

# Can a Chatbot Support Exploratory Software Testing? Preliminary Results

Rubens Copche<sup>1</sup>, Yohan Duarte<sup>2</sup>, Vinicius Durelli<sup>3</sup>, Marcelo Medeiros Eler<sup>4</sup>  
and Andre Takeshi Endo<sup>2</sup> <sup>a</sup>

<sup>1</sup>Grupo TCM, Assis, Brazil

<sup>2</sup>Computing Department, Federal University of São Carlos, São Carlos, Brazil

<sup>3</sup>Federal University of Sao Joao Del Rei, Sao Joao Del Rei, Brazil

<sup>4</sup>University of Sao Paulo (USP), Sao Paulo, Brazil

**Keywords:** Software Testing, Bots, Testing Strategies, Human Testers, Bots for Software Engineering.

**Abstract:** Tests executed by human testers are still widely used in practice and fill the gap left by limitations of automated approaches. Among the human-centered approaches, exploratory testing is the *de facto* approach in agile teams. Although it is focused on the expertise and creativity of the tester, the activity of exploratory testing may benefit from support provided by an automated agent that interacts with human testers. We set out to develop a chatbot named BotExpTest, specifically designed to assist testers in conducting exploratory tests of software applications. We implemented BotExpTest on top of the instant messaging social platform Discord; this version includes functionalities to report bugs and issues, time management of test sessions, guidelines for app testing, and presentation of exploratory testing strategies. To assess BotExpTest, we conducted a user study with six software engineering professionals. They carried out two sessions performing exploratory tests along with BotExpTest. Participants revealed bugs and found the experience to interact with the chatbot positive. Our analyses indicate that chatbot-enabled exploratory testing may be as effective as similar approaches and help testers to uncover different bugs. Bots are shown to be valuable resources for Software Engineering, and initiatives like BotExpTest may help to improve the effectiveness of testing activities like exploratory testing.


## 1 INTRODUCTION

Due to their practical use and broad applicability, a myriad of bots that vary in complexity have been developed and deployed in widely varying contexts. Over the last decade, technological advancements have enabled *bots* to play an ever increasingly important role in many areas, particularly in software development. This emerging technology has garnered the interest of both software development researchers and practitioners, as bots can serve as human assistants for a variety of software development-related tasks. In this particular context, bots that provide support for specific aspects of software development, such as keeping project-related dependencies up-to-date, are referred to as *devbots* (Erlenhov et al., 2019).

Recent developments in machine learning and natural language processing have led to the creation of bots that provide more user-friendly experiences.

Bots that harness natural language processing capabilities to provide more intuitive and user-friendly experiences are commonly referred to as *chatbots*. As their name implies, chatbots are software programs designed to replicate human-like conversations or interactions with users (Shawar and Atwell, 2007).

As mentioned, bots have been utilized to support various software engineering tasks (Storey and Zagalsky, 2016; Paikari et al., 2019; Sharma et al., 2019; Erlenhov et al., 2020; Okanović et al., 2020). We set out to examine how chatbots can be leveraged to assist testers throughout the testing process. Specifically, we posit that chatbots are well-suited for providing assistance to testers throughout the execution of Exploratory Testing (ET) tasks. ET is an approach to software testing that entails carrying out a series of undocumented testing sessions to uncover faults. ET leverages the skills and creativity of testers while they explore the system under test (SUT), and the knowledge gained during ET sessions is then used to

<sup>a</sup>  <https://orcid.org/0000-0002-8737-1749>

further refine the exploration. Hence, ET is a goal-focused, streamlined approach to testing that allows for flexibility in test design and keeps testers engaged throughout the testing process (Bach, 2003; Kaner et al., 1993; Souza et al., 2019). Owing to these benefits, ET has been gaining traction as a complement to fully scripted testing strategies (ISTQB, 2018): when combined with automated testing, ET has the potential to increase test coverage and uncover edge cases. In fact, there is evidence suggesting that ET can be equally or even more effective than scripted testing in practical situations (Ghazi et al., 2017).

In practice, before ET sessions, testers engage with other testers and developers to gather project-related information. However, due to the complexity of most software projects, it becomes impractical to collect all relevant information beforehand. As a result, interruptions that arise during ET sessions for the purpose of gathering additional information can disrupt the flow. One potential solution to overcome this issue is to employ a chatbot that assists testers during ET sessions, providing guidance on the selection of input data for achieving different levels of exploration. Furthermore, the chatbot can encourage critical thinking and enable testers to make informed decisions. To the best of our knowledge, this research is the first foray into the potential of a chatbot in maximizing the effectiveness of ET.

This paper introduces BotExpTest, a chatbot designed to assist testers during ET sessions. BotExpTest was built on top of the Discord platform and includes features tailored to managing ET sessions and reporting bugs and issues. Additionally, it incorporates features aimed at enhancing testers' ability to gain insights that can be utilized to delve further into the exploration of the SUT.<sup>1</sup> To evaluate how BotExpTest performs "in the wild", we conducted a user study with six practitioners. The results from the user study would seem to indicate that BotExpTest was able to help the participants to uncover several bugs. Moreover, the participants expressed a positive opinion about the experience and held an optimistic view regarding the potential future adoption of the tool.

## 2 RELATED WORK

*Chatbots* are becoming increasingly popular in the software development domain because they can be

<sup>1</sup>The development and evaluation of the current version of BotExpTest took place prior to the release of ChatGPT and other large language models (LLMs). However, in future work, we delve into the potential integration of these advanced technologies.

very versatile. In this context, *bots* are frequently classified based on their capacity of supporting different activities such as code review, tests, bug fixing, verification and deployment (Storey and Zagalsky, 2016). Storey et al. (Storey et al., 2020) surveyed developers and researchers to identify in which situations they use *bots* to support software engineering. Here is what they found: to search and to share information, to extract and to analyze data, to detect and to monitor events, to communicate in social media, to connect stakeholders and developers, to provide feedback, and to recommend individual or collaborative tasks associated with software development.

Many studies have proposed *bots* for software development activities. *Performobot* is a chatbot-based application that helps in planning, executing and reporting the results of tasks related to load and performance testing (Okanović et al., 2020). *Smart Advisor* is an intelligence augmentation *bot* that helps developers with project specifics by employing domain and knowledge modeling and in-process analytics to automatically provide important insights and answer queries using a conversational and interactive user interface (Sharma et al., 2019). *Repairnator* is a program repair *bot* that creates software patches and provides an explanation for each bug fixed using natural language as a human collaborator would do (Monperus, 2019). *Tutorbot* uses machine learning to retrieve relevant content, guiding software engineers in their learning journey and helping them keep pace with technology changes (Subramanian et al., 2019).

To the best of our knowledge, there is no specific chatbot devised to help testers to conduct ET. In fact, a study conducted in Estonia and Finland found out that only 25% of the software testing professionals apply ET with some tool support. Mind mapping tools are the most frequently used software, but testers also use text editors, spreadsheets and even actual pen and paper, in addition to checklists and paper notes (e.g. post-its) (Pfahl et al., 2014). In this context, Copche et al. (Copche et al., 2021) introduced a specific kind of mind map called *opportunity map* (OM) as a way to improve the ET of mobile apps. The authors conducted a study that compares OM-based ET with a traditional session-based approach (baseline).

There are some tools to directly support activities involved in ET. For instance, Leveau et al. (Leveau et al., 2020) designed a tool called *Test Tracker* to prevent testers from running tests that have already been executed so they can run more diversified test sessions to further explore the SUT.

There have been some attempts to integrate ET with automated approaches. For instance, Shah et al. (Shah et al., 2014) proposed an hybrid approach

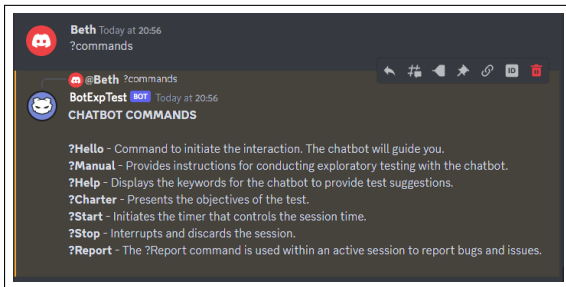


Figure 1: BotExpTest replies its main commands.

that combines the strengths of ET and scripted testing (ST). Considering the strengths of ET, which includes the application of domain knowledge and the observation of the system behavior for rapid feedback, the hybrid process allows the testers to explore the SUT freely and to utilize their intuitions and experience in identifying defects before writing test scripts.

### 3 BotExpTest

This section presents the design and main features of a chatbot we developed to support ET. We set out by investigating existing work about ET, its core practices and envisioned how a chatbot would help the tester to conduct more effective ET sessions. To validate these ideas, we implemented BotExpTest. Figure 1 shows the example of an interaction with BotExpTest: tester Beth types `?commands` and then BotExpTest shows all commands accepted.

#### 3.1 Implementation

As instant messaging platforms are widely adopted and are today an essential part of software projects, we opted to develop BotExpTest on top of them. For this first release, we settled on using the Discord platform. Discord provides an open source platform with highly configurable features for users and bots. BotExpTest is implemented as a Node.js project; it has 22 classes and around 1.3K lines of JavaScript code. It takes advantage of the Discord API to capture interactions from testers in the chat, as well as to generate its own messages. To make the ET process auditable, all messages exchanged between the chatbot and the testers are recorded in a MongoDB database. BotExpTest is available as an open source project at:

<https://github.com/andreendo/botexpTest>

Figure 2 presents an overview of the BotExpTest architecture. The interaction starts with the tester writing a message (command) to BotExpTest via Dis-

cord (step 1). The message passes through the Discord Developer Portal, which is then accessible by means of an API (steps 2-3). Up to this point, BotExpTest interprets the message typed by the tester and reacts by sending a reply (steps 4-5). Finally, the BotExpTest's response is shown to the tester and new interactions may occur. BotExpTest may also be the one that starts the interaction.

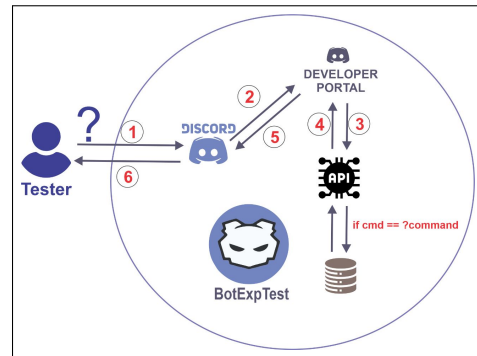


Figure 2: Architecture of BotExpTest.

#### 3.2 Main Features

During the first interaction between tester and BotExpTest, the first reaction of the chatbot is to present itself, giving some pieces of information about how to perform the next steps in the ET session. As a convention, testers begin an interaction with BotExpTest using a message that starts with '?'. Adhering to this convention can be beneficial in scenarios where several testers are communicating with each other in a chat, as it indicates when testers intend to engage with the bot. For this version, the features implemented by BotExpTest were elicited, prioritized and implemented; they are described next.

**Description of the Test Procedure.** Using command `?manual`, BotExpTest shows a step-by-step description about how the test sessions are organized and should be conducted, as well as the main features provided to tester by the chatbot.

**Charters.** In ET, *charters* are used to organize the tests and represent the goals that are supposed to be achieved in a test session. BotExpTest provides an interface to set up the test charters and are available to testers by using the message `?charter`. Besides the charter name, app name, and the goals description, it is also possible to attach images and other files related to the charter.

**Time Management of Testing Sessions.** In ET, testing sessions are conducted within a limited time frame; usually, testers need to keep track of time. As the tester is constantly interacting in the chat, BotExpTest reminds her about the remaining time in the

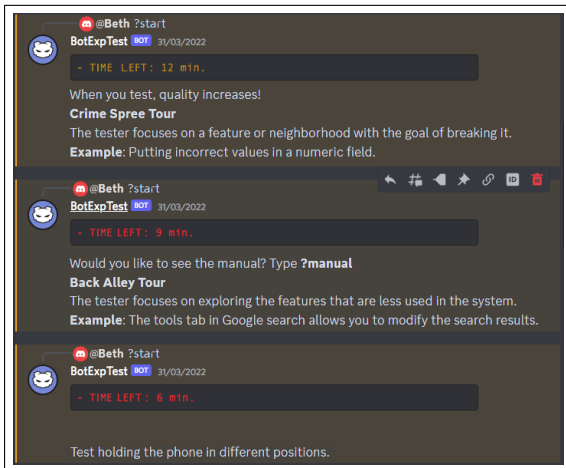


Figure 3: Time alerts and suggestions.

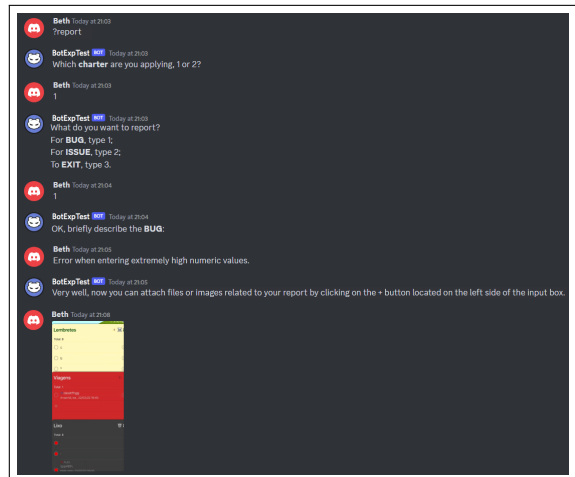


Figure 4: Reporting a bug.

session, from time to time. To signal the start of a session, the tester should type the command `?start`. BotExpTest then asks the time limit and starts to monitor the time elapsed in the session. During the time range of the session, BotExpTest keeps track of all interactions that occurred. Figure 3 shows how the time alerts are presented to the tester.

**Bug and Issue Reporting.** The identification of bugs and issues are the main outcomes of a test session. To avoid using other tools for this task, BotExpTest registers occurrences of bugs or issues. To do so, the tester types command `?report`; this task is exemplified in Figure 4. The tester interacts with BotExpTest so that the charter, type (bug or issue), a detailed description and potential attachments (e.g., screenshots of the bug) are provided. The current version only stores the bug report, but it is possible in future to integrate it with external tools like GitHub, Jira or Azure DevOps.

**Curated Knowledge About Exploratory Testing.** The idea of this feature is for BotExpTest to have a curated list of resources about the use of ET techniques. In future, BotExpTest could be fine-tuned for specific projects so that testers are better equipped to conduct exploratory tests. By typing one of the options presented after command `?help`, BotExpTest shows a detailed explanation about the concept and how to apply it during the tests. For some options, the chatbot replies with questions. We anticipate that this feature may help the testers to gain more insights and execute more effective tests.

Currently, BotExpTest provides resources related to three main groups. Group (i) brings well-known testing criteria that may help the tester to design black-box tests. For example, classical criteria like equivalence partition and boundary-value analysis are presented. Group (ii) is composed of strategies for ET

that are well-known and have been adopted in the literature (Micallef et al., 2016; Whittaker, 2009). For example, *Bad Neighborhood Tour* instructs the tester to revisit buggy parts of software since bugs tend to cluster together. Finally, Group (iii) contains guidelines for mobile app testing. For example, there are several test scenarios related to specific characteristics of mobile apps like network connections, geolocation, Bluetooth, camera, and UI events (scrolling, swipe, etc).

**Active Suggestions.** During a test session, BotExpTest can actively start an interaction with the tester. The chatbot can present some piece of information obtained from the curated knowledge about exploratory testing. We believe that actively interacting with the tester would increase the engagement with the tests. For the current version, the decision about the timing and the information provided is made randomly. In future, we expect that BotExpTest could be evolved to make a more informed decision about the interaction needed at a specific point of time.

## 4 USER STUDY

This section describes an empirical study conducted with the aim to provide an initial evaluation of BotExpTest. To this end, we posed the following research questions (RQs):

- RQ1: How is the interaction with BotExpTest during the exploratory testing?
- RQ2: How do the participants perform with respect to the detection of bugs?
- RQ3: How do the participants perceive BotExpTest?

To answer these questions, we conducted a user study with six participants that were asked to use BotExpTest to support the ET of a mobile app named *Reminders*. All participants work in the industry and have experience with software development and testing. We adopted the app and related charters that were openly available from Copche et al. (Copche et al., 2021); the rationale here is to make some analyses concerning similar approaches. We used the same app version, charters and length of test sessions. To collect the needed data and observe the participants, we set up a computer with screen recording, and mirroring the mobile device running the app under test. Each participant was invited to use this computer in order to perform the tasks of the study. Figure 5 illustrates the testing environment used by the participants; the app is shown on the left, while the Discord UI (along with BotExpTest) is presented on the right.

We provided the participants with detailed instructions about how to perform the testing tasks. They were instructed to strictly follow the charter, report any bug or issue identified, and the time management of the session was supported by BotExpTest. All test sessions were recorded and the participants could think aloud about the tasks being carried out. After the introduction, the study was divided into two parts: **Training**, this session lasted approximately 15 minutes and served as an introduction to the usage of the chatbot. This allowed participants to become acquainted with BotExpTest and Discord, allowing them to experiment with possible interactions and explore the supported commands.

**Test Sessions**, this iteration took approximately half an hour, in which two test sessions with 15 minutes each occurred. Over the course of these two test sessions, the chatbot-enabled ET took place.

All data was then retrieved and analyzed. To answer RQ1, we looked at the interactions that occurred (messages exchanged) between the participant and BotExpTest. We called *Active Interactions* the ones started by BotExpTest, while *Reactive Interactions* are responses to the tester’s inquiries.

As for RQ2, we analyzed and cataloged the bugs reported by participants. In particular, we cross-checked the bugs herein reported with the ones uncovered in Copche et al. (Copche et al., 2021). As an initial analysis, we intended to assess whether a chatbot-enabled ET can detect a different set of bugs with respect to similar approaches like baseline and OM (see Section 2).

We answered RQ3 with a Likert-scale survey that intends to understand the perception of participants. The questions are divided into (i) how easy is to interact and use BotExpTest, (ii) whether the user interface

is adequate for the proposed functionalities, and (iii) understanding the participants’ perception about the effectiveness of chatbot-enabled ET. There is also an open question for comments and suggestions.

**Threats to Validity.** The study contained a limited number of participants. We recognize that the results do not generalize and further studies with more participants and profile diversity are required. We opted for a limited group of experienced professionals to obtain the initial feedback from potential end users of BotExpTest. The study was based on an app, and the comparison with similar approaches was not direct in the same controlled experiment.

## 5 ANALYSIS OF RESULTS

**RQ1: Interactions with BotExpTest.** Table 1 shows the number of active and reactive interactions of both the chatbot and the participants; the values are also divided between training and test sessions. Overall, BotExpTest produced 581 interactions (121 in training and 460 in the test sessions), while the six participants had 496 interactions (144 in training and 352 in the test sessions).

Table 1: Number of interactions (int.).

TRAINING		TEST SESSIONS	
BotExpTest		BotExpTest	
Reactive int.	107	Reactive int.	340
Active int.	14	Active int.	120
<b>Total</b>	<b>121</b>	<b>Total</b>	<b>460</b>
Participants		Participants	
Reactive int.	37	Reactive int.	262
Active int. (acpt)	100	Active int. (acpt)	90
Active int. (inv)	7	Active int. (inv)	0
<b>Total</b>	<b>144</b>	<b>Total</b>	<b>352</b>

During training, BotExpTest was proportionally more reactive (reactive/active – 107/14) and the participants were more active (37/100); they also typed 7 invalid commands. The most typed commands were related to the chatbot usage (?charter, ?commands, ?help, ?manual) and software testing resources. We observed that the participants were more active in the interactions because they were focused on figuring out how to use BotExpTest.

As for test sessions, BotExpTest was still more reactive (340/120) but proportionally had more active interactions. This occurred due to the reporting of bugs and issues (it asks actively for more pieces of information). This fact also impacted the participants’ interactions: they were more reactive (262/90) and did not type any invalid command. The most typed commands were related to bug and issue reporting

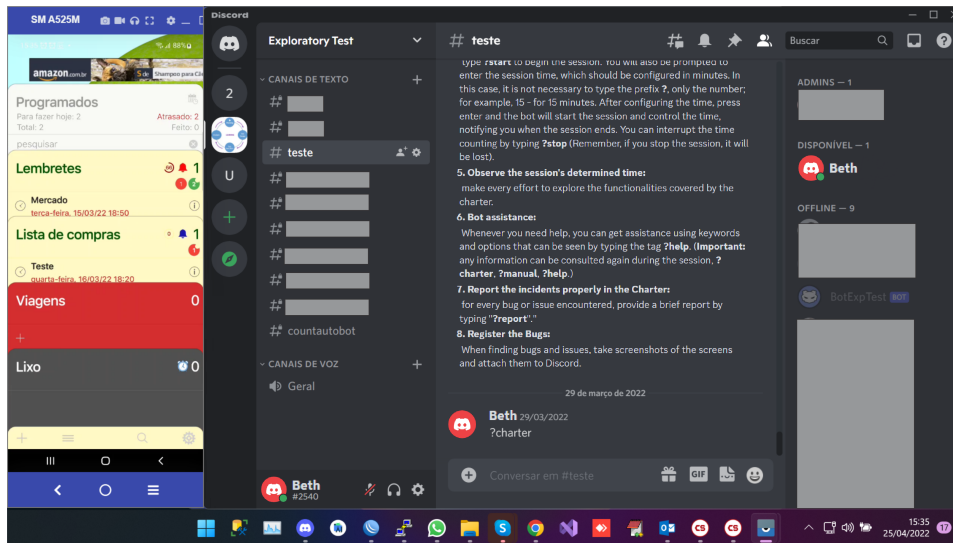


Figure 5: Setup used by the participants.

(e.g., ?report) and management of the test sessions (e.g., ?start, ?charter). We also observed that participants were focused on testing the app and this also limited the interactions with the chatbot.

*Answer to RQ1:* We observed reasonable interactions between the participants and BotExpTest. As the participants were exploring the chatbot in training, they had more active interactions. On the other hand, the interactions were more reactive and limited in the test sessions due to the time spent with exploratory testing of the app itself and reporting bugs and issues.

**RQ2: Bugs.** The six participants reported 31 bugs. The most effective participant uncovered nine bugs, while one of them reported three bugs. On average, the participants detected 5.2 bugs (median: 5). The distribution of bugs detected per participant can be seen in Figure 6, the boxplot/violin in the middle.

Figure 6 also shows the results for the baseline and OM approaches (Copche et al., 2021). Observe that the average and median values of BotExpTest are slightly greater than baseline (avg: 2.7, median: 3) and OM (avg: 4.3, median: 4). The range of values is also smaller, so BotExpTest produced a more uniform performance of participants. Due to the differences of samples and participants' experience, these results may not be generalized. One may argue that the bug detection capability of BotExpTest is at least comparable to approaches without any support (i.e., baseline) or that adopted some supporting artifact (i.e., OM).

Figure 7 shows a Venn diagram with the unique bugs detected by the approaches, numbers between parentheses are bugs tracked to specific suggestions

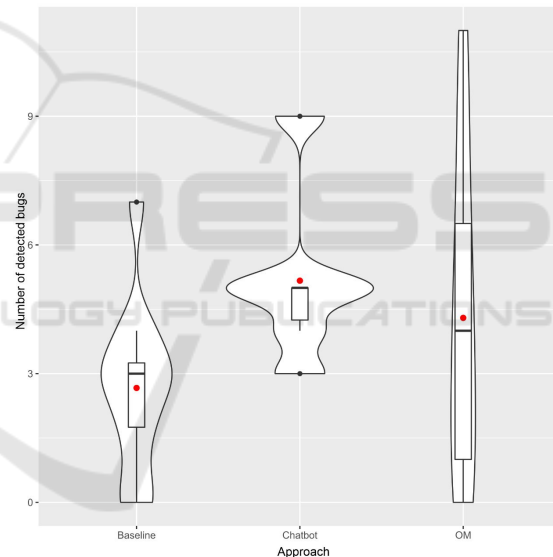


Figure 6: Number of bugs detected by the participants.

of the approach. Out of 31 bugs reported by the participants using BotExpTest, 21 were unique since some reported the same bug. Eight bugs have been uncovered in the Copche et al. study (1 by baseline, 3 by OM and 4 by both), but 13 yet-unknown bugs were detected in this study. We were able to map three out of these 13 bugs to specific insights provided by the chatbot.

*Answer to RQ2:* The participants were capable of uncovering an average of 5.2 bugs using BotExpTest. Their performance is comparable to similar approaches evaluated in the literature. Furthermore, BotExpTest supported the participants in detecting 13 previously unknown bugs.

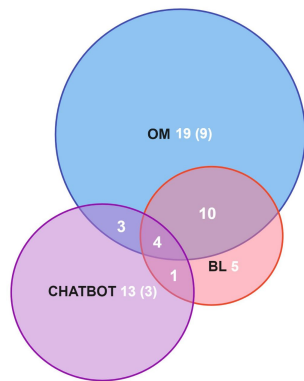


Figure 7: Unique bugs.

**RQ3: Participants' Perception.** For the part (i) of questions, we asked about the easiness of finding information, whether proper instructions are provided, and usability in general. All participants agreed or strongly agreed that BotExpTest is easy to use and interact. As for part (ii), the questions asked about specific features like bug/issue reporting and time management of test sessions. Most participants strongly agreed or agreed that the user interface for those features is adequate. In particular, the bug/issue reporting was unanimously well-evaluated (all strongly agreed).

For the last part of questions, the responses indicated that participants would use a chatbot in similar tasks, and they perceived more organized test sessions. They also thought that BotExpTest helped them to find more bugs. We also asked if the chatbot helped them to understand new concepts of software testing, and whether the suggestions made by BotExpTest were helpful; all participants strongly agreed or agreed with those statements.

From the open question and our observations, we draw the following thoughts. Participants believed that the chatbot helped to shorten the time spent with process tasks, saving more time to test the app. BotExpTest worked as a rich and centralized source of testing information; participants sometimes used the message history to revisit decisions and bugs detected. One suggested that it could support novice programmers testing their software, and another mentioned that it could help teams without QAs. Finally, there were suggestions to add support for other testing tasks, like managing test scripts, tracking the status of test executions, and communication with stakeholders.

*Answer to RQ3:* The participants perceived BotExpTest as a valuable resource while performing exploratory testing. Overall, the participants' perceptions were positive concerning the features, the ease of interaction, and testing resources.

## 6 RESEARCH OPPORTUNITIES

This section delves into research opportunities identified while conducting this study. The following features have the potential to enhance the synergy between the tester and bots, thereby making the testing process more effective, systematic, and traceable.

**Sophisticated Conversational Capabilities.** By emulating more human-like interactions, chatbots can significantly enhance engagement with testers. We hypothesize that this increased engagement will lead to a more thorough exploration of the SUT, thereby helping to prevent a plateau in exploration. Advancements in natural language processing (e.g., ChatGPT, Bard, and Llama 2) and machine learning are key to propelling chatbots towards these more nuanced interactions. Thus, we believe that retrofitting our chatbot with LLM capabilities will foster a more immersive and effective testing experience.

**Access to a Wider Collection of Curated Software Testing Resources.** Currently, BotExpTest supports only a limited set of software testing resources. As mentioned, to allow for more complex and knowledge-intensive exploratory tasks, it is essential to upgrade BotExpTest with an LLM. Additionally, we could integrate the LLM system with external software testing knowledge. This enhancement would ensure greater factual consistency, enhance response reliability, and address the "hallucination" issue commonly encountered in LLM-generated responses. Moreover, such an upgrade would allow our chatbot to better match the tester's proficiency level by tailoring the technical level of responses to align with the tester's knowledge. Achieving this would entail retrofitting our chatbot with LLM capabilities and employing retrieval-augmented generation (RAG), which is an approach that refines LLM-generated responses by grounding them on external software testing information sources, supplementing the LLM's inherent knowledge representation.

**Communication Layer with the SUT.** To provide better feedback and make smarter suggestions for the tester, BotExpTest would be able to interact with the SUT. We surmise that these interactions will be possible by implementing a communication layer with the SUT. This layer will leverage existing testing technologies like code coverage, mocking, E2E test frameworks, virtualization, and so on. An example is to include the ability of observing the SUT (using e.g. monitoring or dynamic analyses as in (Leveau et al., 2020)). This ability would be used to feed the chatbot and help the tester making more educated insights. Another direction is to provide screenshots or videos of the tester's interactions with the SUT and make use of AI solutions.

**Integration-Based Intelligence.** By implementing the aforementioned features, it is possible to improve the overall intelligence of chatbot and make it part of a more integrated testing environment. To make it less reactive, BotExpTest could evolve to learn from past test explorations, or adopt automated initiatives like GPTDroid (Liu et al., 2023). Pairing human developers with a bot like GitHub CoPilot (i.e., pair programming) has been investigated (Imai, 2022; Moradi Dakhel et al., 2023). In a parallel vein, this pivotal feature will facilitate the establishment of a pair-programming like strategy for software testing, thereby leveraging both human and bot capabilities.

## ACKNOWLEDGEMENTS

AT Endo is partially supported by grant #2023/00577-8, São Paulo Research Foundation (FAPESP). Y Duarte is supported by grant FAPESP #2022/13469-6, and CAPES grant 88887.801592/2023-00.

## REFERENCES

- Bach, J. (2003). Exploratory testing explained.
- Copche, R., Souza, M., Villanes, I. K., Durelli, V., Eler, M., Dias-Neto, A., and Endo, A. T. (2021). Exploratory testing of apps with opportunity maps. In *XX Brazilian Symposium on Software Quality*, pages 1–10.
- Erlenhov, L., De Oliveira Neto, F. G., and Leitner, P. (2020). An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective. In *The 28th ACM Symposium on the Foundations of Software Engineering*, pages 445–455.
- Erlenhov, L., de Oliveira Neto, F. G., Scandariato, R., and Leitner, P. (2019). Current and future bots in software development. In *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 7–11. IEEE.
- Ghazi, A. N., Garigapati, R. P., and Petersen, K. (2017). Checklists to support test charter design in exploratory testing. In *International Conference on Agile Software Development*, pages 251–258. Springer.
- Imai, S. (2022). Is github copilot a substitute for human pair-programming? an empirical study. In *The ACM/IEEE 44th International Conference on Software Engineering: Companion Proc.*, page 319–321.
- ISTQB (2018). *Worldwide Software Testing Practices Report Message from the President Executive Summary Survey Questions and Analysis*.
- Kaner, C., Nguyen, H. Q., and Falk, J. L. (1993). Testing computer software.
- Leveau, J., Blanc, X., Réveillère, L., Falleri, J., and Rouvoy, R. (2020). Fostering the diversity of exploratory testing in web applications. In *IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 164–174.
- Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Che, X., Wang, D., and Wang, Q. (2023). Chatting with GPT-3 for zero-shot human-like mobile automated GUI testing. *CoRR*, abs/2305.09434.
- Micallef, M., Porter, C., and Borg, A. (2016). Do exploratory testers need formal training? an investigation using HCI techniques. In *9th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 305–314.
- Monperrus, M. (2019). Explainable software bot contributions: Case study of automated bug fixes. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 12–15. IEEE.
- Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., and Jiang, Z. M. J. (2023). Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software*, 203:111734.
- Okanović, D., Beck, S., Merz, L., Zorn, C., Merino, L., van Hoorn, A., and Beck, F. (2020). Can a chatbot support software engineers with load testing? approach and experiences. In *The ACM/SPEC International Conference on Performance Engineering*, pages 120–129.
- Paikari, E., Choi, J., Kim, S., Baek, S., Kim, M., Lee, S., Han, C., Kim, Y., Ahn, K., Cheong, C., et al. (2019). A chatbot for conflict detection and resolution. In *1st International Workshop on Bots in Software Engineering (BotSE)*, pages 29–33.
- Pfahl, D., Yin, H., Mäntylä, M. V., and Münch, J. (2014). How is exploratory testing used? a state-of-the-practice survey. In *The 8th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–10.
- Shah, S. M. A., Gencel, C., Alvi, U. S., and Petersen, K. (2014). Towards a hybrid testing process unifying exploratory testing and scripted testing. *Journal of software: Evolution and Process*, 26(2):220–250.
- Sharma, V. S., Mehra, R., Kaulgud, V., and Podder, S. (2019). A smart advisor for software delivery—a bot framework for awareness, alerts and advice. In *The 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 22–23.
- Shawar, B. A. and Atwell, E. (2007). Chatbots: are they really useful? In *Ldv forum*, volume 22, pages 29–49.
- Souza, M., Villanes, I. K., Dias-Neto, A. C., and Endo, A. T. (2019). On the exploratory testing of mobile apps. In *The IV Brazilian Symposium on Systematic and Automated Software Testing*, pages 42–51.
- Storey, M. A., Serebrenik, A., Rosé, C. P., Zimmermann, T., and Herbsleb, J. D. (2020). Botse: Bots in software engineering (dagstuhl seminar 19471). In *Dagstuhl Reports*, volume 9.
- Storey, M. A. and Zagalsky, A. (2016). Disrupting developer productivity one bot at a time. In *The 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931.
- Subramanian, V., Ramachandra, N., and Dubash, N. (2019). Tutorbot: contextual learning guide for software engineers. In *IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, pages 16–17.
- Whittaker, J. A. (2009). *Exploratory software testing: tips, tricks, tours, and techniques to guide test design*. Pearson Education.