MutaBot: A Mutation Testing Approach for Chatbots

Michael Ferdinando Urrico
m.urrico@campus.unimib.it
University of Milano-Bicocca
Milan, Italy

Diego Clerissi
diego.clerissi@unimib.it
University of Milano-Bicocca
Milan, Italy

Leonardo Mariani
leonardo.mariani@unimib.it
University of Milano-Bicocca
Milan, Italy

ABSTRACT

Mutation testing is a technique aimed at assessing the effectiveness of test suites by seeding artificial faults into programs. Although available for many platforms and languages, no mutation testing tool is currently available for conversational chatbots, which represent an increasingly popular solution to design systems that can interact with users through a natural language interface. Note that since conversations must be explicitly engineered by the developers of conversational chatbots, these systems are exposed to specific types of faults not supported by existing mutation testing tools.

In this paper, we present MutaBot, a mutation testing tool for conversational chatbots. MutaBot addresses mutations at multiple levels, including conversational flows, intents, and contexts. We designed the tool to potentially target multiple platforms, while we implemented initial support for Google Dialogflow chatbots. We assessed the tool with three Dialogflow chatbots and test cases generated with Botium, revealing weaknesses in the test suites.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Human-centered computing** → Natural language interfaces.

KEYWORDS

Chatbot Testing, Mutation Testing, Botium, Dialogflow

ACM Reference Format:
Michael Ferdinando Urrico, Diego Clerissi, and Leonardo Mariani. 2024. MutaBot: A Mutation Testing Approach for Chatbots. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3639478.3640032

1 INTRODUCTION

The widespread adoption of conversational chatbots (e.g., 88% of people had at least a conversation with a chatbot in 2022 [13]), also known as virtual assistants or conversational agents, in various domains (e.g., e-commerce, booking, tech support, healthcare, and more) [1, 14, 28] raised the need of dedicated testing methodologies and tools [4–8, 15, 24, 27] to assess their effectiveness.

Mutation testing is a popular validation strategy that can be used to assess the effectiveness of test suites and test case generation techniques [17]. It is a fault-based testing technique that consists of producing many faulty versions of a software under test, called *mutants*, by systematically introducing artificial faults in the software. The faults are introduced through *mutation operators*, designed to change the software according to patterns that represent possible faults. For instance, an operator may replace the name of a variable in an expression with another one to represent the case of a wrongly referred variable. The effectiveness of test suites is assessed by measuring the rate of mutants that they can reveal, that is, the percentage of mutants that fail (i.e., *killed*, according to the mutation testing terminology [17]) when executed with the test suite.

In order to apply mutation testing in a given context, it is necessary to define a set of operators that can modify the relevant language elements and entities. Conversational chatbots, differently from other software, implement a number of components with a specific structure, even spanning multiple diverse languages, to encode the supported conversations. For instance, chatbots implemented with Google Dialogflow [12] are composed of several JSON files that specify both the sentences that a chatbot can use and the possible flows of conversation that the chatbot can follow, text files with alternative sentences, and webhooks to trigger external functions. This case is the same for all the chatbot platforms (e.g., Rasa [26] and Amazon Lex [18]), where conversations and interactions are implemented with specific language elements.

Currently, there is *no mutation testing tool that can properly target the elements that implement the conversations*, taking the semantics of the mutated element into consideration. On the other hand, conversations need to be carefully validated since they are the main and only interface used by chatbots to interact with the outside world.

In this paper, we introduce MutaBot, to be best of our knowledge the first mutation testing tool for chatbots. MutaBot is characterized by a general-purpose technology-agnostic design, inspired by the meta-model proposed by Cañizares *et al.* [9], that can be exploited to virtually address any chatbot platforms. When a specific platform has to be addressed, the technology-dependent interfaces, which provide the operations to locate the items to be mutated in a given platform, have to be implemented. Our first prototype version of the tool implements 24 mutation operators specifically designed for chatbots and provides support to Google Dialogflow [12], which is among the most popular chatbot platforms.

We preliminary validated our tool with three Dialogflow chatbots, applying all the supported mutations, and observed how the test suites generated with Botium [5], a state-of-the-art automated test generation framework for chatbots, perform in detecting mutants. Results show that the test suites were capable of killing only

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3640032

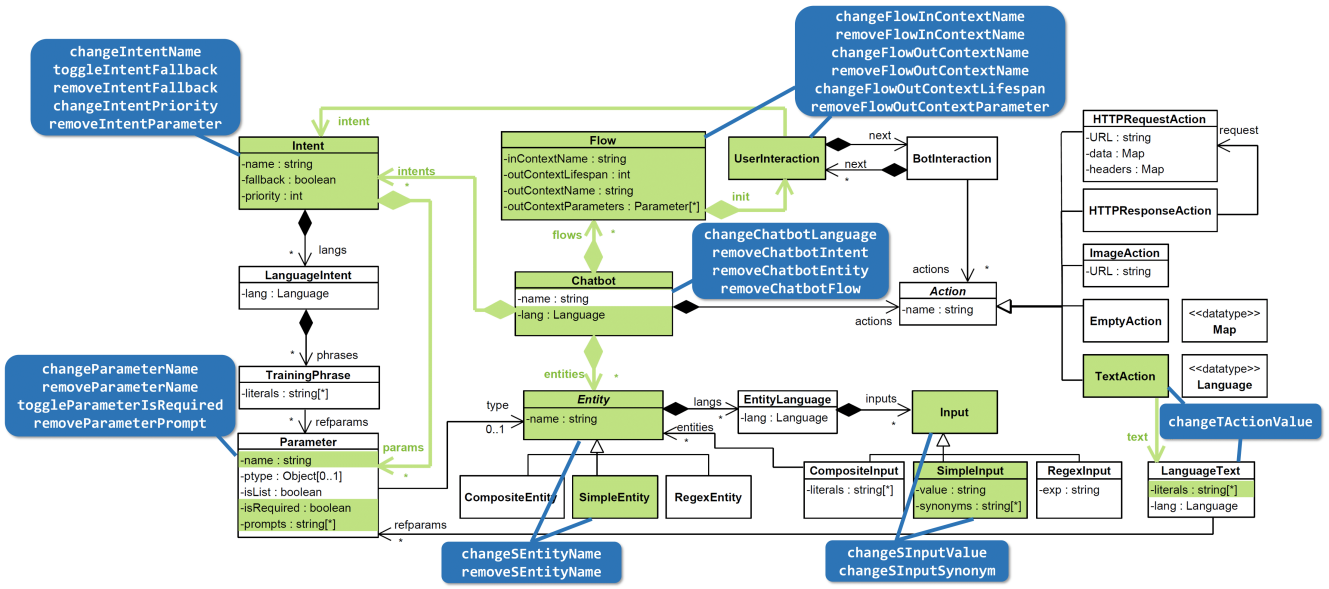

Figure 1: Chatbot structure meta-model adapted from Cañizares *et al.* [9].

between 28% and 37% of the mutants, indicating how test case generation tools for chatbots need to be improved to better cover the space of the possible conversations.

The paper is organized as follows. Section 2 describes MutaBot, its architecture and the supported mutation operators. Section 3 presents the procedure we followed to conduct our empirical evaluation and discusses the results. Section 4 discusses the related work. Finally, Section 5 provides final remarks.

2 MUTABOT APPROACH

The mutation operators implemented in MutaBot are inspired by the chatbot meta-model originally proposed by Cañizares *et al.* [9], which specifies general platform-agnostic chatbot concepts and features, derived by analyzing 15 chatbot development platforms [21].

Figure 1 shows the meta-model. A *Chatbot* is represented as a class composed of *Intents* (i.e., the possible goals of a user), *Entities* (i.e., the data types that can be used in a conversation), *Actions* (i.e., the possible responses of a chatbot), and conversational *Flows* (i.e., the possible user-bot interaction flows). As chatbots are intended to be multi-language, *Intents* declare the *TrainingPhrases* used to train the chatbot for each supported language. *TrainingPhrases* can use *Parameters*, typed according to the supported entities. *Entities* can be *Simple* (i.e., a literal with synonyms), *Complex* (i.e., a composition of other entities and literals), or *Regex* (i.e., values derived from regular expressions). In response to a user request, a chatbot performs an *Action*, which can be of type *Text* (i.e., a plain text response), *Image* (i.e., a graphical object to interact with, shown to the user), a *HTTPRequest* and *HTTPResponse* (i.e., an action invoking an external service to respond to the user), or *Empty* (i.e., no action is performed). Finally, a conversational *Flow* determines the sequence of interactions between the user and the chatbot, alternatively activating *Intents* and *Actions*. We added the notions of *name*, *lifespan*, and *parameters* of the context data (i.e., the memory of the chatbot) associated with each conversational flow to the meta-model, since these are also important concepts that determine how conversations work in multiple platforms.

We annotated Figure 1 to indicate the 24 operators currently supported by MutaBot. We list the name of the operator in a call-out text, and we highlight the area of the meta-model affected by the operator. We defined operators targeting all the main entities and features that play a relevant role in a conversation. Unlike traditional data mutation techniques, MutaBot implements operators carefully designed to manage the complexity of the target platform, to identify the software element exactly corresponding to the conceptual element presents in the meta-model.

The operators defined in MutaBot drastically differ from traditional mutation operators affecting statements, such as array index manipulations or decision mutation operators. Instead, the implemented operators address the peculiarities of conversational agents. In fact, each operator defined on the meta-model may implement changes of different nature depending on the specific chatbot technology. For instance, changing the flow of a conversation requires changing the values stored in the *JSON* contexts objects for DialogFlow, while they can be directly altered by changing *YAML* configuration files in Rasa.

The implemented operators represent faults that may originate from the imprecise design of the conversations, such as, mistakenly swapping the name of one parameter with another one in a response, forgetting to propagate the context from one intent to another, or not setting a mandatory parameter as required. We briefly present below the list of supported operators, and the tool. For a more detailed view of the operators and the tool, please refer to the MutaBot project repository[1].

[1]https://gitlab.com/Michael-Urrico/defectinjector-java

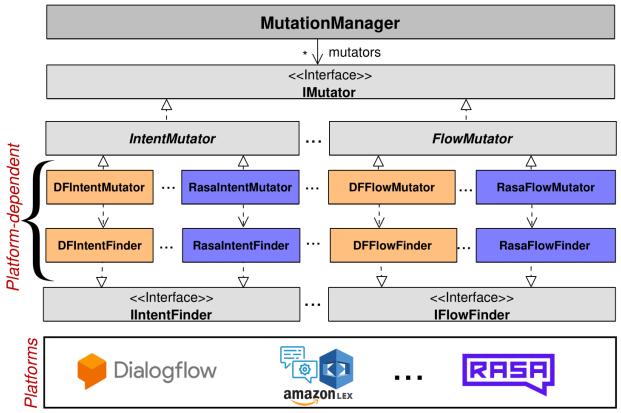

Figure 2: MutaBot architecture.

2.1 Mutant Operators

We list below the mutant operators grouped by the entity that is affected by the operators, according to Figure 1.

Chatbot: (1) *changeChatbotLanguage*: replace a chatbot supported language with another existing[2] language, (2) *removeChatbotIntent*: remove a chatbot intent, (3) *removeChatbotEntity*: remove a chatbot entity, (4) *removeChatbotFlow*: remove a chatbot flow.

Flow: (1) *changeFlowInContextName*: replace a flow input context name with another existing name, (2) *removeFlowInContextName*: remove a flow input context name, (3) *changeFlowOutContextName*: replace a flow output context name with another existing name, (4) *removeFlowOutContextName*: remove a flow output context name, (5) *changeFlowOutContextLifespan*: replace a flow output context lifespan with a random value between 1 and 3, different from the original one, (6) *removeFlowOutContextParameter*: remove a flow output context parameter.

Intents: (1) *changeIntentName*: replace an intent name with another existing name, (2) *toggleIntentFallback*: switch an intent "fallback" flag to true/false, (3) *removeIntentFallback*: remove an intent "fallback" flag, (4) *changeIntentPriority*: replace an intent priority with a random value between 0 and 1,000,000 (max priority value) different from the original one, (5) *removeIntentParameter*: remove an intent parameter.

Parameters: (1) *changeParameterName*: replace a parameter name with another existing name, (2) *removeParameterName*: remove a parameter name, (3) *toggleParameterIsRequired*: switch a parameter "isRequired" flag to true/false, (4) *removeParameterPrompt*: remove a parameter prompt.

Inputs: (1) *changeSEntityName*: replace an entity name with another existing name, (2) *removeSEntityName*: remove an entity name, (3) *changeSInputValue*: replace an input value with a random string, (4) *changeSInputSynonym*: replace an input synonym with a random string, (5) *changeTActionValue*: replace a textual action value with a random string.

[2]To generate more realistic mutations, we defined the semantics of replace operators like *replace * with another existing **(e.g., *changeChatbotLanguage*) as replacing the value of the target property with another existing value of the same property, rather than using a randomly generated value (e.g., replacing the name of an intent with another existing intent name). We plan to add the generation of random values in the future.

Table 1: Subject Chatbots.

Name	Domain	# Int./Ent./Act.	# Tests
AppointmentScheduler [21][a]	Booking	3/3/19	21(0)
Device [21][b]	Home Automation	11/2/11	95(28)
Nutrition [8, 20][c]	Food & Health	4/7/18	46(1)

[a] https://github.com/priyankavergadia/AppointmentScheduler-GoogleCalendar
[b] https://dialogflow.cloud.google.com/ (prebuilt)
[c] https://github.com/viber/apiai-nutrition-sample

2.2 MutaBot Tool

The architecture of the tool is shown in Figure 2. Its core component is a `MutationManager` that is responsible of running the individual mutators available in the tool. Each mutator is implemented as an abstract operator, responsible of actuating the change, and an abstract finder, responsible of finding the code element that must be changed by the operator. These abstract components are refined for the target platform. Our prototype supports Dialogflow chatbots. In order to find the elements to mutate, MutaBot knows how to search within the files that compose a chatbot and process them.

The tool takes in input multiple XML configuration files that specify the target folder that hosts the code of the chatbot, the type of desired output (a full copy of the mutated chatbot, or only a copy of the modified files), and the set of operators that must be applied. The output consists of a set of mutated chatbots, either including the full executable code of the bot or only the modified files, and a report file with the list of seeded mutations.

3 EMPIRICAL EVALUATION

We preliminarily evaluated MutaBot by generating mutants for Dialogflow chatbots, and assessing the effectiveness of the test cases generated with the Botium test case generation tool [5], which is a white-box testing technique developed by Botium GmbH in 2018, widely used in both industrial and academic contexts [8, 19, 22]. Note that here a test is a conversation, with specific expectations on the responses of the chatbot. The conversations automatically generated by Botium cover the user-bot scenarios that could be derived from the implementation of the chatbot under test. In the case of Dialogflow, conversations are derived from the *JSON* files that encode the possible requests and responses.

As subjects, we selected three third-party Dialogflow chatbots of different domains and features, that have been used in previous studies [8, 20, 21]. Table 1 summarizes their characteristics (i.e., name, domain, and number of intents/entities/actions), and the sizes of the test suites generated by Botium.

For each chatbot, we first ran Botium in *generation* mode in order to generate our baseline regression test suite. We then used Botium in *execution* mode, repeating five times each test suite, in order to detect and discard any spuriously failing test or *flaky test case*[3], that Botium may have generated and that could invalidate our results. We report, between parentheses in Table 1, the number of discarded tests.

We applied MutaBot to each chatbot, generating mutants with all the operators listed in Section 2.1, applying each mutation to every entity instance present in the target chatbot. For instance, the

[3] *"A flaky test is a test that both passes and fails periodically without any code changes."* [31]

Table 2: Mutants killed over total by Botium test suites.

Chatbots	# Killed/# Equivalent/# Generated (% Killed)					
	Chatbot	Flows	Intents	Parameters	Inputs	Total
AppointmentScheduler	4/0/5 (80%)	-	4/5/12 (57%)	0/0/3 (0%)	0/0/7 (0%)	8/5/27 (36%)
Device	12/0/19 (63%)	3/2/12 (30%)	11/17/44 (41%)	0/0/8 (0%)	0/0/6 (0%)	26/19/89 (37%)
Nutrition	5/2/12 (50%)	-	3/7/16 (33%)	0/0/6 (0%)	1/16/23 (14%)	9/25/57 (28%)

removeChatbotIntent operator was applied to every intent present in a tested chatbot, generating mutants each one with a different intent removed. Some mutations were not applicable to all chatbots because of missing properties to mutate (e.g., removing a context parameter from a chatbot with no context). We finally independently deployed each mutant on Dialogflow and tested it with the Botium test suite. Table 2 reports the numbers and percentages of *non-equivalent*[4] mutants killed by the test suites, grouped per type of mutated element.

Results show that the test suites were capable of killing only between 28% and 37% mutants in total.

Mutations affecting the chatbot structure, like intent or entity removal, as well as those affecting intents properties, are easier to reveal (50%-80% for **Chatbot** category, 33%-57% for **Intents**), with the exception of those targeting the fallback mechanism, which is activated when the bot cannot understand the message of the user. The application of *toggleIntentFallback* operator to an intent may introduce tricky behaviors in chatbots, which are sometime detected, but are not always covered by the generated test cases, as they are mainly covering the positive scenarios. On the other hand, changing the priority of intents mainly produces equivalent mutants, as intents are rarely in competition in terms of which one is supposed to be activated first when a certain user request is formulated.

Mutations targeting the **Flows** category are difficult to be revealed (30% in the Device chatbot), since they require the construction of lengthy and articulated conversations that Botium often fails to create. This group of mutations requires further investigation on more complex chatbots, as the conversational flows cannot be mutated if intents are self-contained. Finally, the mutations targeting parameters and inputs remain mostly undetected (0% for the **Parameters** category, and 14% at best for **Inputs**).

These initial results obtained with MutaBot show how there is still a significant gap to be addressed in terms of the capability of test generation tools to exercise chatbot conversations thoroughly.

4 RELATED WORK

Conversational chatbots are increasingly exploited in various contexts (e.g., Web, Mobile) to support users 24/7 in a plethora of activities (e.g., booking, home banking, etc.) [1, 2]. Despite their increasing level of adoption, the dedicated quality assurance solutions are still limited [8, 19]. Indeed, they are often restricted to individual modules (e.g., the automated speech recognition systems [3, 16, 25]), or they heavily rely on human knowledge and fixtures [6, 7, 30], or are limited to performance evaluation metrics [10, 11, 24].

Support to test automation mainly consists of frameworks for the automation of test execution. They include solutions to test Alexa skills [29], Rasa chatbots [26], and to generically address end-to-end conversational agents [4, 23]. The commercial tool Botium [5] represents an initial effort in the design of automated functional testing tools for chatbots. Input mutation is further addressed in Charm [8], and by Guichard *et al.* [15], by building paraphrases of user queries generated by another tool [27]. Bozic *et al.* face the oracle problem of chatbot testing via metamorphic relations, involving input transformations, such as synonyms replacements and words removal [6].

The lack of mutation testing solutions for conversational chatbots limits the number of faults against which techniques can be assessed, generating difficulties with large scale experimentation. To address this gap, this paper proposes MutaBot, a mutation testing tool specifically designed to target the conversations implemented by chatbots.

5 CONCLUSION

Mutation testing is an important solution to extensively validate test suites and testing techniques, as it ensures that the software under test is thoroughly exercised. So far, no mutation testing tool that can target the conversations implemented by chatbots have been defined, hindering the opportunity to systematically validate test suites against a large number of faults.

In this paper, we presented MutaBot, a mutation testing tool for chatbots. Our tool targets Google Dialogflow chatbots and implements operators to alter the main elements that may impact a conversation, including intents, conversational flows, entities, and contexts. We reported an initial validation of our tool that shows how the state-of-the-art Botium test generation technique may fail at revealing several mutations, calling for solutions that can more thoroughly validate conversations.

As future work, we aim at exploiting our platform-agnostic architecture to support additional chatbot platforms, such as Amazon Lex and Rasa, and to cover additional elements of the meta-model. Further, we plan to add more variants of the existing operators (e.g., adding operators that mutate properties with random values), and to finally exploit MutaBot to perform large scale fault-based assessment of the existing test case generation tools for chatbots.

ACKNOWLEDGMENTS

This work has been partially funded by the Engineered MachinE Learning-intensive IoT systems (EMELIOT) national research project, which has been funded by the MUR under the PRIN 2020 program (Contract nr. 2020W3A5FY).

[4] An equivalent mutant is a mutant that cannot be killed by any test case [17]. We manually inspected the generated mutants to establish their equivalence to the original program, comparing the differences between the original and mutated properties and detecting those that had no impact on the behavior of the chatbot (e.g., changing the priority of an intent when there are no other intents to compete with).

REFERENCES

[1] Eleni Adamopoulou and Lefteris Moussiades. 2020. Chatbots: History, technology, and applications. *Machine Learning with Applications* 2 (2020), 100006.

[2] Eleni Adamopoulou and Lefteris Moussiades. 2020. An overview of chatbot technology. In *IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*. Springer, 373–383.

[3] Muhammad Hilmi Asyrofi, Ferdian Thung, David Lo, and Lingxiao Jiang. 2020. CrossASR: Efficient differential testing of automatic speech recognition via text-to-speech. In *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 640–650.

[4] Bespoken. 2023. *Bespoken Documentation.* https://read.bespoken.io/ Visited: October 2023.

[5] Botium. 2023. *Botium Documentation.* https://botium-docs.readthedocs.io/en/latest/ Visited: October 2023.

[6] Josip Božić. 2022. Ontology-based metamorphic testing for chatbots. *Software Quality Journal (SQJ)* 30, 1 (2022), 227–251.

[7] Josip Bozic, Oliver A Tazl, and Franz Wotawa. 2019. Chatbot testing using AI planning. In *Proceedings of the 2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 37–44.

[8] Sergio Bravo-Santos, Esther Guerra, and Juan de Lara. 2020. Testing chatbots with Charm. In *Proceedings of the 13th International Conference on the Quality of Information and Communications Technology (QUATIC)*. Springer, 426–438.

[9] Pablo C Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Automating the measurement of heterogeneous chatbot designs. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC)*. ACM, 1491–1498.

[10] Chatbottest. 2023. *Chatbottest.* https://chatbottest.com/ Visited: October 2023.

[11] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. 2021. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review* 54 (2021), 755–810.

[12] Dialogflow. 2023. *Dialogflow Documentation.* https://cloud.google.com/dialogflow/docs Visited: October 2023.

[13] Maryia Fokina. 2023. *The Future of Chatbots: 80+ Chatbot Statistics for 2023.* https://www.tidio.com/blog/chatbot-statistics/ Visited: October 2023.

[14] Jonathan Grudin and Richard Jacques. 2019. Chatbots, humbots, and the quest for artificial general intelligence. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 1–11.

[15] Jonathan Guichard, Elayne Ruane, Ross Smith, Dan Bean, and Anthony Ventresque. 2019. Assessing the robustness of conversational agents using paraphrases. In *Proceedings of the 2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 55–62.

[16] Futoshi Iwama and Takashi Fukuda. 2019. Automated testing of basic recognition capability for speech recognition systems. In *Proceedings of the 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 13–24.

[17] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering (TSE)* 37, 5 (2010), 649–678.

[18] Amazon Lex. 2023. *Amazon Lex Documentation.* https://docs.aws.amazon.com/lex/ Visited: October 2023.

[19] Xiaomin Li, Chuanqi Tao, Jerry Gao, and Hongjing Guo. 2022. A Review of Quality Assurance Research of Dialogue Systems. In *Proceedings of the 2022 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 87–94.

[20] Sara Pérez-Soler, Esther Guerra, and Juan De Lara. 2020. Model-driven chatbot development. In *Proceedings of the 39th International Conference on Conceptual Modeling (CM)*. Springer, 207–222.

[21] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2021. Creating and migrating chatbots with CONGA. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-C)*. IEEE, 37–40.

[22] Sara Pérez-Soler, Sandra Juarez-Puerta, Esther Guerra, and Juan de Lara. 2021. Choosing a chatbot development tool. *IEEE Software* 38, 4 (2021), 94–103.

[23] Playwright. 2023. *Playwright.* https://playwright.dev/ Visited: October 2023.

[24] QBox. 2023. *QBox.* https://qbox.ai/ Visited: October 2023.

[25] Yao Qin, Nicholas Carlini, Garrison Cottrell, Ian Goodfellow, and Colin Raffel. 2019. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 5231–5240.

[26] Rasa. 2023. *Rasa Documentation.* https://rasa.com/docs/ Visited: October 2023.

[27] Elayne Ruane, Théo Faure, Ross Smith, Dan Bean, Julie Carson-Berndsen, and Anthony Ventresque. 2018. Botest: A framework to test the quality of conversational agents using divergent input examples. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces Companion (IUI-C)*. ACM, 1–2.

[28] Bayan Abu Shawar and Eric Atwell. 2007. Chatbots: Are they really useful? *Journal for Language Technology and Computational Linguistics* 22, 1 (2007), 29–49.

[29] Alexa Simulator. 2023. *Alexa Simulator Documentation.* https://developer.amazon.com/en-US/docs/alexa/devconsole/alexa-simulator.html Visited: October 2023.

[30] Marisa Vasconcelos, Heloisa Candello, Claudio Pinhanez, and Thiago dos Santos. 2017. Bottester: Testing conversational systems with simulated users. In *Proceedings of the Brazilian Symposium on Human Factors in Computing Systems*. ACM, 1–4.

[31] Wei Zheng, Guoliang Liu, Manqing Zhang, Xiang Chen, and Wenqiao Zhao. 2021. Research progress of flaky tests. In *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 639–646.