Listen Veronica! Can You Give me a Hand With This Bug?

Listen Veronica! Can You Give Me a Hand With This Bug?

Juan Pablo Sáenz
Politecnico di Torino
Turin, Italy
juan.saenz@polito.it

Luigi De Russis
Politecnico di Torino
Turin, Italy
luigi.derussis@polito.it

ABSTRACT

Developing software implies looking for documentation, following tutorials, making implementation decisions, encountering errors, and overcoming them. Behind each aspect is the developer's reasoning that, if not collected, is lost after the implementation. Conversely, if captured and linked to the code, the developers' reasoning and motivations for each step they accomplish can become a valuable asset, meaningful for them and other developers. Looking for a mechanism to capture such knowledge seamlessly, we present Veronica. It is a conversational agent integrated directly into Visual Studio Code that, based on the developers' self-explanatory reasoning, records memos and links them with the code they are writing. Furthermore, Veronica can interact with the web browser to automatically gather the sources consulted by the developer and attach them to the code. We validated our approach by conducting a usability study with eight participants that positively assessed the tool's usefulness and suggested improvements in the graphical interface.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Software and its engineering** → *Development frameworks and environments.*

KEYWORDS

conversational agent, development environment, software, documentation

ACM Reference Format:
Juan Pablo Sáenz and Luigi De Russis. 2023. Listen Veronica! Can You Give Me a Hand With This Bug?. In *Companion of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '23 Companion), June 27–30, 2023, Swansea, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3596454.3597179

1 INTRODUCTION AND MOTIVATION

When approaching a new programming language or getting proficient with a given development framework, programmers rely on online tutorials, forums, or source code examples to find solutions and overcome development and execution errors [6, 8]. Based on their reasoning and motivations, each developer decides which resources to follow and how to implement their solution. While, typically, developers comment on the code once the implementation is finished or they have reached a working version, the consulted sources, the reasoning over the code, the implementation choices, and the issues they found throughout the development process remain largely undocumented.

In this context, critical aspects such as background knowledge, the rationale for the solution, or step-by-step instructions for arriving at similar or related solutions are typically left out of the documentation [11]. Consequently, the documentation linked to the code is not commonly helpful to themselves or other developers to overcome cognitive barriers or guide the development of new projects. While various works aim to ease the understanding and integration of online code examples [7, 12, 15], fewer research efforts have been devoted to supporting novices in seamlessly documenting their development process on the go and with their own words.

Against this backdrop, we consider that programmers can produce documentation that has the potential to become a valuable asset, meaningful for them and other developers if enabled to capture various points of the development process seamlessly and add self-explanatory insights to it [3]. In particular, we rely on the fact that a **self-explanation strategy** can increase their awareness of the implemented solution [23, 24]. Indeed, when learners provide explanations — even to themselves — they learn more effectively and generalize more readily to novel situations [9, 26]. It is well-known from the literature that novice programmers usually lack metacognitive awareness — the ability to think about and reflect on their problem-solving process — and fail to progress toward a working solution [17].

In this paper, we present *Veronica*, a conversational agent integrated into Visual Studio Code to help developers self-explain and document their development process through "memos" that they can create textually or vocally. Such memos consist of self-explanatory comments or the URLs of the online resources consulted by the programmer, which Veronica automatically gathers from the browser. We choose a conversational agent (CA) since, in recent years, they have become widely used in several contexts, including for software engineering. CAs are software components designed to respond to human statements with a specific set of predefined replies. In the software engineering context, they react to external stimuli, such as events triggered by tools and messages posted by users, and run automated tasks in response. They work as an interface between developers and services, e.g., acting as Q&A bots for information retrieval and are integrated into IDE for automating bug repair [20], like in our proposal.

Finally, after design and implementing Veronica, we evaluated it by conducting a usability study. The study results revealed that

EICS '23 Companion, June 27–30, 2023, Swansea, United Kingdom

ACM ISBN 979-8-4007-0206-8/23/06…$15.00
https://doi.org/10.1145/3596454.3597179

developers with diverse expertise find the tool helpful and distinguished the improvements we must implement in the extension to improve its usability.

2 BACKGROUND

Veronica is built upon three research topics: Software Engineering Bots, Conversational Agents, and Developers' Notetaking. In the following, we provide context and describe how our approach builds upon them.

2.1 Software Engineering Bots

Software Engineering Bots are applications that react to external stimuli, such as events triggered by tools and messages posted by users, and run automated tasks in response. They work as an interface between users and services and include conversational capabilities to interact with end users through textual messages (in chatbots) and speech (in voice bots) [20]. Several bots have been implemented to improve developers' workflow through better productivity [10]. Indeed, according to Wang *et al.* [25], over 60% of one thousand popular GitHub Open Source repositories employ at least one bot to automate routine workflows. Furthermore, Storey *et al.* [22] characterized the types of bots used in software development based on their roles. The identified types were: code bots, test bots, DevOps bots, support bots, and documentation bots.

In this context, Veronica's approach lies in the category of a documentation bot. These bots commonly aim at automating documentation generation from resources such as Stack Overflow[1], easing the process of extracting useful information from software repositories [1], engaging developers by being an interactive interface to static documentation pages [19], or generating reports and dashboards by integrating analytics and visualization services [22]. However, Veronica is targeted at non-experienced developers. Rather than automating the documentation generation, it aims to ease self-explain their development process and thoughts as they arise and keep track of the online sources that motivated their implementation decisions.

2.2 Conversational Agents

Conversational Agents (CA) are user interfaces that interact with users via written or spoken natural language. As input, they accept natural language as speech, text, or video, process it, and provide relevant advice or feedback through text or speech or by manipulating a physical or virtual body [2]. In recent years, CAs have become widely used in several contexts, mainly to increase accessibility for people with physical disabilities or language barriers [14, 18]. When coupled with bots, such conversational interfaces are commonly called conversational bots. In Software Engineering tasks, developers can access and command bots through natural language communication by using conversational bots [19]. They establish a natural communication between bots and developers, reducing the tasks' learning curve. Conversational bots range from chat-based communications to voice-controlled bots [4].

In our proposal, we decided to implement Veronica as a conversational agent that can interact with users, either textually or vocally. Since software development already requires switching among various contexts [16], we propose that integrating the conversational agent directly into the Integrated Development Environment (IDE) might represent an advantage to the programmers. Similarly, our decision to support the vocal input was to avoid distractions or interruptions in the developers' regular coding workflow.

[1]https://stackoverflow.com/, last visited on January 19, 2023

2.3 Developers' Notetaking

Finally, notetaking is the most common way for developers to cope with task suspension and resumption. However, according to Parnin *et al.* [16], although 77% of developers use notes to keep track of their progress during programming, 75% of these notes are unsituated, meaning they are not kept in the editor. Similarly, notes are commonly taken to support developers' own comprehension of the code [13], for keeping track of helpful resources [12], and for keeping track of their progress in a development task [5]. Nevertheless, Shinyama *et al.* [21] determined that information designed for code authors appeared less often than formal types of documentation, indicating that developers are not frequently commenting on the code for their own benefit.

Given this context, Veronica's design, from the notetaking point of view, aims to: (i) enable developers to attach self-explanatory comments in their code, (ii) persist such comments directly in the development environment, and (iii) link helpful online resources to the code.

3 USAGE SCENARIO

To illustrate how Veronica looks and works, we present a usage scenario framed in the context of a novice developer solving programming exercises using Python.

- Laura is a non-experienced developer working on her university programming course exercises. She opens Visual Studio Code, her preferred IDE.
- At some point, she gets stuck in solving an exercise requiring the implementation of a bubble sort algorithm. As commonly happens when a developer gets stuck in the coding, she starts thinking aloud, reviewing the code she has written. Then, after some seconds without changing the code, Laura decides to enable Veronica, the conversational agent.
- A right sidebar opens in Visual Studio Code, displaying Veronica's chat. Veronica starts the interaction with the developer by showing a message greeting and offering its help to create memos and track the Google Chrome browser navigation. Apart from displaying the chat, the microphone is also enabled when Veronica is launched. In this manner, Laura can interact vocally or textually.
- Laura is not sure about where to start. So she opens her browser and heads to the course website, where she finds a step-by-step conceptual explanation of how the bubble sort algorithm works. Additionally, she opens a video on YouTube explaining the algorithm and how to implement it in a programming language different from Python.
- With a clear idea of how the bubble sort algorithm works from a conceptual point of view, Laura switches back to Visual Studio Code and resumes the writing of the code. Veronica automatically notices this behavior and gathers the

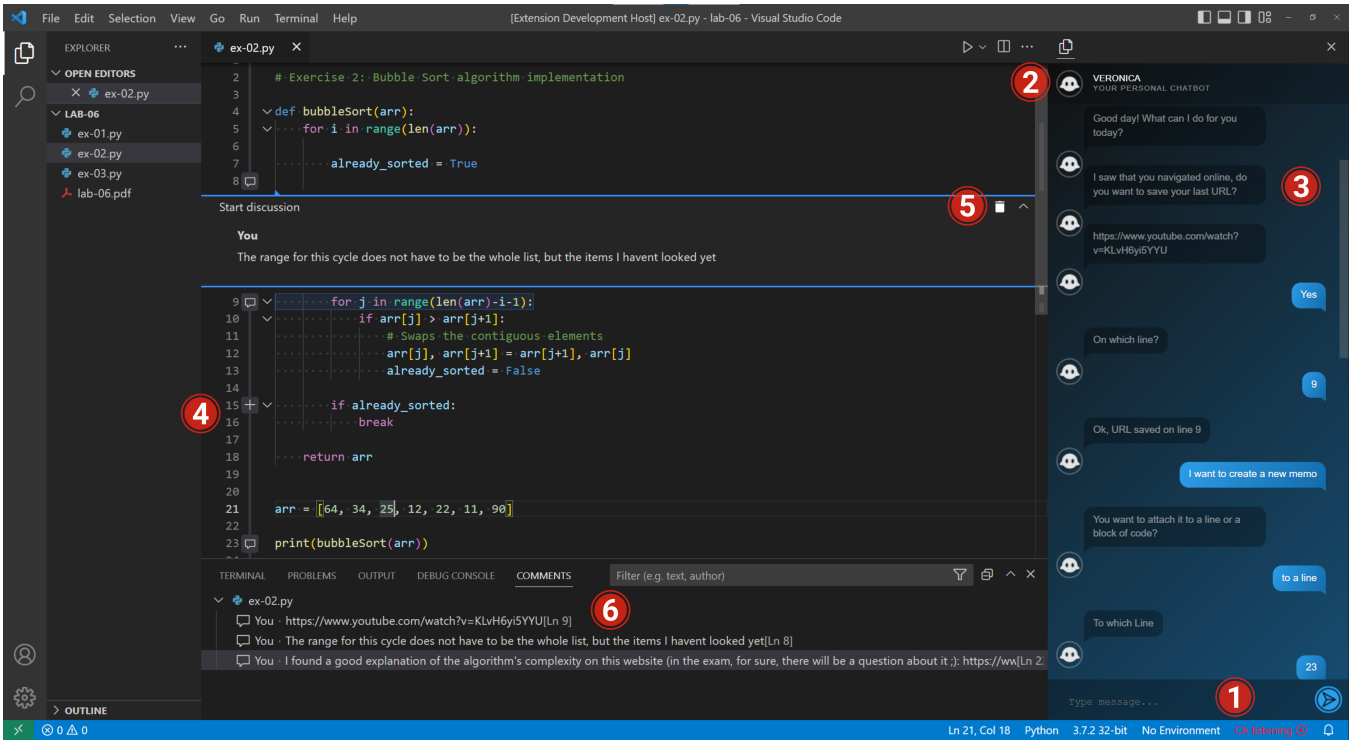

Figure 1: Snapshot of the Veronica Visual Studio Code extension. For illustrative purposes, the figure shows several functionalities co-occurring. Veronica provides the following functionalities: (i) seamlessly record memos natively pinned into the code and directly from the IDE; and (ii) automatically track the user navigation on the web browser to link the consulted online resources to a particular memo. In this manner, developers can trace the relevant pages upon which they implemented their solution.

URLs of the consulted websites. It then asks Laura if she wants to save the visited URLs as a memo. Since she prefers to describe how the algorithm works in her own words, she replies "no."

- Laura has been able to overcome the doubt that was blocking her, she wants to create a memo to remember in her own words what she understood from the documentation and how it helped her to keep going with the implementation. Therefore, she says, "Veronica, I want to create a new memo."
- Veronica asks Laura, via the chat and audio, if she wants to attach the new memo to a line or a block of code. Laura answers out loud that she wants to attach it to line 86. Then Veronica asks what she would like to insert in the memo, and Laura inserts by voice the explanation she wants to remember on how to deal with the indexes of the nested loop correctly.
- Later, while developing another exercise, Laura starts again the conversation with Veronica, asking her to create a new memo to remember why she decided to use a dictionary as the data structure. Although there is the option to do it vocally, Laura makes the new memo by writing it on the chat. Veronica then asks Laura if she wants to attach the website currently open in the web browser. Since her decision was influenced by the online sources she consulted, Laura answered affirmatively.
- Finally, Laura disables Veronica from listening by saying "Goodbye, Veronica." Naturally, all the memos remain attached to the code file at the specific line numbers. Therefore, Laura or anyone else can consult them when the code file is open again.

4 VERONICA DESIGN AND IMPLEMENTATION

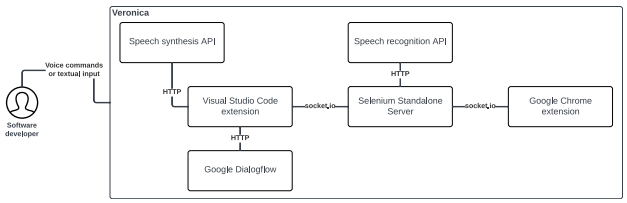

Figure 2: Veronica's architecture

4.1 Design goals

We now revisit the features of Veronica, with a focus on design motivations and implementation details. The tool was inspired by the authors' observations throughout their experience teaching web development courses at their university during the last four years. We refined the graphical interface design over several rounds of discussions producing hand-drawn sketches. Then, we implemented the Visual Studio Code extension as a Typescript project. The main graphical component is the right sidebar (Figure 1). Veronica addresses the following three considerations:

- **Supporting the documentation 'on the go' directly from the IDE.** Veronica aims to encourage programmers to document their development process while programming in the most fluid way possible. For this reason, we decided to keep the extension visible next to the code. Consequently, the user can enable Veronica by clicking the button on the right of the Visual Studio Code Status Bar, as shown in the figure indicated with the number (**1**). Once Veronica is activated, the chat displays on the right sidebar as indicated in (**2**). Besides, the text color becomes red, and a recording icon is placed next to the label 'CA listening,' indicating that from that moment, Veronica will continuously listen to the microphone and transcribe everything into the chat. Similarly, when the user wants to disable the extension, they has to click the same button (**2**). Finally, a memo can be created in three ways: by typing 'Create new memo' in the chat, saying it out loud, or manually clicking on a + button displayed when hovering the cursor over the code (**4**). The deletion of a memo can be achieved with the command 'Delete the memo in line...' or by clicking the trash icon button, as shown in (**5**).
- **Automatically gathering information from the web browser and integrating it into the code.** Programmers commonly switch between the IDE and the browser, searching for documentation and ways to overcome the errors they find along the development process. For this reason, every time the programmer returns to Visual Studio Code after opening the browser, Veronica automatically gathers the URL of the latest visited websites and asks the developers if they want to link such URLs to some portion of the code as indicated in (**3**). This functionality prevents the programmer from forgetting which documentation inspired their implemented solution. In this sense, if another developer opens the code file, he would understand the reasoning behind the implementation decisions. Additionally, the URLs automatically gathered from the browser are always associated with a memo. Therefore, a memo is pinned to a specific line or portion of code, and its content consists of self-explanatory annotations and, eventually, links to the consulted documentation.
- **Visualizing and navigating the memos present in a file.** To enable the developers to navigate through the memos they created, we decided to add a panel listing all the memos in the file currently opened in the editor, as shown in (**6**). In this manner, the developer has a consolidated view of the self-explanatory comments. Naturally, these memos do not aim at replacing the comments on the code file. While the comments in the code commonly follow some conventions and intend to provide technical explanations of how the code works, the memos account for explanations, in the developers' words, regarding why they implemented a given function in a certain way and which documentation inspired their decision.

4.2 Implementation notes

As illustrated in Figure 2, Veronica has three main components: the Visual Studio Code Extension, the Selenium Standalone Server, and the Google Chrome extension. The developer working on Visual Studio Code can communicate their intent to Veronica either with a voice command or textual input. For natural language processing, it relies on Google Dialogflow[2]. If the user has expressed their intentions vocally, the Visual Studio extension invokes a Speech Recognition API that transforms the audio input into text. Then the Visual Studio extension invokes the Google Dialogflow API, passing the textual input as a parameter, and evaluates it against a set of intents. Once the corresponding intent is executed, the result is displayed textually in the chat and audio. As shown in the Figure, a Speech Synthesis API is invoked by the extension to transform the textual response obtained from the Google Dialogflow into audio to achieve the audio answer functionality.

Furthermore, the Google Chrome extension runs in the background and gathers the URLs of the websites opened in the browser. However, to enable communication between the Visual Code extension and the Google Chrome extension (also implemented by us), an instance of the Google Chrome server runs in the background within a Selenium Standalone Server. In this manner, via the Standalone Server, the Visual Studio Code extension can collect the URLs and ask the developer if they wishes to attach them to the newly created memos. The data exchange between the Visual Studio Code extension with the Standalone Server and between the Server and the Google Chrome extension was achieved using sockets.

5 USER STUDY

Once the tool's implementation was concluded, an usability evaluation was conducted to identify any efficiency problems associated with the various tool's functionalities. These tests comprised a first phase in which the users were asked to complete a set of tasks (listed in Table 1), each focused on specific functionality, and a second phase in which we posed feedback questions during a debriefing session to measure their perception on the tool's usability in a standardized way.

Before the usability test, the participants were asked for consent and were given an introductory questionnaire to gather their programming level of experience, familiarity with Visual Studio Code, and familiarity with chatbots. Additionally, since Veronica's language is English, participants were asked about their English proficiency. Precisely, Table 2 presents the questions asked along with the collected answers. In the usability test, participants were given a Python file opened in Visual Studio Code. All the participants, indeed, were familiar with Python. It contained three functions

[2]https://cloud.google.com/dialogflow/es/docs, last visited on January 19, 2023

Table 1: Tasks used during the validation of Veronica

ID	Assigned task	Success criteria and think-aloud	Max. time
T-01	Start Veronica and ask for help	Veronica has been started and the "help intent" has been triggered	5 mins.
T-02	Create a memo attached to the function "bubbleSort"	The memo has been created (manually or with the bot)	8 mins.
T-03	Search online the algorithms computational complexity	The URL related to the page found has been saved by the bot (not manually)	8 mins.
T-04	Delete the memo that contains a wrong description of the attached code	The memo has been found and deleted with success (manually or with the bot)	8 mins.
T-05	Disable Veronica listening	The listening has been disabled, leaving the rest of the functionalities online	5 mins.

Table 2: Answers to the introductory questionnaire

ID	Questions	Answers
Q-01	How would you assess your level of programming?	beginner (**1**) - intermediate (**5**) - expert (**2**)
Q-02	Have you ever used Visual Studio Code?	no (**0**) - sometimes (**3**) - yes, I use it often (**5**)
Q-03	If yes to the previous question, have you ever used an extension on Visual Studio Code? Did any of the extensions you used have a graphical interface?	no (**1**) - yes, someone in background (**5**) - yes, someone with visible graphical interface (**0**) - yes, both types (**2**)
Q-04	Have you ever interacted with a chatbot?	yes (**3**) - no (**5**)
Q-05	What is your level of English?	A1/A2 (**0**) - B1/B2 (**6**) - C1/C2 (**2**)
Q-06	What programming languages do you know, at least at a basic level?	Python (**8**) - C (**7**) - JavaScript (**5**) - C++ (**3**) - Java (**2**) - Assembly (**1**) - Bash (**1**) - HTML (**1**) - Kotlin (**1**) - PHP (**1**)
Q-07	What browser do you usually use?	Google Chrome (**3**) - Microsoft Edge (**2**) - Brave (**1**) - Mozilla Firefox - (**1**) Safari (**0**)

corresponding to sorting algorithms and a memo wrongly associated with one of those functions. Participants did not receive any instruction on how to use the tool for further understanding the intuitiveness of the presented interface.

5.1 Methodology

Eight participants were recruited from various programming courses in the computer engineering degree at our university. Seven of them self-identified as male, and one self-identified as female. Their average age was 22.7 years (min = 21, max = 27), and they covered different levels of expertise. After the experiment with Veronica, we conducted a debriefing session by asking some feedback questions regarding the experience and use of the extension. We provided the participants with a laptop we configured for the study, and the screen was video recorded. The whole study, including the experiment and the follow-up interview, happened in person and lasted an average of 27 minutes. We audio-recorded and transcribed the interviews.

Finally, by rewatching the video recordings, we evaluated each of the tasks listed above according to the following metrics: (i) **the number of critical errors**: errors that prevent the completion of the task, (ii) **the number of non-critical errors**: errors that enable the fulfillment of the task but incompletely or erroneously, (iii) **completion rate**: percentage of the students that did not make any critical errors, and (iv) **error-free rate**: percentage of students who did not make mistakes.

6 RESULTS

In this section we present the observations that emerged from the tasks' metrics results and we link them to the feedback gathered in the debriefing session.

Table 3: Tasks' metrics

ID	Critical errors	Non-critical errors	Completion rate	Error-free rate
T-01	0	7	1	0.25
T-02	0	6	1	0.5
T-03	1	1	0.875	0.875
T-04	0	1	1	0.875
T-05	0	6	1	0.375

6.1 Tasks' metrics results

Table 3 reports the tasks' metrics. Although there were no critical errors, only half of the participants completed **T-01** (Start Veronica and ask for help). Most tried to activate it through a command in the integrated terminal. Additionally, just one participant completed it smoothly, and the other three completed it almost at the end of the maximum time after trying to execute it from the terminal.

Task **T-02** (Create a memo attached to the function "bubble sort") did not present critical errors. Five participants completed it manually (clicking on the "+" button displayed when hovering the cursor over the code), two used Veronica's chat with textual input, and one used the vocal input. Non-critical errors concerned attempts to create the memo with commands that were not understandable by Veronica.

Task **T-03** (Search online the algorithms computational complexity) had one critical error: when Veronica asked the participant if

he wanted to save the URL, by mistake, he answered negatively. After trying to get back to the browser and then again to Visual Studio Code, Veronica did not ask him again if he wanted to save the URL because it had already been discarded. Apart from this case, all the participants completed the task successfully.

Task **T-04** (Delete the memo that contains a wrong description of the attached code) was completed by all the participants without critical errors. One participant made one non-critical error because instead of deleting the memo, he manually deleted the memo's content. However, then he noticed his mistake and deleted the memo. Three participants deleted the task by giving textual instructions to the bot, while the others did it manually.

Finally, no critical errors were made in completing **T-05** (Disable Veronica listening). However, participants made six non-critical errors, all related to trying to disable Veronica by writing a command in the chat instead of clicking the same button that was used to activate the extension. Nevertheless, in the end, 7 out of the 8 participants succeeded in disabling Veronica.

6.2 Debriefing feedback

The most common piece of feedback among the participants concerned the lack of guidance on which functionalities Veronica provides and how to use them. They wished to see a contextual menu showing a list of available commands upon opening the extension and a help button to be always available for consultation in case they needed guidance on how to respond to Veronica. In the P-02 words, "apart from the functionalities specified in the tasks, I would like to see what else Veronica can do." Similarly, P-04 suggests including "a help module more explanatory; for example, to understand better what it meant when Veronica asked me if I wanted to save changes." Finally, P-08 envisioned that "the first textual message once you open the extension might explain the available features. It is not clear what kind of voice commands to use.".

Furthermore, participants did not understand intuitively that from the moment the extension is activated; it remains in continuous listening. Indeed, they would prefer the listening to be activated or deactivated by the users. P-01 said: "I thought the chat was separated from the voice commands. I did not consider it would synthesize everything I said." P-04 commented: "I suggest adding a microphone icon because I was unsure if the microphone was recording when I opened the extension." Furthermore, P-06 proposes: "I would expect more for a kind of push-to-talk mechanism, something optional. In this way, when there is ambient noise, I can decide whether I want to be heard by Veronica."

Participants struggled to find out how to enable Veronica. The button in the status bar on the bottom left side of the IDE was not visible enough for them. In this sense, P-03 commented: "Once I understood how to activate the extension, I could do everything very intuitively and smoothly. The only challenging thing was to find how to activate it." P-02 said: "It took me a while to understand where to activate Veronica. I was not expecting it to be at the bottom."

However, participants appreciated that Veronica automatically gathered the URL they consulted and linked it to their code as a new demo. In P-03 words: "It is very convenient to save your last search on the browser. This way, you do not forget where you found the documentation, and this feature speeds up the code development." Furthermore, as previously described, practically all the participants understood and used this feature correctly. Indeed, the feedback provided by P-04 is in line with the design goals we established for this tool. He commented:

> *I find the tool helpful, especially for those starting to program. With the memos, someone who sees an implemented piece of code for the first time can say, 'fine, now I see what it does,' which is nice because each person has a programming style.*

Similarly, P-02 said: 'I like that having the self-explanatory comments pinned to the code instead of having them interleaved with the code keeps the files more navigable and cleaner", which again align with the main goal of Veronica.

7 CONCLUSIONS AND FUTURE WORK

This paper introduces Veronica, a Visual Studio Code extension in the form of a conversational agent, to help developers *self-explain* their development process and thoughts as they arise. Veronica does this through "memos," which can be added by text and voice. In addition, Veronica supports the process of collecting developers' reasoning by tracking the web pages that developers visit while programming. This way, Veronica can propose the developer save, in a memo, the URLs of the relevant web pages they visit.

After the design and first development of Veronica, the usability study we performed with eight university students showed that the tool has the potential to support the overall goal of producing documentation with self-explanatory insights, which has the potential to become a valuable asset for them and other developers. The usability study also highlights some issues in the current implementation, such as the difficulty of activating the chatbot or the need to provide more help for the commands that Veronica accepts.

Future work on Veronica will start with solving the usability issues that have emerged so far, which mainly pertain to the specific implementation of the tool. In addition, we would like to work toward two different yet complementary directions. First, we want to perform a more extensive, in-the-wild evaluation of the tool with developers of varied expertise to assess its capabilities of promoting documentation based on self-explanations. Results from this study will enable us to identify if the adopted mechanisms (i.e., the memos, the chat-based interaction, and the URL tracking from a web browser) are enough or if more advanced strategies are needed. Lastly, we want to use Veronica as a starting point to experiment with more direct strategies to elicit the recording of self-explanations, for instance, through periodical prompts and rich-text memos.

ACKNOWLEDGMENTS

The authors would like to thank Gianluca Filippo Fasulo, who contributed to the development and evaluation of Veronica as part of his master's thesis. This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

REFERENCES

[1] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. 2020. MSRBot: Using bots to answer questions from software repositories. *Empirical Software Engineering* 25, 3 (01 May 2020), 1834–1863. https://doi.org/10.1007/s10664-019-09788-5

[2] Merav Allouch, Amos Azaria, and Rina Azoulay. 2021. Conversational Agents: Goals, Technologies, Vision and Challenges. *Sensors* 21, 24 (Dec 2021), 8448. https://doi.org/10.3390/s21248448

[3] Kiran Bisra, Qing Liu, John C. Nesbit, Farimah Salimi, and Philip H. Winne. 2018. Inducing Self-Explanation: a Meta-Analysis. *Educational Psychology Review* 30, 3 (01 Sep 2018), 703–725. https://doi.org/10.1007/s10648-018-9434-x

[4] Bruno da Silva, Chloe Hebert, Abhishu Rawka, and Siriwan Sereesathien. 2020. Robin: A Voice Controlled Virtual Teammate for Software Developers and Teams. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 789–791. https://doi.org/10.1109/ICSME46990.2020.00092

[5] H. Happel and W. Maalej. 2009. From work to word: How do software developers describe their work?. In *2009 6th IEEE International Working Conference on Mining Software Repositories. MSR 2009.* IEEE Computer Society, Los Alamitos, CA, USA, 121–130. https://doi.org/10.1109/MSR.2009.5069490

[6] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1019–1028. https://doi.org/10.1145/1753326.1753478

[7] Andrew Head, Elena L. Glassman, Björn Hartmann, and Marti A. Hearst. 2018. Interactive Extraction of Examples from Existing Code. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173659

[8] Michelle Ichinco and Caitlin Kelleher. 2015. Exploring novice programmer example use. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 63–71. https://doi.org/10.1109/VLHCC.2015.7357199

[9] Kyungbin Kwon and David H. Jonassen. 2011. The Influence of Reflective Self-Explanations on Problem-Solving Performance. *Journal of Educational Computing Research* 44, 3 (2011), 247–263. https://doi.org/10.2190/EC.44.3.a

[10] Carlene Lebeuf, Margaret-Anne Storey, and Alexey Zagalsky. 2018. Software Bots. *IEEE Software* 35, 1 (2018), 18–23. https://doi.org/10.1109/MS.2017.4541027

[11] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. 2013. What help do developers seek, when and how?. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 142–151. https://doi.org/10.1109/WCRE.2013.6671289

[12] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. 2019. Unakite: Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 67–80. https://doi.org/10.1145/3332165.3347908

[13] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 31 (sep 2014), 37 pages. https://doi.org/10.1145/2622669

[14] Michael F. McTear. 2017. The Rise of the Conversational Interface: A New Kid on the Block?. In *Future and Emerging Trends in Language Technology. Machine Learning and Big Data*, José F Quesada, Francisco-Jesús Martín Mateos, and Teresa López Soto (Eds.). Springer International Publishing, Cham, 38–49.

[15] Stephen Oney and Joel Brandt. 2012. Codelets: Linking Interactive Documentation and Example Code in the Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2697–2706. https://doi.org/10.1145/2207676.2208664

[16] Chris Parnin and Robert DeLine. 2010. Evaluating Cues for Resuming Interrupted Programming Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 93–102. https://doi.org/10.1145/1753326.1753342

[17] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 531–537. https://doi.org/10.1145/3287324.3287374

[18] Peter Robe and Sandeep Kaur Kuttal. 2022. Designing PairBuddy—A Conversational Agent for Pair Programming. *ACM Trans. Comput.-Hum. Interact.* 29, 4, Article 34 (may 2022), 44 pages. https://doi.org/10.1145/3498326

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

[20] Emad Shihab, Stefan Wagner, Marco A. Gerosa, Mairieli Wessel, and Jordi Cabot. 2022. The Present and Future of Bots in Software Engineering. *IEEE Software* 39, 5 (2022), 28–31. https://doi.org/10.1109/MS.2022.3176864

[21] Yusuke Shinyama, Yoshitaka Arahori, and Katsuhiko Gondow. 2018. Analyzing Code Comments to Boost Program Comprehension. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 325–334. https://doi.org/10.1109/APSEC.2018.00047

[22] Margaret-Anne Storey and Alexey Zagalsky. 2016. Disrupting Developer Productivity One Bot at a Time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 928–931. https://doi.org/10.1145/2950290.2983989

[23] Camilo Vieira, Alejandra J. Magana, Michael L. Falk, and R. Edwin Garcia. 2017. Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education. *ACM Trans. Comput. Educ.* 17, 4, Article 17 (aug 2017), 21 pages. https://doi.org/10.1145/3058751

[24] Camilo Vieira, Alejandra J. Magana, Anindya Roy, and Michael L. Falk. 2019. Student Explanations in the Context of Computational Science and Engineering Education. *Cognition and Instruction* 37, 2 (2019), 201–231. https://doi.org/10.1080/07370008.2018.1539738

[25] Zhendong Wang, Yi Wang, and David Redmiles. 2022. From Specialized Mechanics to Project Butlers: The Usage of Bots in Open Source Software Development. *IEEE Software* 39, 5 (2022), 38–43. https://doi.org/10.1109/MS.2022.3180297

[26] Joseph J. Williams and Tania Lombrozo. 2010. The Role of Explanation in Discovery and Generalization: Evidence From Category Learning. *Cognitive Science* 34, 5 (2010), 776–806. https://doi.org/10.1111/j.1551-6709.2010.01113.x