Testing challenges for NLP-intensive bots

Jordi Cabot*†, Loli Burgueño†, Robert Clarisó†, Gwendal Daniel†, Jorge Perianez-Pascual‡, and Roberto Rodriguez-Echeverria‡
*ICREA, Barcelona, Spain
jordi.cabot@icrea.cat
†Universitat Oberta de Catalunya, Barcelona, Spain
{lburguenoc,rclariso,gdaniel}@uoc.edu
‡University of Extremadura, Cáceres, Spain
{jpery,rre}@unex.es

***Abstract*—The popularity of bots is on the rise, with many bots able to interact with users via a chat or voice interface thanks to the embedding of a Natural Language Processing (NLP) component. Still, companies often express concerns about the quality of such bots, as their malfunctioning could have a severe impact on the company revenue or image. Unfortunately, the field of testing NLP-intensive bots is still in its infancy. This paper aims to characterize the testing properties and techniques (and their adaptation) relevant to this type of bots. We believe this will be helpful as a reference framework to compare and evaluate future bot testing research initiatives.**

***Index Terms*—Bots, Testing, NLP, SWEBOK**

I. Introduction

The success of Artificial Intelligence (AI) has sparked a substantial interest in the software engineering (SE) field to improve its scalability and quality [1]. AI applications face common challenges in their SE processes [2]. Among those, they are hard to specify [3], test, verify and debug [4].

This is even more problematic for NLP-intensive bots. We define NLP-intensive bots (or NLP bots for short) as software bots where the processing of natural language text is crucial to the bot functionality. This category includes chatbots and voicebots[1] but also a large number of bots that need to react to events whose payload involves user utterances (e.g., a bot that reports toxic comments in a GitHub repository).

Testing NLP-intensive bots is especially difficult as testing needs to cover the NLP side of the bot, the software side—often implemented as a finite-state machine (FSM)—and the interaction between both. Thus, even if testing the core software aspects of a bot is a well established research topic [5], the other two are still open research challenges for which we do not even have a clear definition of the testing properties and techniques needed to address them.

In this paper, we review and adapt current testing concepts to NLP-intensive bots in order to advance towards a unified reference framework to promote, classify, compare and evaluate future bot testing research initiatives. Our characterization will rely on the SWEBOK [6] body of knowledge as the source of the testing challenges terminology and organization.

[1]A voicebot employs a speech-to-text component to translate the user input to text, which is what it is really processed by the bot.

II. Quick introduction to the elements of a NLP-intensive bot

NLP-intensive bots typically rely on *intent-based* NLP engines to understand user inputs. *Intents* are a class of events representing the user intention in an abstract way. They are defined through a data set composed of example sentences, which are used to train the underlying ML/NLP engine. For instance, a *Greetings* intent represents the different ways the user can greet the bot such as “Hi”, “Hello”, “Good morning”, etc. These alternative formulations for the intent will comprise the training set to teach the bot what types of user utterances should be matched with this intent.

An intent can carry *parameters*, which are contextual information automatically extracted from the user input. A parameter refers to an *entity*, which can be a preset type managed by the NLP engine (e.g., a city name, a date, etc.), or user-defined enumerations (e.g., a list of products available in an e-commerce website).

As Pereira and Díaz reported [7], bots cannot be reduced to raw NLP capabilities, and other dimensions such as complex system engineering, service integration, and testing have to be taken into account when developing such applications. Indeed, the intents produced by the NLP component are typically integrated in complex FSMs representing the bot’s execution logic. This FSM describes the bot *reaction* to a given intent, but also the sequences of intents/reactions that constitute a conversation path (or flow).

Finally, it is important to stress that bots are not standalone applications, and they have to be deeply integrated with the platform they target (e.g., messaging or voice platforms). This implies taking into account not only the limitations of the platform, but also its specific capabilities (e.g., advanced UI components such as buttons, datepickers, etc.).

An example of a NLP-intensive bot could be our bot to chat with GitHub[2] that can be used to both open issues in GitHub from Slack or, conversely, to subscribe and receive GitHub notifications and display them in Slack.

[2]https://github.com/xatkit-bot-platform/xatkit-examples/tree/master/GitHubBots/GithubBot

III. State of the art

Despite the importance of NLP-intensive bots, few tools and research initiatives tackle the problem of bot testing. The importance of having this discussion is also emphasized in [7].

The "big players" like Microsoft Bot Framework or Amazon Lex provide facilities that can be used to define simple tests for individual intents but there is no support for more complex tests, their automatic generation or any kind of more platform-agnostic testing language. This gap is (slightly) covered by some other commercial and open source tools. Botium [8] is an open source chatbot testing framework (and commercial service) especially targeting what they call "Conversational Flow Testing" that enables you to record (and store) a chatbot session as a test to reexecute in the future, mostly for regression testing purposes. Zypnos [9] offers a similar functionality. Chatbottest [10] is an open source guide (rather than a framework) for testing chatbots, which is complemented with a Chrome extension (called Alma) that provides guidelines for chatbot testing in Telegram and Facebook Messenger. QBox [11] is a commercial product focusing on testing training datasets to identify whether a chatbot identifies the correct intent for a given input. DashBot [12] is more of a dashboard/analytics service for bots that helps visualize, for instance, where the bot is failing to respond.

With a more focused research perspective, Botest [13] is a framework that generates divergent input examples from inputs that the chatbot accurately recognizes to assess the robustness of the bot. Paraphrasing is another promising approach to test diverse inputs [14]. Ribeiro et al. [15] propose a matrix of general linguistic capabilities and test types that facilitate the ideation of comprehensive tests. Lara et al. [16] are able to generate additional tests for chatbot using mutation functions. Bozic et al. [17] proposes an AI planning approach for testing bots where the chatbot must properly answer the questions posed to it as part of a generated plan. The same authors discuss how metamorphic relations can help in defining the test oracles for chatbot tests [18]. Finally, an evaluation from a usability perspective is described in [19].

Note that most research evaluations study the NLP component in isolation, ignoring the required interactions between this component and the rest of the bot and the external services it communicates with.

As we will see in the following section, these tools and papers are a first step towards a complete NLP-intensive testing solution but so far they cover a small fraction of the full spectrum of the bot testing needs.

IV. Testing NLP-intensive bots

According to the SWEBOK, software testing consists of the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain. This definition is expanded and broken down into the following list of testing topics: testing fundamentals, test levels, test techniques, test-related measures, test process and testing tools.

We adhere to this same classification to discuss how these topics must be adapted to deal with the particularities of testing NLP bots. Skipping a (sub)category does not mean that it is not important for their testing, it just means that the "standard" practice in this subfield can be seamlessly applied to bot testing (e.g. the fundamentals section is mostly about introducing basic terminology).

A. Testing Levels

This subsection covers two different aspects: the target and the objective of the testing. The target aims to cover the different levels in which the testing of bots can be divided. The objectives refer to the specific testing property we aim to evaluate.

1) Test target: **Unit testing** of a NLP bot is defined as making sure an individual intent has been trained with a proper dataset to match the user utterances is supposed to match (and do not match or match only with low confidence the other ones). This is, a basic unit test passes if all testing sentences produce the expected match for the tested intent. A more basic unit test should include checking that the utterance entities have been recognized and matched to the intent parameters. At this level, we cannot also evaluate the bot response as this typically depends on a number of other components and the current bot state.

Integration testing for NLP bots must include testing that: 1) There are no overlaps between intents that could deteriorate the individual intent matching due to intent definitions being too similar, and 2) a correct set of user inputs moves the bot to the expected state in the FSM as defined in the bot's conversation flow.

System testing tests the design and behavior of the whole bot at the same time (and not only of subparts of it). For NLP bots, this implies testing together the NLP part with the rest of the bot components (including calls to external services) and making sure they all work together as expected, both from a functional and non-functional perspective. Note that some of these tests must target the training phase of the bot as the NLP part may be delegated to cloud-based NLP components and the availability of such components and the performance of the bot deployment and training on them are important. System tests should also test the communication between the bot itself and other parts of the software system (e.g., some bots may need to react differently depending on the ongoing interaction between the user and the GUI, for instance, a bot to help users fill a complex form).

2) Test objective: **Acceptance testing**. Checking that the NLP bot behaves as expected by the client is especially relevant as the NLP definition is an *underspecification*: the set of training sentences are examples from which the bot must learn. Therefore, we need the client to validate that the learning process actually produces a bot "smart" enough to understand the idea that its users want to communicate.

Installation testing for bots is similar to other types of software. Nevertheless, as it is easily dismissed, we would like to emphasize it. Note that bots typically are deployed over a number of communication channels (e.g., web pages, social media channels, messaging platforms, . . .). Testing that the bot is properly configured (including keys and authentication aspects but also display parameters) and can actually communicate with users is not trivial.

Regression testing must be in place and executed regularly even if the bot definition has not changed. If the bot relies on a cloud NLP provider, even new incremental releases of the provider could affect the bot behavior due to changes in the provider's parameters or strategies for training the networks used for intent classification. Another critical role of regression testing is making sure that adding new intents does not deteriorate the overall quality of the bot (e.g., due to overlaps between an existing intent and the new one as discussed above).

Performance testing is also challenging in a NLP bot as it requires simulating a large number of concurrent users having (potentially long) conversations with the bot. As such, it is not just about load-testing the access to the platform where the bot is deployed but generating sets of input text that will properly trigger bot conversations.

Security testing. NLP bots are a target of denial-of-service (DDoS) attacks. NLP bots may also suffer from confidentiality and privacy issues as they typically collect data from users and may be able to link that data with the user identity (or proxies of it as location or usernames). And conversely, the bot should not leak internal data as part of the conversation with non-privileged users. Access control methods for bots must be developed and integrated in bot frameworks. Security testing could also involve detecting user trolls that may be trying just to get the bot to answer to some provocative input text to damage the company image.

Back-to-back testing could be used to test the bot quality when using different NLP engines or configurations. For example, a bot deployed over DialogFlow may work better/worse than the same bot deployed over IBM Watson.

Recovery testing should at least test whether, after a crash, the bot is able to recover the conversation at the state where it was before to minimize the crash impact on the user experience with the bot.

Usability and Human Computer Interaction testing evaluate how easy it is for end users to learn and use the bots. A good usability is a combined responsibility of the bot and the platform the bot is deployed on. We need to check that: 1) users know how to use the bot and trigger the conversation with it, 2) the bot is able to benefit from the platform UI features (e.g., use of buttons and other UI controls) to simplify the interaction, and 3) the NLP side of the bot can actually understand what the user is trying to communicate. For chatbots, this understanding is related to the unit and integration testing capabilities discussed above. For voicebots, this also involves testing that the speech-to-text component is able to deal with the multiple accents of users. Actual metrics for this usability evaluation could be based on the length of the conversations, the abandonment rate and the sentiment analysis of the conversation among others.

B. *Testing techniques*

1) Input Domain-Based Techniques: **Equivalence partitioning** splits the input domain into a collection of equivalent classes from which one or several representatives are taken for testing purposes. In NLP bots, each intent represents an equivalence class in itself. For intents with no parameters, we can simplify the test suite by removing utterances matching the same intent more than once (or more than once in the same state if it implements an FSM).

Boundary-Value Analysis is straightforward when intent parameters are enums but has the well-known challenges of detecting bounds when parameter types correspond to predefined data types. There is an additional challenge which is dealing with the frequent "@any" parameter that, in principle, can match any substring of a user utterance. As such, this type of parameters may fail because they match the wrong word but also because they match the wrong number of words. Specific tests for *any* parameters are envisaged.

Random testing for NLP bots involves generating random input sentences. These sentences can be extracted from textual sources available online, produced using generative models such as GPT-3 [20] or with adversarial models [21]. Random values should include both sentences that are matched to the bot intents and others that are not. Furthermore, *fuzz testing* can be used to generate almost correct sequences of user inputs in order to identify whether potentially undesired states can be reached. The FSM can provide information to guide the generation of these almost valid input sequences.

2) Code-based techniques: **Control Flow-Based Criteria**. Not specific for bot testing but we emphasize it as, more often than not, most of the bot testing focus on the intent matching part while ignoring the underlying FSM that drives the bot behavior. Both aspects need to go hand-by-hand. Note that not all frameworks use an internal FSM, instead they go for simpler formalisms (as a simple partial ordering in the intent definitions to describe conversations) but any non-trivial bots end up in a FSM.

Reference Models for Code-Based Testing A reference model for testing bots can be defined by a flow graph that tests the different possible sequences in which intents can be matched (setting upper bounds to avoid infinite cycles). In the case of stateless bots, the reference model will be composed by all the flow graphs that represent all the possible sequences in which its intents are matched (bounding each intent to be matched k times, or by limiting the length of the sequence to k intents to avoid infinite sequences). In the case of bots with an underlying FSM, the flow graph is given by all the bot transition paths going from the initial state until the final state (limiting the cycles to k intents). Tests must execute each one of the paths to ensure a 100% coverage.

3) Fault-based techniques: **Mutation testing** consists of introducing small syntactic changes (i.e., faults) that generate slightly different versions of the bot (called mutants). For instance, a mutation could be produced by assigning the wrong training sentences to an intent or, in the case of bots with underlying FSMs, by changing the conditions associated with its transitions. If a test identifies the difference between the original bot and a mutant, the mutant is "killed". A key challenge is to decide how different is the mutant from the original as NLP engines have a high-level of tolerance (e.g., typos in the input text may be recognized and corrected by the NLP engine, hence mutants that modify the utterances of an intent would not always affect the results of the tests).

4) Model-based Testing Techniques: **Finite State Machines**. As bots conversations themselves are typically described based on a FSM, all the arsenal of testing techniques for FSMs could be ported to NLP bot model-based testing.

Formal Specifications. Given the ambiguity of natural language, formal specifications are not amenable for the NLP component. On the other hand, the FSM, when present, can be analyzed (e.g., using model checking) in order to check desirable correctness properties such as reachability or to generate traces that can act as test cases.

C. Test-related measures

1) Evaluation of the Program Under Test: **Program Measurements That Aid in Planning and Designing Tests** should include the number of intents in the bot, the number of parameters and training sentences per intent, the length of conversation paths, the number of branches in a conversation, etc. Run-time measures should then include the frequency each of the above is matched/used.

Fault density for the NLP part could be defined as the ratio between the number of intents that failed to match a user input targeting that intent and the total number of intents of the bot.

2) Evaluation of the Tests Performed: **Coverage** in NLP bots must include *intent coverage*, defined as the ratio of intents matched at least once during the test and *intent parameter coverage* as the ratio of intent parameters matched at least once during the test. Besides, intents must be considered when creating tests that exercise the different paths in the FSM as matched intents are required to trigger most of them.

V. Conclusion

We have seen the many specificities of testing NLP-intensive bots, mixing "classical" software testing challenges with NLP/AI-based ones. Most of them are still open research challenges for the bot community. As such, we hope to see continuous efforts in this area in the near future and believe this paper can be helpful to characterize them.

Moreover, many of the open problems in software testing [22] such as test suite reduction [23], [24], prioritization [25], flaky tests [26] or diversity [27] are also of interest here. Similarly, well-known testing techniques in other domains, like A/B testing, could be applied to bots. Furthermore, beyond the SWEBOK properties, some testing concepts more specific to nlp-based interfaces may need to be considered.

References

[1] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *ICSE-SEIP'2019*. IEEE, 2019, pp. 291–300.

[2] I. Ozkaya, "What is really different in engineering AI-enabled systems?" *IEEE Softw.*, vol. 37, no. 4, pp. 3–6, 2020. [Online]. Available: https://doi.org/10.1109/MS.2020.2993662

[3] B. C. Hu, R. Salay, K. Czarnecki, M. Rahimi, G. M. K. Selim, and M. Chechik, "Towards requirements specification for machine-learned perception based on human performance," in *7th IEEE Int. Workshop on Artificial Intelligence for Requirements Engineering, AIRE@RE 2020*, 2020, pp. 48–51.

[4] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5193–5254, 2020.

[5] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM TOPLAS*, vol. 8, no. 2, pp. 244–263, 1986.

[6] P. Bourque and R. E. Fairley, Eds., *SWEBOK: Guide to the Software Engineering Body of Knowledge*, version 3.0 ed. IEEE Computer Society, 2014. [Online]. Available: http://www.swebok.org/

[7] J. Pereira and Ó. Díaz, "Chatbot dimensions that matter: Lessons from the trenches," in *Int. Conf. on Web Engineering*. Springer, 2018, pp. 129–135.

[8] Botium. [Online]. Available: https://www.botium.ai/

[9] Zypnos. [Online]. Available: https://zypnos.com/

[10] chatbottest. [Online]. Available: https://chatbottest.com/

[11] QBox. [Online]. Available: https://qbox.ai/

[12] Dashbot. [Online]. Available: https://www.dashbot.io/

[13] E. Ruane, T. Faure, R. Smith, D. Bean, J. Carson-Berndsen, and A. Ventresque, "Botest: A framework to test the quality of conversational agents using divergent input examples," in *23rd Int. Conf. on Intelligent User Interfaces Companion*, ser. IUI '18 Companion, 2018.

[14] J. Guichard, E. Ruane, R. Smith, D. Bean, and A. Ventresque, "Assessing the robustness of conversational agents using paraphrases," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2019, pp. 55–62.

[15] M. T. Ribeiro, T. Wu, C. Guestrin, and S. Singh, "Beyond accuracy: Behavioral testing of NLP models with CheckList," *arXiv preprint arXiv:2005.04118*, 2020.

[16] S. Bravo-Santos, E. Guerra, and J. de Lara, "Testing chatbots with Charm," in *Quality of Information and Communications Technology*. Cham: Springer International Publishing, 2020, pp. 426–438.

[17] J. Bozic, O. A. Tazl, and F. Wotawa, "Chatbot testing using AI planning," in *IEEE Int. Conf. On Artificial Intelligence Testing (AITest)*, 2019, pp. 37–44.

[18] J. Bozic and F. Wotawa, "Testing chatbots using metamorphic relations," in *Testing Software and Systems*, C. Gaston, N. Kosmatov, and P. Le Gall, Eds. Springer International Publishing, 2019, pp. 41–55.

[19] R. Ren, J. W. Castro, S. T. Acuña, and J. de Lara, "Evaluation techniques for chatbot usability: A systematic mapping study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 11n12, pp. 1673–1702, 2019.

[20] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[21] J. X. Morris, E. Lifland, J. Lanchantin, Y. Ji, and Y. Qi, "Reevaluating adversarial examples in natural language," 2020.

[22] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *FOSE'07*. IEEE, 2007, pp. 85–103.

[23] G. Rothermel, M. J. Harrold, J. Von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 219–249, 2002.

[24] F. Elberzhager, A. Rosbach, J. Münch, and R. Eschbach, "Reducing test effort: A systematic mapping study on existing approaches," *Information and Software Technology*, vol. 54, no. 10, pp. 1092–1106, 2012.

[25] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[26] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *FSE'2014*, 2014, p. 643–653.

[27] B. Baudry and M. Monperrus, "The multiple facets of software diversity: Recent developments in year 2000 and beyond," *ACM Comput. Surv.*, vol. 48, no. 1, Sep. 2015.