marvin Wyrich and Justus Bogner. 2019. Towards an autonomous bot for automatic source code refactoring. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 24–28.

[4] https://stackoverflow.com/