CHALLENGES AND OPPORTUNITIES IN INTEGRATING INTERACTIVE CHATBOTS INTO CODE REVIEW EXERCISES: A PILOT CASE STUDY

J.C. Farah[1], B. Spaenlehauer[1], K. Bergram[2], A. Holzer[2], D. Gillet[1]

[1]École Polytechnique Fédérale de Lausanne (SWITZERLAND)
[2]Université de Neuchâtel (SWITZERLAND)

Abstract

While chatbots are commonly used to support software developers with repetitive tasks, their use in programming education is still limited, particularly for students in non-technical domains. To better understand the potential for chatbots to support programming education, we conducted a pilot case study aimed at integrating interactive chatbots into an online lesson on programming best practices. Using an educational application simulating the code review process, students were introduced to Python code style guidelines as a part of their practical work for a course on computational thinking aimed at non-technical graduate students. Our mixed-method analysis sheds light on the opportunities and challenges affecting the implementation of our chatbot integration and its potential for adoption in programming education. While we identified opportunities to guide students through predefined conversational pathways, provide students with personalized content, allow students to request support from the instructor, and incorporate humor into the learning process, we also encountered challenges involving configuration, repeated exposures, and limited scripts. These findings have implications that affect the design considerations of the technological and pedagogical aspects of our chatbot integration, which are relevant to educators and researchers in digital education looking to incorporate similar chatbot technologies into their practice. Strategies for harnessing these opportunities and addressing these challenges in future work are discussed.

Keywords: chatbots, programming education, code review, computational thinking, best practices, digital education

1 INTRODUCTION

Conversational agents—commonly referred to as *chatbots*—have been shown to have the potential to support educators with a wide range of tasks, including answering frequently asked questions [1], encouraging learning [2], and providing personalized tutoring [3]. Indeed, recent surveys have shown that chatbots have been incorporated into applications aimed at several educational domains, spanning science, technology, engineering, and mathematics (STEM), as well as languages, business, and the arts [4]. However, while task-oriented chatbots play an important role in software engineering [5] and are popular on social coding platforms [6], research focusing on their use in programming education is limited.

Nevertheless, recent work has highlighted the potential chatbots have to support programming education. Binkis et al. implemented a rule-based chatbot, which they used in a software engineering course for undergraduate students [7]. A usability assessment of their chatbot using the System Usability Scale (SUS) resulted in an acceptable usability score. Laiq and Dieste built and evaluated a chatbot aimed at teaching software engineering students how to perform requirements elicitation interviews, achieving promising results [8]. Ismail and Ade-Ibijola proposed a chatbot to support novice programmers and conducted an evaluation with students, who reported the tool as useful [9]. More recently, Winkler et al. designed *Sara*, a chatbot to scaffold online video lectures, which they evaluated using introductory Python programming tutorials [10]. The potential for chatbots to support programming education is best summarized by Hobert, who developed *Coding Tutor*, a chatbot to support university students in introductory programming courses, which was also positively rated by students [11]. Referring to *Coding Tutor*, Hobert indicated that “such a conversational intelligent programming tutor is suited to take over tasks of teaching assistants in times when no human teaching assistant or lecturer is available for help” [11].

One application of chatbots to programming education that has not been addressed in the literature relates to their ability to support code review exercises, even though the code review process has long been championed for software engineering education [12]. Our work aims to address this gap. In a previous study, we built and evaluated a code review application (CRA) that could be embedded in *code review notebooks*, which we defined as online lessons resembling computational notebooks, but focused on reviewing and annotating code [13]. Results from our evaluation showed that students found the CRA to have positive usability. We then enhanced this application to support chatbot identities and conducted a follow-up Wizard of Oz study that used these chatbot identities as a way to illustrate the code review process to students [14]. Instructors could impersonate chatbots to annotate code snippets with sample comments, but these annotations were static—meaning that students could not engage in real-time conversations with the chatbot—and required manual intervention from the instructor.

In this study, we enhanced the static interface by equipping it with functionalities supporting chatbots following automated, interactive scripts. We then configured our CRA to include *Code Style Bot*—a chatbot designed to support a series of online lessons on Python programming best practices—and conducted a pilot case study aimed at identifying the challenges faced and opportunities provided by our chatbot integration. Our goal was to build on these findings to refine the CRA and the code review notebook format. While the challenges and opportunities reported were identified within our particular pedagogical and technological contexts, our findings could be applicable to similar chatbot technologies and serve educators and researchers in digital education looking to incorporate these technologies into their practice.

2 METHODOLOGY

We conducted our pilot case study in the context of a computational thinking course aimed at students pursuing a master's degree in the Faculty of Economics and Business at the University of Neuchâtel in Switzerland. A total of 28 students registered for the course and 27 completed it. While the course covered both theoretical concepts related to computational thinking and programming concepts using Python (e.g., lists, classes, and dictionaries) this paper focuses on the code style exercises, which featured the CRA that hosted the interactive chatbot used in our study.

2.1 Context

In this section, we outline the pedagogical and technological contexts of our study.

2.1.1 Pedagogical Context

The course took place over one week. Each day, students attended an in-class lecture where they were introduced to a Python programming concept. The same day, they had a lab covering the topics studied in the lecture as well as an exercise related to Python code style guidelines. Students could complete these labs in person or remotely. Table 1 summarizes the lectures and corresponding Python and code style exercises addressed in the course.

Table 1. Lecture topics alongside their corresponding Python lab and code style exercise.

Lecture	Python Lab	Code Style Exercise
Basics	Basics I	Pre-Test
Lists	Basics II	Layout
Functions	Lists	Coding Standards
Dictionaries	Functions	Naming Standards
Classes	Dictionaries	Post-Test
Data	—	—

Students were graded on their participation in the Python labs and on an individual assignment where they were required to design and implement a simple chatbot using the Python concepts covered in class. The individual assignment consisted not only of the code of the chatbot but also of an explanation of the design approach using computational thinking concepts. Students were expected to describe the problems they faced and how they broke these into more manageable parts, using abstractions, and identifying patterns to solve the problem through an elegant algorithmic solution. The Python labs consisted of an overall series of 75 questions that required students to implement some Python code.

Exercises specifically addressing Python code style were included in each lab. In the first lab, students were introduced to the CRA and asked if they could identify the code style issues that were present in a short snippet of code. This exercise served as a pre-test to gauge students' grasp of code style guidelines at the beginning of the course. Over the following three labs, students were then introduced to code style guidelines covering code layout, coding standards, and naming standards. In each lab, the CRA was used to expose the student to code snippets containing a particular code style issue and explain how to fix it (see Fig. 1). In the second lab, these explanations were included in the text surrounding the CRA, while in the third and fourth labs, the explanations were presented by *Code Style Bot*. The final lab contained a second exercise asking students to identify issues using the CRA. This exercise served as a post-test and was followed by an explanation of the issues present in the exercise, which was once more explained using *Code Style Bot*.

The labs were self-corrected. When students executed their code, they would get a submission key, which they could enter into an input box for verification. The labs followed a gamification approach and displayed an individual point counter for students, who would get three points for a correct answer on the first attempt, and two points for the second or third try. They could also ask for hints, which would remove one point from their total. Students were expected to complete 75% of the labs to get the full participation points. Both the individual assignment and the labs were due one month after the course. As COVID-19 restrictions were still in place, the course was recorded and students who attended the course were required to wear masks. As such, the course could be followed remotely, which is what most students did. Approximately 7 to 10 students were physically present in class.

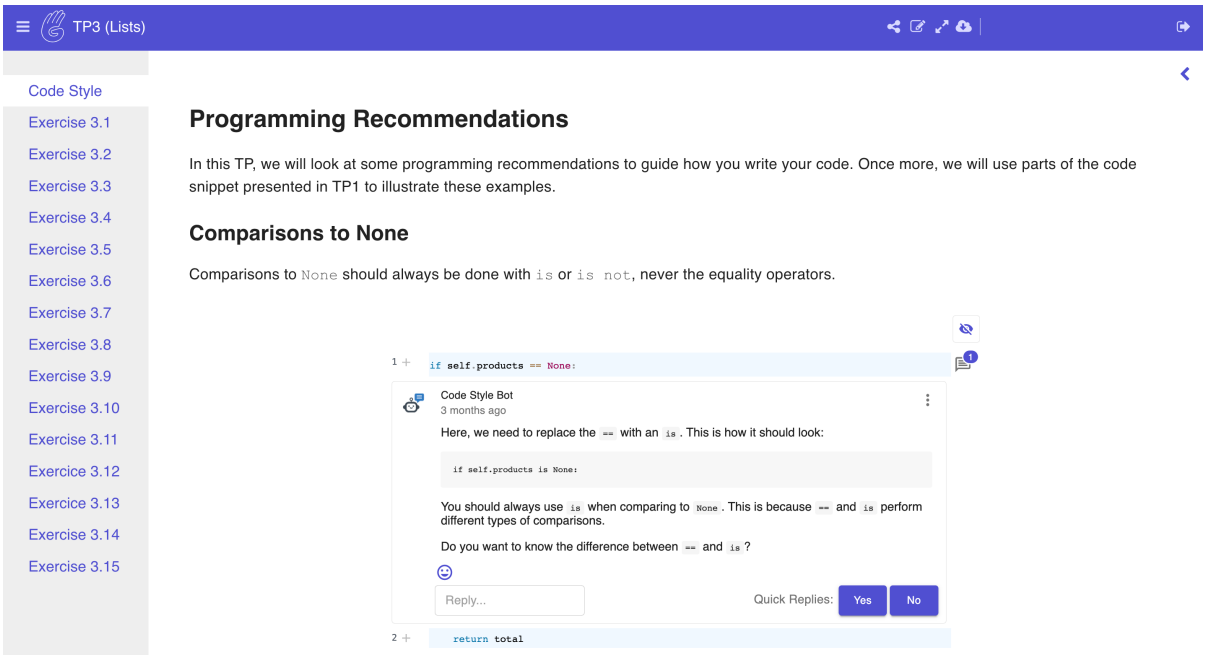

Figure 1. Our chatbot was integrated into a code review application that was embedded in an online learning capsule resembling a computational notebook aimed at teaching Python programming.

2.1.2 Technological Context

As previously noted, the focus of this paper is on the code style exercises, which featured our CRA equipped with an interactive chatbot. The CRA was embedded into five different online learning capsules corresponding to the five Python labs enumerated in Table 1. These capsules were built and disseminated using the Graasp online education platform [15]. Each learning capsule consisted of a series of phases covering exercises related to the corresponding lecture. The first of these phases

featured a code style exercise, as described in Section 2.1.1, while the first and last code style exercises corresponded to a pre-test and post-test exercise. Fig. 1 depicts the online learning capsule corresponding to the Python lab on lists and highlights the first phase, which features the code style exercise on coding standards.

To enable interactive dialog within a CRA, the instructor must configure the CRA so that it includes a chatbot that follows a rule-based script. A script defines how a chatbot will process student input. The CRA includes a dialog engine that can interpret simple JSON configuration files following a predefined structure (see Fig. 2, left). For clarity, we refer to chatbot messages as *comments* and student messages as *responses* or *replies*. Each script starts with a comment that serves as a *seed*, which is a comment used to elicit an interaction with the student. Depending on the student's response, different conversational paths can be taken. The dialog engine matches a student's response to the appropriate next step in the conversational path using regular expressions. For convenience, a subset of the responses that the chatbot would know how to interpret are provided as quick-reply buttons (e.g., the *Yes* and *No* buttons shown in Fig. 1). There is also a *fallback* comment that is used when no comment could be matched with the response provided by the student. Finally, to mark the end of an interaction, the chatbot outputs a comment that does not provide any quick-reply options. If the student responds to this last comment, a special *end* step is used, which informs the student that the chatbot has nothing more to say and provides the student with a button to restart the interaction.

For this case study, we equipped the CRA with *Code Style Bot*. This chatbot was used to annotate the lines of code that contained potential code style issues within a series of Python code snippets used in the third, fourth, and fifth labs. The chatbot was configured to engage students by asking them if they understood and agreed with the logic behind the code style issue at hand, provide explanations of Python programming best practices, and motivate the reasons behind those best practices.

2.2 Data Analysis

We obtained data from the Graasp platform related to both the configuration and use of the code review application as well as the interactions with the chatbot. To identify challenges and opportunities, we performed a mixed-method analysis of (i) quantitative data consisting of the number of interactions a student had with the chatbot and Likert-scale ratings of the chatbot, and (ii) qualitative data consisting of the transcripts of the interactions between the chatbot and the student, the document used to configure the chatbot script, as well as feedback from students in the form of open-ended comments. Where applicable, we report counts, sample means ($\overline{x}$), medians ($\tilde{x}$), and standard deviations (s). To guide our approach, we focused on two aspects of our chatbot integration: (i) the *configuration* of the chatbot and (ii) the *interaction* between the chatbot and the students.

3 RESULTS

In this section, we present the main challenges and opportunities that we identified during the pilot study, focusing on the two aspects tackled by our approach.

3.1 Challenges

We identified three challenges related to the configuration of the chatbot and two challenges that surfaced during the students' interactions with the chatbot. We present these challenges below.

3.1.1 Configuration

The first challenge when configuring a chatbot interaction for the CRA surfaced when writing the script for the chatbot. To support interactivity—as explained in Section 2.1.2—each CRA requires a chatbot to be configured to follow a predefined script in JSON format. Fig. 2 shows a side-by-side view of the JSON script and its representation as a flow diagram. This visualization is handy, but it can be hard to mentally convert the linear structure of the script into a graph of interactions (and vice-versa), especially when there exist multiple paths with variations and a node can be reused multiple times in different places. Given the structure of the scripts presented in Section 2.1.2, very complex interactions are hard to create. In fact, configuring the flow of these rule-based scripts took repeated trial-and-error iterations that involved the course instructors and the software engineers in charge of developing the chatbot

integration. This process consisted of a chatbot script being configured and deployed to the online learning capsule, where it could be tested. Feedback would then be gathered and used to refine the script. We refer to this first configuration challenge as *complicated configuration* (CC1).

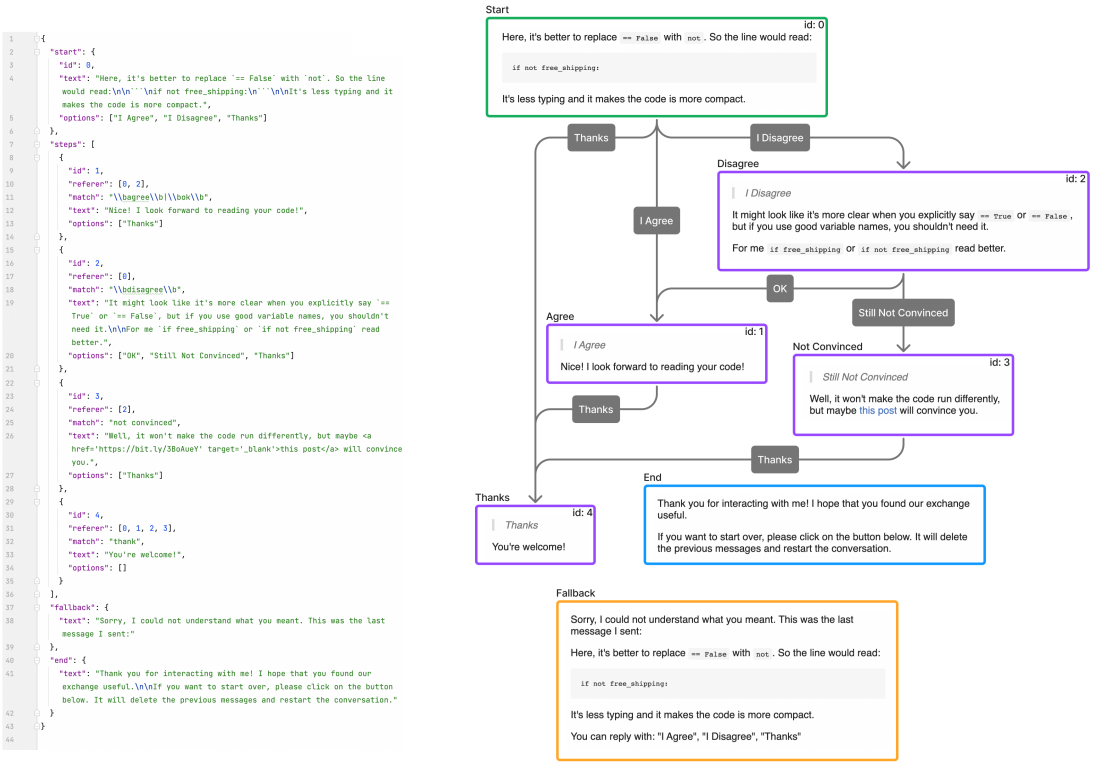

Figure 2. Our chatbot interaction script was configured using the JSON format (left), which supported a rule-based conversational flow comprising several potential pathways (right).

Another challenge related to the chatbot's configuration arose from the nature of the script's content. Given that the chatbot was interacting with the students in the context of a very specific topic, the structure of the interaction with the chatbot was relatively consistent. Nevertheless, the chatbot had to be repeatedly configured with the correct phrases and content pertaining to each code style issue that the instructors wanted to address in the exercises. While each script was fully encapsulated within each CRA, a side-by-side comparison of the two scripts shown in Fig. 3 reveals a large amount of redundant configuration. We refer to this configuration challenge as *redundant configuration* (CC2).

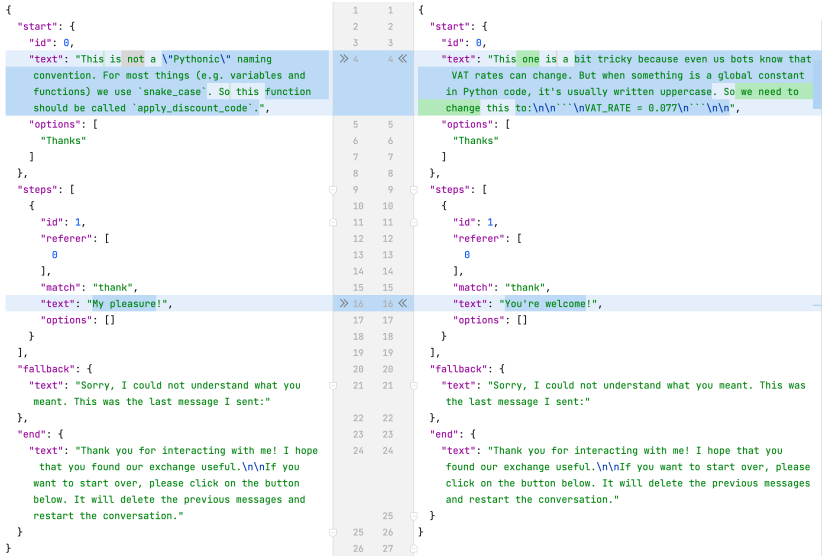

Figure 3. A side-by-side comparison of two similar chatbot scripts highlighting the differences between them reveals a large amount of redundancy between the script configurations.

A third issue that arose in the configuration of the chatbot scripts was related to the problem of matching student inputs to the most appropriate response from the set of all possible responses that the chatbot would be able to interpret. The approach taken in our implementation used regular expressions, focusing on important terms that could be present in student responses. This meant that the matching patterns had to accommodate the different variations of phrases that could be used to express a particular intent (e.g., *Yes*, *Yeah*, *Yup*, *OK*, and *Sure* can all be used to express agreement). Accommodating this richness of natural language required more exhaustive regular expressions that became hard to understand and maintain. We refer to this configuration challenge as *intent matching* (CC3).

3.1.2 Interaction

Regarding the interaction aspect, we first faced a potential issue arising from the fact that we were repeatedly exposing students to our chatbot. Comparing the number of interactions students had with the chatbot over the course of the Python labs, we see a downwards trend in the number of interactions while the number of students participating in the lab stayed relatively constant. This diminishing engagement with the chatbot after each exposure could limit the effective use of the chatbot to support learning, given that some students might not be exposed to the full set of interactions planned by the instructor. We refer to this interaction challenge as *repeated exposures* (IC1).

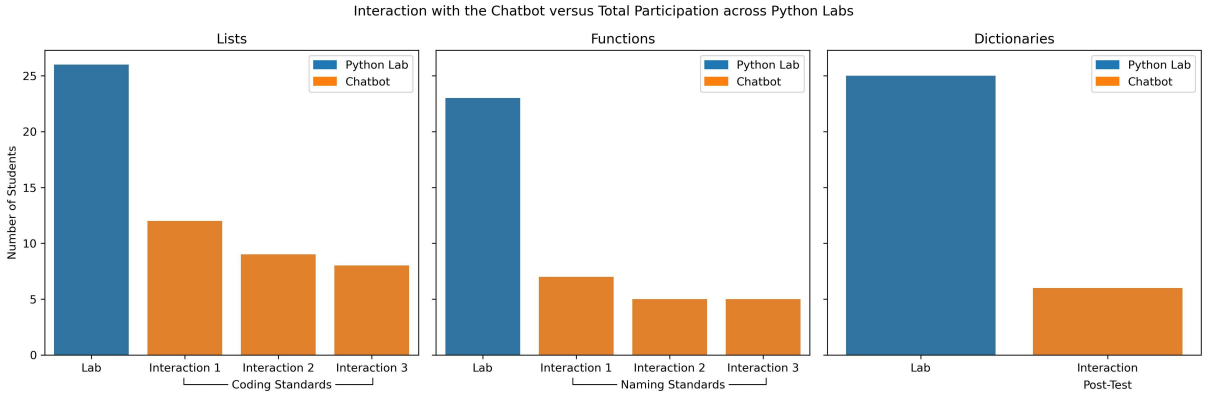

Figure 4. While the number of students participating in the Python labs remained almost constant, we observed a diminishing trend in the number of students who interacted with the chatbot.

Second, we observed that while most students interacted with the chatbot using the predefined quick-reply buttons, a few students sent messages using the free text input. One student appeared to probe the chatbot's ability to make small talk by asking the chatbot how it was doing at the end of the first interaction. After receiving a comment from the chatbot expressing that it did not understand what the student was trying to say, this student exchanged only one more message with the chatbot and did not interact again with the chatbot for the remainder of the course. We refer to this interaction challenge as *unmet expectations* (IC2).

3.2 Opportunities

We identified three opportunities related to the chatbot's configuration and two opportunities that emerged from the way students interacted with the chatbot. We present these opportunities below.

3.2.1 Configuration

A first advantage of our rule-based configuration is that the instructor can create predefined conversations and ensure that students' interactions with the chatbot are on-topic. Given that the specific context in which the chatbot interacted was limited—i.e., focused on explaining a set of Python programming best practices—it was easier to prepare predefined conversations that related to the scope of the code review exercise at hand. In our case study, our scripts had the sole purpose of providing the student with a path that would lead them toward a better understanding of the code style concept that was being presented. An example of this can be seen in Fig. 2. (right), which depicts a conversational flow in which the student is first asked whether the rule makes sense. In the affirmative case, the chatbot positively reinforces the use of this rule by exclaiming that it looks forward to reading the student's code. In the negative case, the chatbot proposes a more developed explanation of the rule. If this second

explanation does not convince the student that this rule is useful, the chatbot then provides a link to an external resource in an attempt to provide the student with a deeper treatment of the rule. We refer to this configuration opportunity as *predefined conversational pathways* (CO1).

Second, in case the chatbot is not able to fulfill the needs and requests of the student, the chatbot can be configured to allow the student to request the instructor to intervene in the interaction. This provides a failsafe option for situations in which the student's interaction with the chatbot is not yielding positive results. In these cases, allowing an instructor to intervene enables the delivery of personalized learning while still leaving the repetitive conversational tasks to a chatbot that can perform them automatically. We refer to this configuration opportunity as *instructor intervention* (CO2).

Finally, the conversational interactions supported on social coding platforms support the use of rich text comments that can be harnessed to express emotions through the use of emojis, animated gifs, and other media. Our CRA provides a Markdown text editor for comments, which allows our chatbot to interact expressively. Thus, instructors can be creative in the way they design the chatbot's responses. In our case study, we included interactions featuring emoji and animated gifs to allow the chatbot to express—among other emotions—humor and surprise, as shown in Fig. 5. We refer to this configuration opportunity as *rich expressions* (CO3).

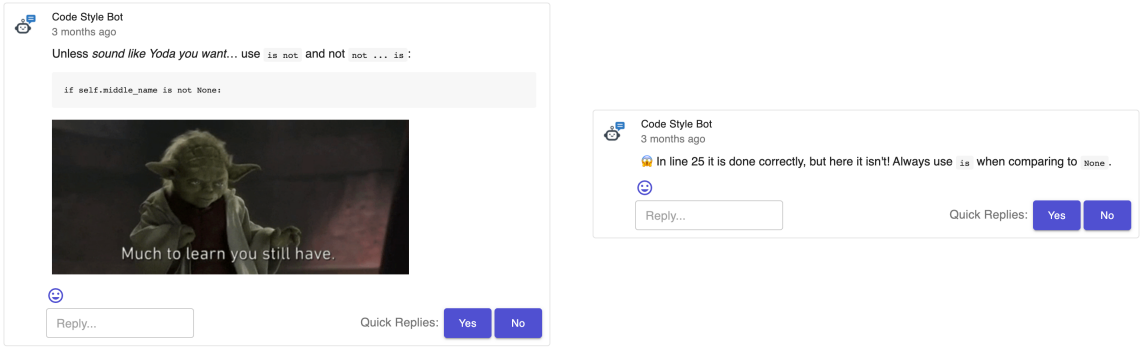

Figure 5. Some of the chatbot's comments featured animated gifs (left) and emojis (right) to elicit humor and express surprise, respectively.

3.2.2 Interaction

A total of 9 students rated their interaction with the chatbot. Their satisfaction is illustrated in Fig. 6, where the mean chatbot rating is displayed for each Python lab in which the chatbot was present. Students gave a rating based on a question asking them to rate the usefulness of the chatbot on a scale from 1 to 7, with 1 being *Not Useful at All* and 7 being *Very Useful*.

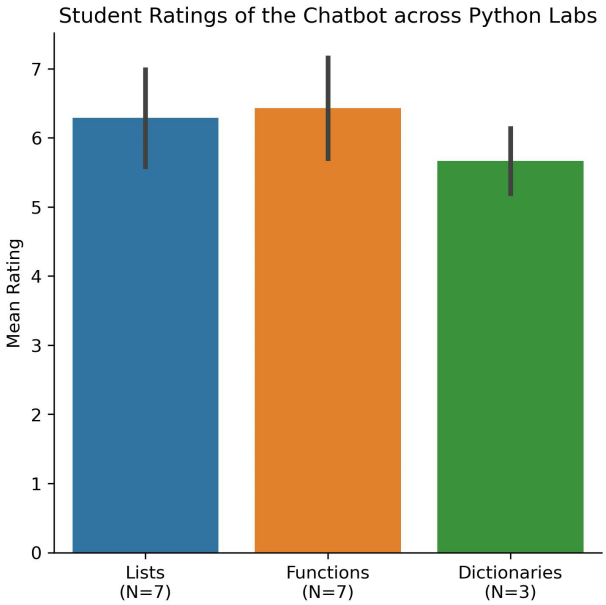

Figure 6. Students rated the chatbot positively across the three Python labs in which it was deployed.

Ratings were relatively consistent across all three Python labs. While 7 students rated the *List* lab ($\overline{x} = 6.29$, $\tilde{x} = 6.00$, $s = 0.76$) and *Functions* lab ($\overline{x} = 6.43$, $\tilde{x} = 7.00$, $s = 0.79$), only 3 students rated the *Dictionaries* lab ($\overline{x} = 5.67$, $\tilde{x} = 6.00$, $s = 0.58$). Nevertheless, the overall mean rating was $\overline{x} = 6.24$ ($\tilde{x} = 6.00$, $s = 0.73$), suggesting that the chatbot was useful in performing its task. We refer to this first interaction opportunity as *useful support* (IO1).

Second, while scripts enabled by the rule-based configuration could be seen as limiting, they can still be used to deliver personalized content to students. An advantage of an interactive chatbot is that it can provide personalized content to students interacting with it. This can allow instructors to adapt the chatbot's interactions to a particular student's understanding of a given topic. Following a strategy similar to the conversational flow shown in Fig. 2 (right), scripts can be conceived so that a chatbot first asks students a question to gauge their grasp of a given concept. Depending on the answer, the student will be shown a different conversational pathway, possibly including more in-depth explanations or follow-up questions. This in turn allows students who do not require the extra information to experience a shorter interaction with the chatbot and thus avoid overexposing students to interactions they might find tedious or repetitive. Personalized content can even be delivered based on a student's self-reported results for activities that take place outside of the chatbot's context. After the post-test exercise included in the last Python lab, we asked students to respond to the chatbot with *Yes* or *No* to indicate whether they had been able to identify the issues presented in the post-test. Of the 6 students that engaged with the chatbot, 3 reported their results with respect to all 12 issues, 2 reported on 10 out of the 12 issues, and 1 reported only on 3 issues. We refer to this interaction opportunity as *personalized content* (IO2).

4 DISCUSSION

In this section, we discuss how these findings affect the use of the CRA and what could be done in future work to mitigate these challenges and harness these opportunities. We structure our discussion by referring to the specific challenges and opportunities described in Section 3.

First, we identified that the format used for the configuration of our scripts is complicated, requires several iterations to get right, and might be prohibitively complex for a non-technical instructor (CC1). Challenges involving educational chatbot design and configuration have been previously highlighted in the literature [16]. While this type of challenge could be addressed by a technical person supporting instructors to set up and integrate interactive chatbots using our architecture, further challenges involving redundant configuration (CC2) and problems resolving student responses to the appropriate conversational pathway (CC3) highlight the need to simplify the configuration process, support automated script testing, replace pattern matching with more advanced intent-recognition techniques, and reduce redundant configuration. Moreover, these challenges could result in repetitive interactions, as instructors avoid the complexity of writing configurations from scratch by using similar configurations with the same students. Repeated exposures to similar scripts could quickly dissuade students from interacting with an all too predictable chatbot, as we observed in our case study (IC1). Similar results attributed to a *novelty effect* were also obtained by Fryer et al. in the context of an experimental study evaluating a chatbot in a language course [17].

To address the challenges faced in configuring our chatbot, one could envision a more modular system for chatbot scripts, providing a library of building blocks that could be easily assembled by instructors in order to intuitively create their own scripts. Building on features already present in other chatbot scripting languages (e.g., AIML [18] and RiveScript [19]), placeholders could be inserted into a script's content to support the manual or automatic creation of several, richer variations of a base interaction without having to modify the technical configuration. These placeholders would maximize a given configuration's portability and allow for more reusability and easier sharing of scripts across educational contexts.

While we faced several challenges in the process of configuring our chatbot scripts, opportunities emerged to guide students through predefined conversational pathways (CO1) and encourage expressive content (CO3). These opportunities highlight the applicability of chatbots to small-scale, scoped interactions, where they can still harness expressive affordances to elicit positive ratings from students (IO1). Although these task-based, scoped chatbots might not have to support a wide array of interactions—as in the case of general-purpose chatbots—they might still require to gracefully handle errors. Error handling could be achieved, as suggested by Niculescu and Banchs [20], through strategies harnessing humor or possibly through closer collaboration between the chatbot and the instructor, as enabled by our affordance to request instructor intervention (CO2). Implementation of these strategies could help avoid the effect of unmet expectations discouraging students from continuing to interact with

the chatbot (IC2). This effect was also noted by Molnár and Szüts, who warned that students can become frustrated by unsuccessful interactions with chatbots [21].

Chatbot technologies that can efficiently harness these opportunities and address these challenges could provide instructors with a chatbot they could rely on as a virtual teaching assistant, as previously suggested by Hobert [11]. This chatbot could support teaching staff in identifying students that might require further guidance and help deliver personalized content to these students (IO2). Indeed, our pilot case study reveals that scripts do not need to be complex to positively engage students and prompt them to self-report their learning process.

5 CONCLUSIONS

In this paper, we outlined the challenges faced and opportunities identified during a pilot case study aimed at integrating interactive chatbots into introductory programming education. Our chatbot was configured to follow a rule-based script that provided explanations of Python programming best practices and was deployed within our CRA to support students in completing online code review exercises. A mixed-method analysis allowed us to identify several friction points encountered and affordances enabled by our chatbot integration. Notably, while our chatbot's script was hard to configure and possibly repetitive when encountered multiple times over a short time span, students that interacted with the chatbot tended to follow the conversational pathway set forth by the instructor and rate the chatbot positively. Our findings will allow us to refine the design of our chatbot integration as well as that of our technological and pedagogical contexts for future evaluation. Furthermore, these findings could help guide educators and researchers in digital education looking to incorporate our chatbot integration—or other similar solutions—into their practice.

REFERENCES

[1] J. Quiroga Pérez, T. Daradoumis, J.M. Marquès Puig, "Rediscovering the Use of Chatbots in Education: A Systematic Literature Review," in *Computer Applications in Engineering Education*, vol. 28, no. 6, pp. 1549–1565, 2020, doi:10.1002/cae.22326.

[2] A.C. Graesser, P. Chipman, B.C. Haynes, A. Olney, "AutoTutor: An Intelligent Tutoring System with Mixed-Initiative Dialogue," in *IEEE Transactions on Education*, vol. 48, no. 4, pp. 612–618, 2005, doi:10.1109/TE.2005.856149.

[3] F. Clarizia, F. Colace, M. Lombardi, F. Pascale, D. Santaniello, "Chatbot: An Education Support System for Student," in *CSS 2018: Cyberspace Safety and Security* (A. Castiglione, F. Pop, M. Ficco, F. Palmieri, eds.), pp. 291–302, Cham: Springer, 2018, doi:10.1007/978-3-030-01689-0_23.

[4] G.J. Hwang and C.Y. Chang, "A Review of Opportunities and Challenges of Chatbots in Education," in *Interactive Learning Environments*, pp. 1–14, 2021, doi:10.1080/10494820.2021.1952615.

[5] 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)*, pp. 928–931, 2016, doi:10.1145/2950290.2983989.

[6] M. Wessel, B. Mendes de Souza, I. Steinmacher, I.S. Wiese, I. Polato, A.P. Chaves, M.A. Gerosa, "The Power of Bots: Characterizing and Understanding Bots in OSS Projects," in *Proceedings of the ACM on Human-Computer Interaction*, vol. 2, no. CSCW, pp. 1–19, 2018, doi:10.1145/3274451

[7] M. Binkis, R. Kubiliūnas, R. Sturienė, T. Dulinskienė, T. Blažauskas, V. Jakštienė. "Rule-Based Chatbot Integration into Software Engineering Course," in *ICIST 2021: Information and Software Technologies* (A. Lopata, D. Gudonienė, R. Butkienė, eds.), pp. 367–377 Cham: Springer, 2021, doi:10.1007/978-3-030-88304-1_29.

[8] M. Laiq and O. Dieste, "Chatbot-Based Interview Simulator: A Feasible Approach to Train Novice Requirements Engineers," in *2020 10th International Workshop on Requirements Engineering Education and Training (REET)*, pp. 1–8, 2020, doi:10.1109/REET51203.2020.00007.

[9] M. Ismail and A. Ade-Ibijola, "Lecturer's Apprentice: A Chatbot for Assisting Novice Programmers," in *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pp. 1–8, 2019, doi:10.1109/IMITEC45504.2019.9015857.

[10] R. Winkler, S. Hobert, A. Salovaara, M. Söllner, J.M. Leimeister. "Sara, the Lecturer: Improving Learning in Online Education with a Scaffolding-Based Conversational Agent," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2020, doi:10.1145/3313831.3376781

[11] S. Hobert, "Say Hello to 'Coding Tutor'! Design and Evaluation of a Chatbot-Based Learning System Supporting Students to Learn to Program," in *40th International Conference on Information Systems (ICIS 2019)*, vol. 3, pp. 1776–1792, 2020.

[12] M. Towhidnejad and A. Salimi, "Incorporating a Disciplined Software Development Process in to Introductory Computer Science Programming Courses: Initial Results," in *Technology-Based Re-Engineering Engineering Education Proceedings of Frontiers in Education FIE '96 26th Annual Conference*, vol. 2., pp. 497–500, 1996, doi:10.1109/FIE.1996.572893.

[13] J.C. Farah, B. Spaenlehauer, M.J. Rodríguez-Triana, S. Ingram, D. Gillet, "Toward Code Review Notebooks," in *2022 22nd IEEE International Conference on Advanced Learning Technologies (ICALT)*, 2022. Retrieved from https://infoscience.epfl.ch/record/293832.

[14] J.C. Farah, B. Spaenlehauer, M.J. Rodríguez-Triana, V. Sharma, S. Ingram, D. Gillet, "Impersonating Chatbots in a Code Review Exercise to Teach Software Engineering Best Practices," in *2022 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1634–1642, 2022, doi:10.1109/EDUCON52537.2022.9766793.

[15] D. Gillet, I. Vonèche-Cardia, J.C. Farah, K.L. Phan Hoang, M.J. Rodríguez-Triana, "Integrated Model for Comprehensive Digital Education Platforms," in *2022 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1587–1593, 2022, doi:10.1109/EDUCON52537.2022.9766795.

[16] J.A. Kumar, "Educational Chatbots for Project-Based Learning: Investigating Learning Outcomes for a Team-Based Design Course," in *International Journal of Educational Technology in Higher Education*, vol. 18, no. 65, pp. 1–28, 2021, doi:10.1186/s41239-021-00302-w.

[17] L.K. Fryer, M. Ainley, A. Thompson, A. Gibson, Z. Sherlock, "Stimulating and Sustaining Interest in a Language Course: An Experimental Comparison of Chatbot and Human Task Partners," in *Computers in Human Behavior*, vol. 75, pp. 461–468, 2017, doi:10.1016/j.chb.2017.05.045.

[18] R.S. Wallace. "The Anatomy of A.L.I.C.E.," in *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer* (R. Epstein, G. Roberts, G. Beber, eds.), pp. 181–210, Dordrecht: Springer, 2009, doi:10.1007/978-1-4020-6710-5_13.

[19] N. Petherbridge, RiveScript, Accessed 17 May 2022. Retrieved from https://www.rivescript.com/

[20] A.I. Niculescu and R.E. Banchs, "Strategies to Cope with Errors in Human-Machine Spoken Interactions: Using Chatbots as Back-Off Mechanism for Task-Oriented Dialogues," in *Proceedings of the Errors by Humans and Machines in Multimedia, Multimodal and Multilingual Data Processing Workshop (ERRARE 2015)*, pp. 1–6, 2015.

[21] G. Molnár and Z. Szüts, "The Role of Chatbots in Formal Education," in *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 197-202, 2018, doi:10.1109/SISY.2018.8524609.