

Community Smell Detection and Refactoring in SLACK: The CADOCS Project

Gianmario Voria*, Viviana Pentangelo*, Antonio Della Porta*, Stefano Lambiase*,
Gemma Catolino†, Fabio Palomba*, Filomena Ferrucci*

*Software Engineering (SeSa) Lab, Department of Computer Science - University of Salerno, Salerno, Italy

†Jheronimus Academy of Data Science – Tilburg University, 's-Hertogenbosch, Netherlands

Abstract—Software engineering is a human-centered activity involving various stakeholders with different backgrounds that have to communicate and collaborate to reach shared objectives. The emergence of conflicts among stakeholders may lead to undesired effects on software maintainability, yet it is often unavoidable in the long run. Community smells, i.e., sub-optimal communication and collaboration practices, have been defined to map recurrent conflicts among developers. While some community smell detection tools have been proposed in the recent past, these can be mainly used for research purposes because of their limited level of usability and user engagement. To facilitate a wider use of community smell-related information by practitioners, we present CADOCS, a client-server conversational agent that builds on top of a previous community smell detection tool proposed by Almarini et al. to (1) make it usable within a well-established communication channel like SLACK and (2) augment it by providing initial support to software analytics instruments useful to diagnose and refactor community smells. We describe the features of the tool and the preliminary evaluation conducted to assess and improve robustness and usability.

Tool repository: <https://github.com/gianwario/CADOCS>

Tool video: <https://www.youtube.com/watch?v=a2hOoE1M8hk>

Index Terms—Conversational Agents, Community Smells, Socio-Technical Analysis.

I. INTRODUCTION

Software engineering is a socio-technical activity that requires managers to deliver the best software product, matching the client requirements and finding the right compromise between the needs of the stakeholders involved in the process [1]–[3]. This is even more critical during software maintenance and evolution activities, which often require maintainers to prevent the progressive deterioration of product quality [4], [5]. For this reason, providing tools to handle collaboration and communication patterns in development communities is crucial for the success of the project.

Research has been proposing a number of socio-technical metrics, e.g., *socio-technical congruence* [6] and *turnover* [7], other than coined the term “*social debt*”: this refers to the unforeseen project costs connected to the presence of poor collaboration and communication conditions within a software community [8], [9]. *Community smells*, i.e., sub-optimal socio-technical characteristics and patterns, represent one of the major causes of social debt [10]. Defiant contributors who carry out their work with little consideration of their peers or the presence of siloed areas of the developer community that do not communicate are just some examples of community

smells that are not only associated to social debt [11], but also to the likely increase of technical debt [12].

In the recent past, researchers have been also proposing various tools for detecting community smells [11], [13]. The current state of the art is represented by CSDETECTOR, a command-line tool proposed by Almarini et al. [13] and able to (1) collect socio-technical metrics and (2) detect ten types of community smells. Despite the effort spent in engineering automated solutions to deal with community smells, most of the available tools were thought for research purposes, e.g., to enable large-scale empirical studies on socio-technical concerns [11], [14], without considering their usability nor the level of user engagement. As a consequence, the adoption of these tools by practitioners is still challenging.

To address this limitation, we introduce CADOCS (Conversational Agent for the Detection Of Community Smells). Our tool first wraps CSDETECTOR [13] to make it available as a readily-usable SLACK bot. Secondly, CADOCS builds on top of previous research [15], [16] to provide practitioners with suggestions on the most suitable refactoring actions to deal with the community smells detected. To increase user engagement, CADOCS relies on a machine learning model that enables natural language processing and understanding. Developing a conversational agent is driven by our willingness to provide project managers with an instrument that can detect community smells and guide them by suggesting refactoring actions within an environment they typically used to communicate with their team members; we used SLACK since it is considered the most popular and well-known team communication tool used by practitioners.¹ Thanks to our tool, we believe that project managers could have quick feedback on the status of the community and possibly react by rework on the communication structure implemented in SLACK, e.g., by monitoring how developers communicate over the channel. We make CADOCS publicly available in our online appendix [17] and open-source to enable its use for practitioners and extension for researchers.

II. BACKGROUND AND RELATED WORK

We developed a *conversational agent* for the detection of community smells. A *conversational agent* is a particular type of *bot*—an application that automates repetitive, dull, or

¹<https://tinyurl.com/yyyya8r5j>

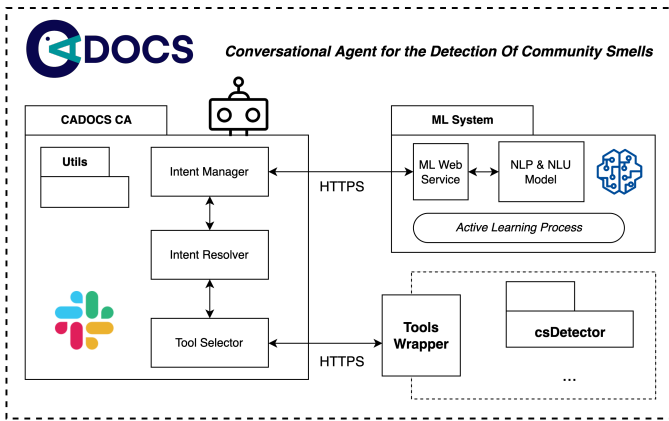


Fig. 1. The CADOCS's architecture.

predefined tasks [18], [19]—characterized by the fact that it communicates with users using a communication channel and natural language.

As an example of a bot for maintenance and evolution purposes, Khanan et al. built JITBOT [20], a Just-In-Time defect prediction GITHUB app which automatically generates feedback to developers about the commit's riskiness to introduce defects. From the communication side, Kim et al. developed GROUPFEEDBOT [21], i.e., a conversational agent meant to facilitate team discussion.

In the context of social aspects of software development, Tamburri et al. [8], [22] defined *community smells*, i.e., a set of socio-technical patterns that may lead to social debt. For example, the smell known as *Organizational Silo* effect represents a situation in which a software community is divided into siloed areas that do not communicate except through one or two of their respective members.

Various tools were proposed for the detection of community smells. For example, Tamburri et al. developed CODEFACE4SMELLS [23], a tool able to detect four types of smells through the use of repository mining and mailing lists analysis. More related to our work, Almarimi et al. developed CSDETECTOR [13], a tool able to automatically detect ten community smells using genetic algorithms that evolve decision rules. The tool was evaluated on 103 open-source projects resulting in an F-measure of 89%.

With respect to the state of the art, CADOCS is the only conversational agent for detecting community smells that explicitly aims at making this task *accessible* to practitioners through a usable interface implemented as a SLACK extension. Also, the tool exploits recent advances in refactoring community smells [15], [16] to provide insights into the most suitable actions that can be performed to deal with them.

III. CADOCS: CONVERSATIONAL AGENT FOR THE DETECTION OF COMMUNITY SMELLS

A. Conversational Agent Architecture

CADOCS was designed to adhere to a client-server architecture, i.e., the reference architecture to build conversational

agents [24], [25]. As shown in Figure 1, we divided the tool into three separable and modular modules described below.

Tools Wrapper. In its initial version, CADOCS was designed to wrap CSDETECTOR. This means that our tool currently makes available the functionalities provided by CSDETECTOR, both in terms of socio-technical metrics measurements and community smell detection [13]. The module takes the outcome of the command-line interface provided by Almarini et al. [13] and makes it available to the other modules in form of a `json` representation. When wrapping the tool, we addressed some limitations of CSDETECTOR. First, by implementing a web service layer architecture we enhanced the modes in which users can interact with the tool and abstracted him from the installation process of CSDETECTOR. Second, we improved the tool's output by reorganizing it in a more structured way, i.e., using CSV and JSON files. It is worth pointing out that our implementation was made on the basis of a *Strategy* design pattern [26] that enables the integration of additional community smell detectors and the implementation of strategies able to handle more sophisticated combinations of these detectors.

Machine Learning (ML) System. The machine learner behind the tool had the goal of easing the interaction between users and system's logic. Such an interaction was thought in terms of users' intents, i.e., the goals the user has in mind when questioning the bot [27], and were implemented by means of natural language processing (NLP) and natural language understanding (NLU). More specifically, we implemented a module that allows the interpretation of the users' intents by means of a machine learner that has been trained by collecting the possible ways to ask the bot to fulfil the requests of the tool. In particular, the machine learner was trained through a survey study that we performed to understand how stakeholders of the tool—i.e., researchers, practitioners, and students—would use to ask the bot about community smells and their refactoring. We distributed the survey to 9,207 stakeholders identified on SURVEYCIRCLE.² For the sake of space limitation, we make the survey available in our online appendix [17]. In addition, we developed an *active learning* mechanism to ensure the bot's increasing ability to recognize users' intentions and improve its usability over time. In particular, we used a learning algorithm that interactively queries a user under a certain threshold to label and increase the dataset with new instances [28]. Defining a threshold with active learning is challenging since it can be strictly connected to the context where it is applied [29], [30]. For this reason, CADOCS allows users to customize the threshold value. For instance, suppose the bot's confidence, i.e., the degree of certainty about the user's intent, stands below a set threshold value; in this case, the tool asks the user to clarify the intent, leading to a request for updating the training data. Practically, when the user confirms the correctness of the intent, CADOCS calls a custom API—implemented in the ML system—and

²The SURVEYCIRCLE website: <https://www.surveycircle.com>

TABLE I
COMMUNITY SMELLS AND MITIGATION STRATEGIES.


Community Smells	Mitigation Strategies
Organizational Silo. Siloed areas of the community that do not communicate, except through one or two of their respective members.	Restructure the community, Create communication plan, Mentoring, Cohesion exercising, Monitoring, and Introducing a social-rewarding mechanism.
Black Cloud. Information overload due to lack of structured communications or cooperation governance.	Create communication plan, Restructure the community, and Introduce a Social sanctioning mechanism.
Radio Silence. One interposes herself into every formal interaction across more sub-communities with little flexibility to introduce other channels.	Restructure the community, Create communication plan, Mentoring, Cohesion exercising, Monitoring, and Introduce a Social sanctioning mechanism.

Note: the complete list of detectable community smells is available in our online appendix [17].

saves the sentence and the predicted intent in the dataset used for training the NLP and NLU modules. In this way, CADOCS increases the size of the training set, and every ten new instances retrain the model. If the F-Measure of the retrained model is higher than the previous one, CADOCS replaces the older model.

CADOCS CA. This is the core part of the bot, containing the logic to interact with the user. It interacts with the *Slack* workspaces and has been implemented using the *Slack Events API*.³ To increase the tool’s usability and accessibility, we followed the heuristics provided by Langevin et al. [31] to design the interaction flow. The module also contains the logic to suggest refactoring strategies [15], [16].

The CADOCS Tool.

 **CADOCS** is a conversational agent developed for the *Slack* platform and able to use third-party tools to identify and manage *community smells* in software development communities on GITHUB.

B. Conversational Agent Features

By wrapping CSDETECTOR, our tool uses genetic algorithms [13] to detect the presence of community smells—some of which are described in Table I. In terms of refactoring suggestions, CADOCS provides support to handle three types of community smells, i.e., *Organization Silo*, *Black Cloud*, and *Radio Silence*. This is because of the lack of literature on how to mitigate the other community smells [15], [16]. In this sense, CADOCS should be considered as an early instrument for community smell refactoring: it indeed provides the architecture to support refactoring of community smells, hence enabling other researchers to build on top of the tool to make new refactoring actions available.

To interact with the conversational agent, we rely on the concept of *intent*. Specifically, CADOCS can:

- I1** Detect community smells and other related socio-technical information provided by CSDETECTOR, e.g., social graphs and metrics, in a development community given the GITHUB repository;
- I2** Detect community smells and other related socio-technical information in a development community from a specified date onward, given the GITHUB repository;
- I3** Show a report of the last execution;

³*Slack Events API*: <https://api.slack.com/apis/connections/events-api>

I4 Show background information about the community smells the bot can detect;

I5 Suggest refactoring strategies for the detected smells.

The first three intents (**I1**, **I2**, and **I3**) directly derive from the features of CSDETECTOR and have been enhanced only by a graphical point of view. The last two (**I4** and **I5**) are instead novel: the first is a simple way to increase tool usability, while the second aims to promote knowledge sharing, decorating the detected community smells with possible refactoring strategies identified by state of the art [15], [16].

IV. HOW TO USE THE TOOL

The installation guide of the tool is available in our online appendix [17]. Since the tool consists of 3 separate modules, i.e., the conversational agent, the machine learner, and the CSDETECTOR wrapper, the local installation of CADOCS may be time-consuming. For this reason, we also provided a web service to test the tool avoiding the full installation process. More details are reported in our online appendix [17].

As for its use, CADOCS is designed to work as a *Slack* app. A generic user has to (1) add CADOCS to one of their workspaces and (2) communicate with it in one of the workspace channels to reach one of the available intents.

Once the user has typed a sentence, CADOCS sends it to the natural language processing services to determine which of the five intents is being requested. Then, the tool sends an answer to the users in the same communication channel used for the input on the intent bases. If CADOCS is unsure which intent must be fulfilled, it asks the user for confirmation.

Figure 2 shows an example of an interaction between the tool and a user. The intent requested is to identify community smells in the repository QUANTUMKATAS (**I1**). As depicted, CADOCS identified two community smells and provided possible mitigation strategies—due to space limitation, the figure was edited to show refactoring suggestions just for one smell and to not show additional socio-technical metrics. In the end, CADOCS stored the result of the execution in a database (to handle **I3**) and stands by for any new interactions.

V. EVALUATION OF THE TOOL

Since CADOCS is built on top of CSDETECTOR, we did not evaluate the tool with respect to its detection accuracy but we deemed the performance reported by Almarini et al. [13] as valid in our case as well.

Our evaluation was instead focused on the main functionalities introduced by the conversational agent. First, we assessed

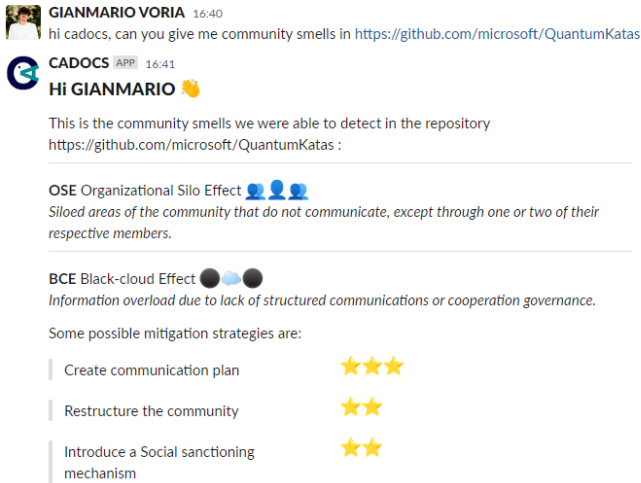


Fig. 2. Example GUI for tool interaction.

the robustness of the machine learning model responsible for the intent prediction. We performed *input testing* [32], i.e., an approach based on metamorphic testing [33] that aims at identifying potential reasons for unsuccessful training in the used data. Specifically, we performed two steps: (1) we morphed parts of our training data—questions to be asked to CADDOCS—into paraphrases having a relation of evenness with the original sentence; (2) we defined our test oracle so that the test will pass if the confidence of the prediction for the paraphrase does not exceed of 0.2 the confidence for the original sentence. Results showed that our dataset was good enough to train a model able to predict users’ intents consistently. Hence, we deemed our model as robust enough to handle the requests received by users.

Our second evaluation concerned the usability of the bot. In this case, we applied *iterative usability testing* [34]. This strategy is based on an iterative process in which, at each step, users give feedback about the tool’s user interface—after executing a series of tasks—and developers modify the tool accordingly. We recruited four graduated students who attended and achieved a Human Computer Interaction course during their degree. We asked them to conduct three tasks during each process iteration—the tasks are detailed in Table II. After each iteration, we directly interviewed participants to measure the bot’s usability in terms of well-known metrics—e.g., *learnability*, *efficiency*, and *satisfaction*—and consequently improved the tool interface [34]. We kept iterating the usability evaluation until reaching saturation. When assessing the usability of our tool, this process required five iterations. In those, we extracted four main improvements related to the addition of information in the user interface, like (1) the username of the user who requested the analysis, (2) the definition of the community smell detected, (3) the look-and-feel of the possible mitigation strategies, and (4) the link to the repository analyzed. Thanks to this feedback, we reached the final user interface shown in Figure 2.

Finally, concerning the active learning evaluation, users used

TABLE II
ITERATIVE USABILITY TEST.

#	Task Description
1	Add CADDOCS to your <i>Slack</i> workspace using the provided guidelines.
2	Ask CADDOCS to provide the social metrics of one of your repositories.
3	Report the identified metrics at task 1 and the mitigation strategies.

this component during their tasks without reporting issues. However, we are aware that this component needs to be studied over time to analyze its stability and evolution when collecting new instances. We are currently working on such an evaluation since it requires the tool’s analysis for a longer period of time.

VI. POTENTIAL IMPACT

CADDOCS has the potential to impact both researchers and practitioners. More specifically:

CADDOCS for Maintenance and Evolution. A wider adoption of our tool may empower the decision-making activities of project managers and other leading figures, who may easily collect socio-technical metrics about their teams and take informed decisions on how to improve maintenance and evolution processes. For instance, being aware of the presence of a *Truck Factor Smell (TFS)*—which arises when most of the project information and knowledge are concentrated in one or few developers—a team leader may take appropriate countermeasures to prevent project maintenance and evolution activities from increasing in cost and complexity in response to turnover of some developers.

CADDOCS for Knowledge Sharing. Our tool was designed to ease further extensions. In this sense, the tool may be used as an instrument to make other software maintenance and evolution research tools accessible to practitioners, hence increasing the knowledge sharing between research and practice. This is especially true when considering the ease of deployment: the architecture of CADDOCS was indeed designed to alleviate the effort required by practitioners to install and use research tools.

VII. CONCLUDING REMARKS

We developed CADDOCS, a conversational agent able to detect community smells, compute socio-technical metrics, and provide refactoring suggestions. We preliminarily assessed the tool’s capabilities, but as part of our future work, we plan to perform more empirical experiments. We plan to conduct those experimentations as part of our future research agenda. We also plan to keep evolving the tool’s features, supporting more refactoring recommendations and integrating more community smell detection tools.

ACKNOWLEDGMENT

Fabio is partially supported by the Swiss National Science Foundation - SNF Project No. PZ00P2_186090. This work has been partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY).

REFERENCES

- [1] P. M. Institute, *A Guide to the Project Management Body of Knowledge*, 7th ed., 8 2021.
- [2] F. P. Brooks Jr, *The mythical man-month: essays on software engineering*. Pearson Education, 1995.
- [3] P. Ralph, M. Chiasson, and H. Kelley, "Social theory for software engineering research," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '16. New York, NY, USA: Association for Computing Machinery, 2016.
- [4] A. Gupta and S. Sharma, "Software maintenance: Challenges and issues," *Issues*, vol. 1, no. 1, pp. 23–25, 2015.
- [5] B. Ulziit, Z. A. Warraich, C. Gencel, and K. Petersen, "A conceptual framework of challenges and solutions for managing global software maintenance," *Journal of Software: Evolution and Process*, vol. 27, no. 10, pp. 763–792, 2015.
- [6] G. Valetto, M. Helander, K. Ehrlich, S. Chulani, M. Wegman, and C. Williams, "Using software repositories to investigate socio-technical congruence in development projects," in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 25–25.
- [7] D. Homscheid and M. Schaarschmidt, "Between organization and community: investigating turnover intention factors of firm-sponsored open source software developers," in *Proceedings of the 8th ACM Conference on Web Science*, 2016, pp. 336–337.
- [8] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: Insights from industry," *Journal of Internet Services and Applications*, 2015.
- [9] —, "What is social debt in software engineering?" in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2013, pp. 93–96.
- [10] D. A. Tamburri, P. Lago, and H. v. Vliet, "Organizational social structures for software engineering," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, pp. 1–35, 2013.
- [11] D. A. Tamburri, F. Palomba, and R. Kazman, "Exploring community smells in open-source: An automated approach," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 630–652, 2019.
- [12] F. Palomba, D. Andrew Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik, "Beyond technical aspects: How do community smells influence the intensity of code smells?" *IEEE Transactions on Software Engineering*, vol. 47, no. 1, pp. 108–129, 2021.
- [13] N. Almarimi, A. Ouni, M. Chouchen, and M. W. Mkaouer, "csdetector: an open source tool for community smells detection," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 1560–1564.
- [14] N. Almarimi, A. Ouni, M. Chouchen, I. Saidani, and M. W. Mkaouer, "On the detection of community smells using genetic programming-based ensemble classifier chain," in *Proceedings of the 15th International Conference on Global Software Engineering*, 2020, pp. 43–54.
- [15] D. A. Tamburri, R. Kazman, and H. Fahimi, "The architect's role in community shepherding," *IEEE Software*, vol. 33, no. 6, pp. 70–79, 2016.
- [16] G. Catolino, F. Palomba, D. A. Tamburri, A. Serebrenik, and F. Ferrucci, "Refactoring community smells in the wild: the practitioner's field manual," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*, 2020, pp. 25–34.
- [17] G. Voria, V. Pentangelo, A. Della Porta, S. Lambiase, G. Catolino, F. Palomba, and F. Ferrucci, "CADOCS: a conversational agent for the detection of community smells — online appendix," 2022. [Online]. Available: <https://github.com/gianwario/CADOCS>
- [18] 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*, 2016, pp. 928–931.
- [19] S. Pérez-Soler, E. Guerra, and J. de Lara, "Flexible modelling using conversational agents," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 478–482.
- [20] C. Khanan, W. Luewichana, K. Pruktharathikoon, J. Jiarpakdee, C. Tantithamthavorn, M. Choetkiertikul, C. Ragkhitwetsagul, and T. Sunetnanta, "Jitbot: An explainable just-in-time defect prediction bot," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 1336–1339.
- [21] S. Kim, J. Eun, C. Oh, B. Suh, and J. Lee, "Bot in the bunch: Facilitating group chat discussion by improving efficiency and participation with a chatbot," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
- [22] D. A. Tamburri, "Software architecture social debt: Managing the incommunicability factor," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 1, pp. 20–37, 2019.
- [23] F. Palomba and D. A. Tamburri, "Predicting the emergence of community smells using socio-technical metrics: a machine-learning approach," *Journal of Systems and Software*, vol. 171, p. 110847, 2021.
- [24] S. Srivastava and T. Prabhakar, "A reference architecture for applications with conversational components," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2019, pp. 1–5.
- [25] C. Brown and C. Parnin, "Sorry to bother you again: Developer recommendation choice architectures for designing effective bots," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 56–60.
- [26] E. Gamma, R. Helm, R. Johnson, R. E. Johnson, J. Vlissides *et al.*, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.
- [27] L. Chen, D. Zhang, and L. Mark, "Understanding user intent in community question answering," in *Proceedings of the 21st international conference on world wide web*, 2012, pp. 823–828.
- [28] B. Settles, "Active learning literature survey," 2009.
- [29] B. Bryan, R. C. Nichol, C. R. Genovese, J. Schneider, C. J. Miller, and L. Wasserman, "Active learning for identifying function threshold boundaries," *Advances in neural information processing systems*, vol. 18, 2005.
- [30] B. Settles, "Active learning literature survey," 2009.
- [31] R. Langevin, R. J. Lordon, T. Avrahami, B. R. Cowan, T. Hirsch, and G. Hsieh, "Heuristic evaluation of conversational agents," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–15.
- [32] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5193–5254, 2020.
- [33] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–27, 2018.
- [34] A. Genov, "Iterative usability testing as continuous feedback: A control systems perspective," *Journal of Usability Studies*, vol. 1, no. 1, pp. 18–27, 2005.