Conversational agents may improve problem-solving times but don't affect perceived satisfaction

Dorothy Howard
Communication
UC San Diego
dhoward@ucsd.edu

Kandarp Khandwala
Computer Science
UC San Diego
kkhandwala@ucsd.edu

ABSTRACT
As distance programming education grows, exploring ways to provide one-on-one tutoring is important. We conducted an experiment to study how conversation agent-tutors impact programming. Participants solved two problems in an online learn-to-code platform. In the control, they were given no outside assistance. In the treatment, we simulated an assistive chat providing feedback including encouragement and adapted responses from a corpus of relevant conversations. We measured changes in participants' time-to-completion and surveyed their satisfaction in the experience. The results indicate that conversational agent-tutors may improve the problem-solving speed, but don't affect perceived satisfaction. This experiment provides insights into how the design of a conversational agent can meaningfully augment learn-to-code interfaces.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

Author Keywords
Conversational agents; chatbots; programming education

INTRODUCTION
When students are learning to code, they often need to ask someone for assistance when they are stuck. As distance programming education grows, learn-to-code platforms need to design tools to help students overcome roadblocks. For example, 78% of chat sessions on `pythontutor.com` [3] in a recent two-month period involved a lone participant expecting to be provided help and eventually expressing frustration. In online courses, the time in which students have access to direct feedback might be limited to the duration and resource constraints of the class. Conversational agents are a promising option to help students work through problems on their own, while maintaining low response times and helping manage costs.

Tutors fulfill many roles including: concept reinforcement, encouragement, reward, and guidance if a learner is off-track. For in-classroom tutoring, there is a wide variation of the quality of support as each tutor has their own strengths, weaknesses, and biases towards certain methods of problem-solving and presenting information. Conversational agents offer a more reliable option for several reasons, including that they can be automated to provide consistent responses to various question types and skill levels.

How could conversational agent-tutors impact people programming in a learn-to-code environment? Our first conjecture is that the presence of chatbots will improve users' speed in coding tasks. Secondly, we predict that they will also increase satisfaction in the experience because they can offer consistent feedback to users hoping to overcome moments when they are stuck. We also predict that satisfaction would increase as speed decreases, because the chat offers supportive feedback.

RELATED WORK
Conversational agents are called by many names, including chatbots, chatterbots, dialogue systems, and bi-directional communication assistants. For simplicity, we refer to conversational agents-tutors as chatbots in the rest of the paper.

Recent research has explored the general use, flexibility, and responsiveness of chatbots in a variety of domains [8]. Research specific to chatbots in programming education has suggested that they offer opportunities for increasing learner reflection and accuracy [4, 5]. Previous work has tested the effect of chatbots on student ability to answer text-based problems which test for conceptual understanding. One such study measured "depth of responses" for text-based questions, a measure which cannot be easily applied to programming tasks [6]. Another group of work has prototyped cognitive tutors to help students solving problems in LISP, and geometry and algebra using measures for types of knowledge attained and skill strengthening over time [1,11]. Our project can be differentiated from this previous research by its evaluation measures of speed and satisfaction.

There are a number of classification schemes for the natural language, conversational dynamics between tutors and students more broadly, including aspects of conversation like prompted v. unprompted feedback, concept articulation, ability to "pass" as bot, and friendliness. For example, using high-level typologies for tutor-student dialogue like: yes/no question; explicit conceptual question; summarizing; indication of uncertainty, researchers can find common question areas and create responses [5, 7, 10]. These dialogue schemes informed our creation of canned responses.

METHODS
The study was carried out on `pythontutor.com`, a tool to write and visualize code (Figure 1). We chose a visualization tool in order to assist debugging, which is a vital aspect of coding and one that a chatbot is not well equipped to handle. The researcher not directly facilitating the study would pretend to

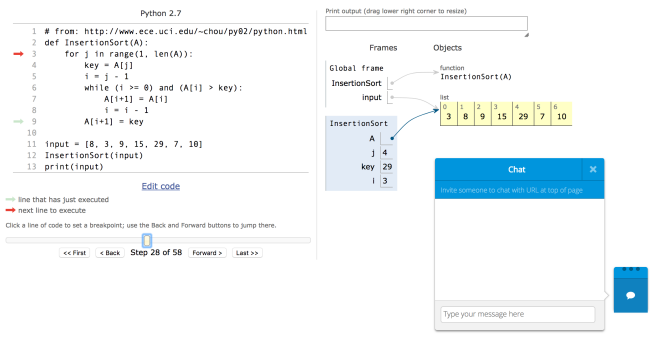

Figure 1. Screenshot of the interface: code on the top-left, buttons to step through execution to the left, artifacts on top-right, chat to bottom-right

be a chatbot in a Wizard-of-Oz (WoZ) model [5, 6], providing responses via the shared session feature. We decided on this structure after performing two pilot studies, in which it was hard to maintain the illusion if both researchers were in the room with the participant.

We used a combination of pre-determined scripts and minimal improvisation to WoZ the chat. These were based on natural language conversations observed in over 100 chat logs we obtained from Python Tutor. In this corpus of chats, we noted questions about specific errors, concepts (such as memory addresses), topics about code (such as what a parameter is) and syntax. We converted some common phrases into templates using a variety of tutor response utterance typologies [9]. Some examples follow:

- `Need any more help?`
- `you need to remember _______`
- `squared = list(map(lambda x: x**2, items))`
- `I'm sorry, I don't understand your question.`

Note that the third example was provided in context of using lambdas along with a filter function, which would be hard to automate correctly (hence the indirect response).

One aspect we iterated during the study was to provide code snippets from Python documentation as examples. As graceful fallback, when an appropriate response could not be found, content from the top search results using keywords from the participant's query, or a generic response asking for clarification was provided. Errors are characteristic of chatbots, and occasional errors were made in the chat by us, deliberately or otherwise.

Experiment

To begin, each participant was provided a sample piece of code to execute, as a way to become familiar with the interface and warm-up before attempting the questions. Participants were also informed that there was no time limit to any task we assigned, but they were free to give up (or quit the study) for any reason. Next, each participant was given the same two problems to code, one after another. They could take a break in between if desired. After completion of each problem, participants filled out the [same] 3-question survey:

- `Do you find the interface easy to use?`
- `Do you think that the interface helped you solve the problem?`
- `Do you think the problem was appropriate for your skill level?`

Possible responses were on a 5-point Likert scale: (strongly) disagree, neutral or (strongly) agree. The study ended with an informal chat for any qualitative feedback/ questions from participants.

The order of the two problems was randomly assigned to counterbalance any learning effects. One problem involved returning the even numbers from a list (P1); the second involved optimizing a naive recursive implementation of Fibonacci using memoization (P2). The control condition, where the problem was solved using the interface without a chat was always tested first. Apart from the official Python documentation and a blog post that described memoization, no other resource/assistance was allowed in this condition. We did not specify whether the chat was operated by a human or automated, even if asked by a few curious participants.

Participants

A total of 12 participants (3 female) completed the 45-minute in-person study. All participants were graduate students at UC San Diego. We required participants to have some familiarity with coding in Python. Participants were compensated with $5 amazon.com gift cards. One of the participants was excluded in the results as the problems were disproportionate to his/her skill (strongly disagree on Q3 of the survey).

RESULTS

Time to Completion

We defined time to completion as the period elapsed between a participant seeing the problem to when they executed a correct solution. Between the two conditions, we did not find a significant difference in problem-solving speed (p=0.30 in the one-tailed t-test).

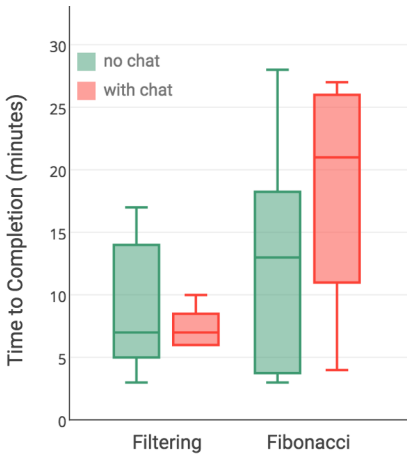

Figure 2. Speed, across problems, is not consistent.

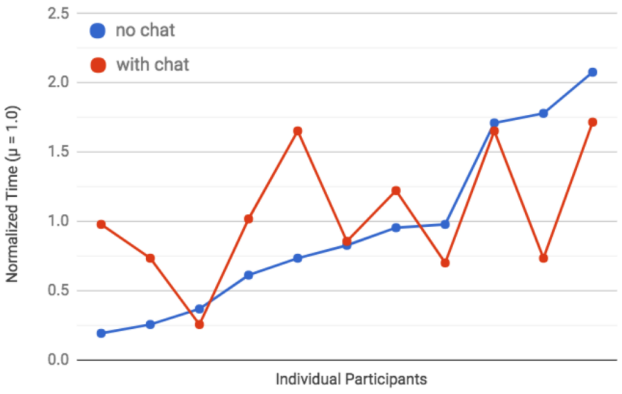

Figure 3. Within-subjects comparison of speed.

First, performance varied for each problem, as seen in Figure 2. In P1, variance in the treatment is lower (SD=1.67 with chat; SD=5.49 without). This result may indicate filtering a list (P1) is a largely syntactic problem rather than one with conceptual difficulty like memoizing a recursive Fibonacci function (P2). In P2, we see that the median time taken by participants in the treatment is greater (21 minutes with chat, 13 minutes without), contrary to what was expected. This result may be because some responses of the chat were not helpful. For example, in their feedback, a few participants said they did not appreciate being provided a Wikipedia definition of memoization.

Next, for a within-subjects comparison we normalized the durations by dividing the time taken in a particular problem by the mean across all participants for that problem, since each participant solved both problems. Figure 3 shows participants ordered by the time taken without chat, which serves as the baseline. Participants that were slowest without a chat are the ones who saw improvement (control mean=1.64μ, treatment mean=1.20μ, normalized time: lower is better), as depicted in the right half of the plot. On the other hand, the faster participants did worse in the treatment (control mean=0.56μ; treatment mean=0.96μ, normalized times: lower is better). This result may be because the treatment disrupted their well-established workflows, as opposed to providing a reasonable (if not better) alternative for the slower group. In their feedback, however, participants liked having a chat as they did not have to switch context to refer to documentation or search.

Perceived Satisfaction

We compared the quality of the experience with the questionnaire rated on a 5-point Likert scale. 8 of 11 participants' perceptions of the ease of use of the interface between the treatment and the control did not change at all, resulting in similar ratings (control mean=4.09, SD=0.83; treatment mean=3.91, SD=1.04). This indicates that the chat was felt to be as natural to use as the rest of the programming interface, and any discomfort is offset by provision of help in the same context.

In the next question about whether the interface helped solve the problem, 8 of 11 participants' opinions changed for better or worse. However, there was no significant difference in aggregate (control mean=3.91, SD=0.70; treatment mean=4.09, SD=0.94). We believe this can be explained by considering the time taken by participants. 4 of 5 participants with higher ratings in the treatment were in the faster group of participants in the control. There may be an association between time and satisfaction that supersedes *how* they solved the problem, despite the chat slowing them down relatively. Alternatively, participants might have thought that without appropriate assistance from the chat, they might have struggled for even longer.

DISCUSSION

This study provides insights into how the design of a conversational agent can meaningfully augment learn-to-code interfaces.

There are several elements of the experimental design which might be revised to make the results more realistic. We chose not to allow participants to use search to decrease the potential interference of other variables such as the quality of online documentation. However, if we were to repeat this study, we would consider allowing participants to do so to make our study more realistic to everyday behavior.

Our results show that between-subjects time-to-completion is unaffected by chat v. no chat, mainly because, while participants with above the mean times without the chat improved with the chat, participants with below the mean times worsened in performance. Variation in the time-to-completion between the two problems tested also indicates that they are of different difficulty levels. Future work would benefit from demarcating the conceptual and procedural difficulty of the programming problems assigned. Such work would strengthen our ability to measure the effects of the chatbot.

Further, some participants were not as familiar with Python as others we tested, yet they were not new to programming. In further work, adaptive conversational chatbot tactics based on the user's skill might help scale to learners from different backgrounds. We also believe this information will be useful to explore aspects of design of the chat that change why people do and don't interact with it.

In our results, ease of use as a measure of satisfaction didn't change appreciably either. We believe this might be because the chat was well integrated in the interface. An inexplicable result is that those participants who may have been slowed down by the chat still preferred it in the second measure (of helpfulness). Exploring reasons for this could be an interesting exercise.

Finally, it is not in the scope of this study to examine short and long-term learning retention. But, we believe that this is a gap, considering that learning retention is a key aim of educational projects.

ACKNOWLEDGEMENTS

We thank Prof. Philip Guo for access to Python Tutor data and our Interaction Design Research classmates for their feedback and of course, participating in our user study!

REFERENCES

1. Anderson, John R., Corbett, Albert T., Koedinger, Kenneth R., and Pelletier, Ray. 1996. Cognitive Tutors: Lessons Learned. *The Journal of The Learning Sciences*.

2. DeAngeli, A., G.I. Johnson, L. Coventry. The unfriendly user: exploring social reactions to chatterbots. 2001. *International Conference on Affective Human Factors Design*.

3. Guo, P. 2015. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. *UIST '15*.

4. Kerly, A. & Bull, S. 2006. The Potential for Chatbots in Negotiated Learner Modelling: A Wizard-of-Oz Study. *Intelligent Tutoring Systems*.

5. Kerly, A., Hall, P., & Bull, S. 2007. Bringing chatbots into education: Towards natural language negotiation of open learner models. *Knowledge-Based Systems Volume 20, Issue 2*.

6. Kumar, R., & Rosé, C. P. 2011. Architecture for building conversational agents that support collaborative learning. *IEEE Transactions on Learning Technologies*.

7. Lu, X., Di, E. B., Kershaw, T. C., Ohlsson, S., & Corrigan-Halpern, A. 2007. Expert vs. Non-expert Tutoring: Dialogue Moves, Interaction Patterns and Multi-utterance Turns. *Lecture Notes in Computer Science*.

8. Radziwil, N., Benton, M. 2017. Evaluating Quality of Chatbots and Intelligent Conversational Agents. https://arxiv.org/abs/1704.04579

9. Stolcke, A; Ries, K., et al. 2000. Dialogue Act Modeling for Automatic Tagging and Recognition of Conversational Speech." *Association for Computational Linguistics*.

10. McGill, T. & Volet, S. 1997. A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education. Vol 29, Issue 3*.

11. Koedinger, K. & Corbett, A. Cognitive Tutors: Technology Bringing Learning Science to the Classroom. *The Cambridge Handbook of the Learning Sciences*.