Testing chatbots with Charm

Sergio Bravo-Santos, Esther Guerra, and Juan de Lara

Modelling and Software Engineering Research Group
`http://miso.es`
Computer Science Department
Universidad Autónoma de Madrid (Spain)
{`sergio.bravos, esther.guerra, juan.delara`}`@uam.es`

Abstract. Chatbots are software programs with a conversational user interface, typically embedded in webs or messaging systems like Slack, Facebook Messenger or Telegram. Many companies are investing in chatbots to improve their customer support. This has led to a proliferation of chatbot creation platforms (e.g., Dialogflow, Lex, Watson). However, there is currently little support for testing chatbots, which may impact in their final quality.
To alleviate this problem, we propose a methodology that automates the generation of *coherence*, *sturdiness* and *precision* tests for chatbots, and exploits the test results to improve the chatbot precision. The methodology is supported by a tool called Charm, which uses Botium as the backend for automated test execution. Moreover, we report on experiments aimed at improving Dialogflow chatbots built by third parties.

Keywords: Chatbots · Testing · Botium · Dialogflow.

1 Introduction

Chatbots – also called conversational agents – are software programs that interact with users via conversation in natural language (NL) [9]. Many companies are developing chatbots to provide access to their services or automate customer support, and they are increasingly being used to automate software engineering tasks [4, 6]. Their use is booming as they do not require installing dedicated apps but can be embedded in social networks – like Slack, Telegram or Twitter – for their use in mobile devices, as if talking with a colleague.

Because of this growing interest in chatbots, many tools for their development have appeared, such as Google's Dialogflow[1], IBM's Watson Assistant[2], Microsoft's bot framework[3] or Amazon's Lex[4]. Some of them are cloud-based low-code development environments that greatly facilitate the main chatbot construction steps, from the application of NL processing (NLP) for identifying the

[1] `https://dialogflow.com/`
[2] `https://www.ibm.com/cloud/watson-assistant/`
[3] `https://dev.botframework.com/`
[4] `https://aws.amazon.com/en/lex/`

user intents, to the chatbot deployment in social networks. However, these tools barely provide support for testing chatbots, even if testing is essential to ensure the chatbot quality. At most, they offer a console where developers can manually test if the chatbot reacts properly to the NL inputs. While this helps during the development, a proper software process requires systematic, automatable testing mechanisms.

To address this need, a few chatbot testing tools are starting to emerge, most notably Botium[5]. This tool successfully automates the chatbot testing process. Moreover, it permits synthesizing an initial set of test cases derived from the training phrases of the chatbot. However, the generated test cases only consider basic conversation flows, and need to be extended by hand. Our aim is to automate this manual process as much as possible.

In this paper, we propose a methodology for chatbot testing that extends the test case synthesizer of Botium to cover more complex cases, such as context-dependent conversations. The generated test cases have two aims: testing the robustness of the NLP engine, and the precision of the chatbot to identify the user intents. For this purpose, our tests include variations of the chatbot training phrases, constructed via *fuzzing*/mutation functions [12]. Moreover, the test results can be used to improve the chatbot precision. The method is supported by a tool called Charm, and has been evaluated through some experiments on chatbots developed by third parties.

In the remainder of the paper, Section 2 provides background on chatbots and their testing with Botium; Section 3 presents our approach for test synthesis; Section 4 describes our methodology and its tool support; Section 5 reports on an initial evaluation; Section 6 compares with related work; and Section 7 presents the conclusions and lines of future work.

2 Background on chatbots and their testing

This section overviews the working scheme of chatbots (Section 2.1) and how they can be tested with Botium (Section 2.2).

2.1 What's in a chatbot

Chatbots are programs with a conversational user interface. As Figure 1 illustrates, the interaction starts when the user writes a sentence or *utterance* (label 1). Then, the chatbot tries to match the utterance to the most appropriate *intent* among a predefined set (label 2). For example, upon the receipt of the user utterance *"what types of pizza do you have?"*, a chatbot for food delivery would recognize that the user intent is obtaining information about the availability of some kind of food, and would reply with a list of pizza types. To identify the intent that corresponds to an utterance, intent definitions include sample phrases (i.e., different ways to express the intent) which are used for training the chatbot.

[5] `https://www.botium.ai/`

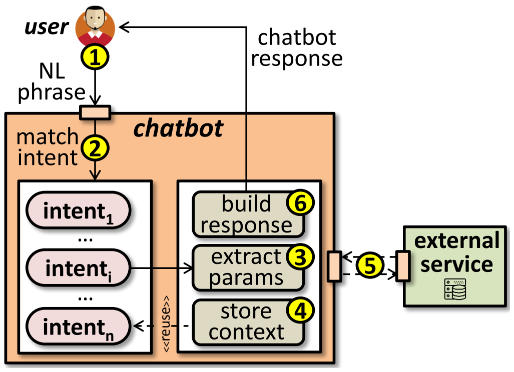


Fig. 1: Chatbot working scheme.

Upon matching an intent, the chatbot may need to extract information from the utterance (label 3). In the previous example, it may need to know the query target, which is *"types of pizza"*. Each piece of information is called *parameter*, and is typed by an *entity* which can be either predefined (e.g., date) or chatbot-specific (e.g., food type). Entities define a list of possible values (e.g., pizza, noodles) and synonyms, and some platforms like Dialogflow allow *fuzzy matching* to overcome misspellings and mistakes. If a parameter is mandatory but the utterance does not include it, the chatbot may ask for it in a follow-up intent. Moreover, chatbots sometimes need to store information about a conversation (e.g., desired type of pizza) to reuse it in subsequent intents. In Dialogflow, the conversation state is stored in *contexts* (label 4).

Finally, the chatbot may need to invoke an external service (e.g., the information system of a food delivery shop) to handle the user intent (label 5), and ultimately responds to the user (label 6) with a text, media elements, or widgets specific of the deployment platform (e.g., buttons in Telegram).

2.2 Testing chatbots with Botium

Botium is a suite of open-source components for automated chatbot testing. It communicates with the chatbot under test via connectors. These are available for many chatbot platforms (like Dialogflow, Watson or Lex), and new ones can be added. Botium executes all test cases found in a given folder against the chatbot. It follows a behaviour-driven development approach [10] similar to Cucumber[6], in which test cases consist of *convo* files that hold the global structure of the test conversation, and *utterance* files that contain the phrases used in the conversation.

As an example, Listing 1 shows a convo where the user (#me) provides any utterance in order_drink_utterance (i.e., any phrase in Listing 2), and the chatbot (#bot) is expected to match the intent order.drink. Overall, the convo would be

[6] https://cucumber.io/

executed three times (once per utterance). As a result, Botium reports the number of passed and failed tests, the reason for failure, and a confusion matrix with the percentage of tests that matched the expected intent. The latter matrix allows detecting loosely defined intents.

```
1 #me
2 order_drink_utterance
3
4 #bot
5 INTENT order.drink
```

Listing 1: Convo file.

```
order_drink_utterance

do you have iced latte?
can I get a small iced cappuccino with low−fat milk?
I want tea
```

Listing 2: Utterance file.

While test cases can be created by hand, Botium also supports their automated generation from the chatbot specification. Specifically, it generates one convo and one utterance file per intent, the latter containing the intent training phrases. The generated tests are simple, e.g., they do not consider conversations with context or chatbot responses. Hence, the developer needs to create additional tests to tackle those scenarios. In the next section, we propose an extension of these test synthesis capabilities.

3 Test synthesis

Charm extends the set of test cases generated by Botium in two ways. First, it produces further convos to test behaviour uncovered in the synthesized test set. This process is explained in Section 3.1. Second, Charm augments the utterance files by means of mutation. This technique is introduced in Section 3.2.

3.1 Convo generation

Charm produces convos to test the following aspects:

– **Chatbot response**: Botium produces convos that specify the intent that should be matched (see, for example, line 5 in Listing 1). Charm extends these convos to include and assess the expected chatbot response as well.
– **Required parameters**: Intents may have required parameters, and the chatbot response may depend on their value (or lack of value). Hence, Charm extends the base convos to tackle different parameter values.
– **Context**: Charm generates new convos for testing the use of contexts (i.e., previously stored information). To this aim, for every intent that uses context variables, it creates all possible convo combinations that fill those variables and lead to the intent.

Example. Listing 3 shows a convo generated by Charm for an intent with context and two required parameters: type of drink and delivery method. The convo emulates an interaction where the user utterance omits the delivery method (line 2). This triggers a follow-up question of the chatbot asking for it (line 6), to which the user replies delivery (line 9). Then, the chatbot recaps the order details and asks for confirmation (line 13), which requires retrieving the ordered drink and delivery method from the previous context. Thus, to generate this convo, Charm needs to statically build a conversation flow that feeds the context with the necessary information, as done in lines 1–9.

```
 1  #me
 2  order_drink_nodeliv_utt
 3
 4  #bot
 5  INTENT order.drink
 6  order_drink_nodeliv_response
 7
 8  #me
 9  delivery
10
11  #bot
12  INTENT order.drink
13  order_drink_confirmation
14
15  #me
16  order_drink_nodeliv_yes_utt
17
18  #bot
19  INTENT order.drink.yes
20  order_drink_confirmation_yes
```

Listing 3: Convo for testing intent with context.

```
 1  me:
 2  Two medium cappuccinos
 3
 4  bot:
 5  Would you like delivery or pickup?
 6
 7
 8  me:
 9  delivery
10
11  bot:
12  You want two medium cappuccinos
13     for delivery, is that right?
14
15  me:
16  Yes
17
18  bot:
19  Have a nice day!
20
```

Listing 4: Conversation.

As an example, Listing 4 shows an instance of the execution of the convo with concrete utterances. We use the same line numbers as in Listing 3 to facilitate traceability.

3.2 Utterance generation

Starting from the utterance set generated by Botium, Charm creates new utterance variants by applying the mutation operators shown in Table 1. We distinguish the following four kinds of mutation operators, which are applied with a customizable probability:

- **Character** operators emulate typing errors according to a given probability. Specifically, swap-char swaps a character with another one, swap-char-close swaps one character to another one which is close in the keyboard, and delete-char deletes one character.
- **Language** operators translate an utterance between a list of user-defined or random languages, and the result is translated back to the initial language. The goal is creating utterances with equivalent meaning but different form.
- **Word** operators change a word (adjectives, nouns or adverbs) by synonyms or antonyms. The aim is creating utterances accepted by the same intents as the original utterance.
- **Number** operators substitute numbers by equivalent words and vice versa.

Table 1: Mutation operators for utterances.

Mutation	Description	Example
Character		
swap-char	swaps a character to any other character	hello → hkllo
swap-char-close	swaps a character to another one close in the keyboard	hello → hwllo
delete-char	deletes a character	hello → hllo
Language		
translation-chain	translates between a list of languages	hello → hola → hi
Word		
word-to-synonym	changes an adjective, adverb or noun to a synonym	2 pants → 2 trousers
word-to-antonym	changes an adjective or adverb to an antonym	hot tea → cold tea
Number		
number-to-word	changes a number into an equivalent word	2 pants → two pants
word-to-number	changes a word to a number	two pants → 2 pants

4 Testing methodology and tool support

In this section, we first introduce our proposed methodology for testing chatbots (Section 4.1), and then we overview our supporting tool CHARM (Section 4.2).

4.1 Testing process

Figure 2 shows the chatbot testing process in our tool CHARM. It supports three kinds of tests: coherence, sturdiness and precision.

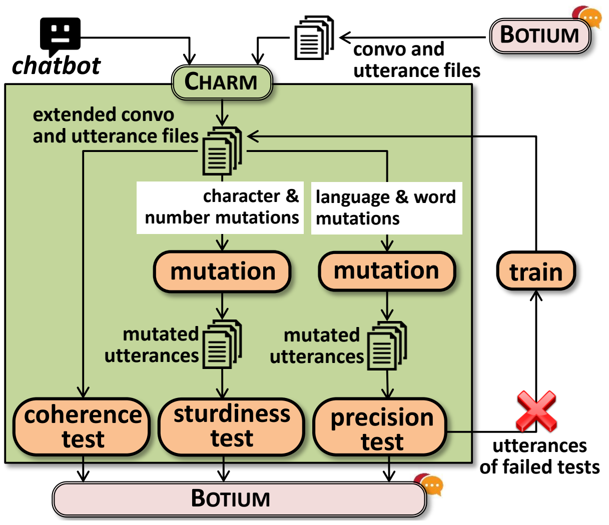

chatbot
CHARM
convo and utterance files
BOTIUM
extended convo and utterance files
character & number mutations
language & word mutations
mutation
mutation
train
mutated utterances
mutated utterances
coherence test
sturdiness test
precision test
utterances of failed tests
BOTIUM

Fig. 2: CHARM's testing process.

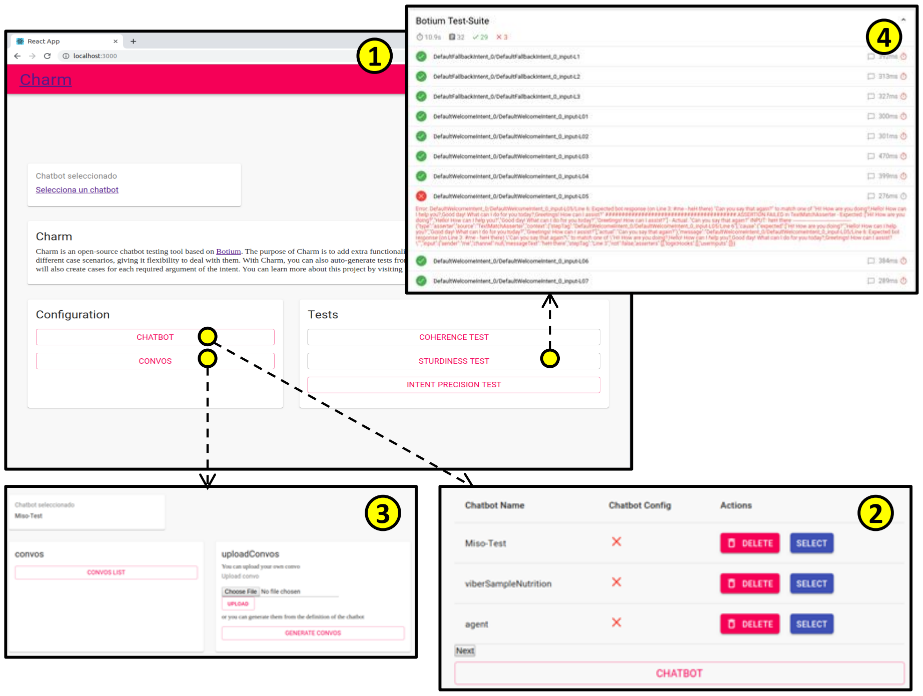
First, CHARM invokes BOTIUM to create the base convo and utterance files, and extends the convo files as explained in Section 3.1. Then, depending on the kind of test, Charm creates new utterances by applying a subset of the mutation operators detailed in Section 3.2. This stage may require the intervention of the tester to confirm that the new utterances preserve the original utterance semantics. Finally, the test cases are executed atop Botium, and the results are interpreted according to the test kind:

– **Coherence test**: This is the simplest test. It executes the extended convo files but does not perform any utterance mutation. This test is typically performed first, to detect coarse-grained defects like duplicated training phrases in different intents, or too similar intents and entities.
– **Sturdiness test**: This test assesses how good the chatbot is at dealing with typing mistakes or different writing styles. For this purpose, Charm applies the *character* mutations to emulate typing mistakes, and the *number* mutations to have a same utterance written in different ways (numbers vs words). This type of test actually evaluates the robustness of the NLP engine of the underlying chatbot platform. If the results are deemed bad, some platforms allow fine-tuning the intent matching process, e.g., by enabling *fuzzy matching*.
– **Precision test**: The precision test evaluates the ability of the chatbot to predict the correct intent when utterances have a different formulation from the intent training phrases. To do so, Charm produces new utterances using the *language* and *word* mutations. If a test with a mutated utterance fails, then the utterance can be used as a training phrase to improve the chatbot precision. This testing-improvement cycle can be repeated until the chatbot precision is deemed adequate.

4.2 Tool support

Charm is implemented in Python and uses Botium as a backend. The tool is freely available at `https://charmtool.github.io/Charm/`. It permits generating convo files and parameterizing the distribution probabilities of the mutation operators programmatically.

In addition, we provide a web application, implemented in Django and React, that enables the use of Charm from a web-based user interface. Figure 3 shows this web application, which can be accessed from the webpage of Charm. Its main page, with label 1, shows on the left top the chatbot that is currently active. If no chatbot has been selected, as in the figure, the user can select one from the list of available chatbots, or upload a new one. The latter is done using the page with label 2, where the user can also delete existing chatbots. We currently support Dialogflow chatbot definitions, but we plan to support further formats in the future. The user can upload hand-made convos, and generate convos using Botium, from the page with label 3. Finally, the main page contains buttons to execute the presented coherence, sturdiness and precision tests. As an example, the page with label 4 shows the results of the sturdiness test. The displayed report is generated by Botium.



Charm

Charm is an open-source chatbot testing tool based on Botium. The purpose of Charm is to add extra functionali
different case scenarios, giving it flexibility to deal with them. With Charm, you can also auto-generate tests fro
will also create cases for each required argument of the intent. You can learn more about this project by visiting

```
10.9s  32  ✓ 29  ✕ 3
✓ DefaultFallbackIntent_0/DefaultFallbackIntent_0_input-L1      312ms
✓ DefaultFallbackIntent_0/DefaultFallbackIntent_0_input-L2      313ms
✓ DefaultFallbackIntent_0/DefaultFallbackIntent_0_input-L3      327ms
✓ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L01       300ms
✓ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L02       301ms
✓ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L03       470ms
✓ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L04       399ms
✕ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L05       276ms
Error: DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L05/Line 6: Expected bot response (on Line 3: #me - heH there) "Can you say that again?" to match one of "Hi! How are you doing?,Hello! How can I help you?,Good day! What can I do for you today?,Greetings! How can I assist?" ########## ASSERTION FAILED in TextMatchAsserter - Expected: ["Hi! How are you doing?","Hello! How can I help you?","Good day! What can I do for you today?","Greetings! How can I assist?"] - Actual: "Can you say that again?" INPUT: heH there [illegible]
✓ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L06       384ms
✓ DefaultWelcomeIntent_0/DefaultWelcomeIntent_0_input-L07       289ms
```

Fig. 3: Web-based user interface of CHARM: (1) Main window. (2) Chatbot management page. (3) Page for uploading and generating convos. (4) Results of sturdiness test (report obtained from BOTIUM).

Chatbot Name	Chatbot Config	Actions
Miso-Test	✕	DELETE SELECT
viberSampleNutrition	✕	DELETE SELECT
agent	✕	DELETE SELECT

5 Evaluation

In this section, we report on the results of an experiment aimed at answering the following two research questions (RQs):

RQ1 *Can* CHARM *uncover problems in chatbots that the default test cases generated by* BOTIUM *do not detect?*

RQ2 *Can the iterative testing process of* CHARM *improve the chatbot quality?*

5.1 Experiment set-up

The experiment considers the three Dialogflow chatbots shown in Table 2. The first one was built by us, and the other two are third-party chatbots found on github. The Baseline[7] chatbot has neither contexts nor entities, and so, the chatbot responses do not depend on parameter values or previous conversation states. The Nutrition[8] chatbot has 7 entities, some of them with more than 100 entries,

[7] https://github.com/CharmTool/Charm/blob/master/chatbots/Miso-Test.zip
[8] https://github.com/Viber/apiai-nutrition-sample

and it defines several intents with required parameters, so conversations can become complex. The RoomService[9] chatbot has 5 intents, 1 of them dependant on another via a context, and it uses 4 predefined entities and 1 chatbot-specific entity.

Table 2: Chatbots under test.

Chatbot	**#Intents**	**#Entities**	**#Contexts**
Baseline	4	0	0
Nutrition	4	7	0
RoomService	5	5	1

In the experiment, we set a maximum of 10 utterances per utterance file. Moreover, the mutation operators were applied to each utterance with a certain probability: in the sturdiness tests, the application probability was 30% for swap-char-close and delete-char, 20% for number-to-word and word-to-number, and 0% for swap-char; while in the precision tests, we gave probability 30-45% to translation-chain, and 5% to the word mutation operators. These values were decided after calibration, based on the quality of the resulting tests.

5.2 Experiment execution

We run each type of test on every chatbot, and next, we extended the chatbot training set with the utterances of the failing cases of the precision tests, performing two improvement cycles. Table 3 summarizes the results. All chatbots obtained perfect score in the coherence test (2^{nd} column). This means that the chatbots have no evident errors in their specification, and the default Botium tests detect no faults.

Table 3: Results of the experiment.

	Coherence, 1st cycle		Sturdiness, 1st cycle		Sturdiness, fuzzy matching		Precision, 1st cycle		Precision, 1st cycle new training		Precision, 2nd cycle		Precision, 2nd cycle new training	
	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail
Baseline	32	0	31	1	31	1	29	1	30	0	30	1	31	0
Nutrition	48	0	43	5	46	2	45	2	47	0	48	1	49	0
RoomService	32	0	29	3	28	4	29	2	31	0	33	2	35	0

To test sturdiness, the character and number mutation operators produced 32, 48 and 32 additional test utterances. All chatbots failed some test case (3^{rd} column). If we activate *fuzzy matching* on the problematic entities (4^{th} column), then the results of Baseline do not change because it has no entities, the results

[9] `https://github.com/dialogflow/dialogflow-java-client-v2`

of Nutrition improve, but RoomService worsens. The latter is because the chatbot defines an entity "room name" with entries A, B and C, and CHARM generated an utterance with a different room name, which the chatbot (incorrectly) took as valid. This shows that *fuzzy matching* is not appropriate for this intent.

The precision tests produced the same number of utterances as the sturdiness tests, though using the language and word operators. Moreover, we manually filtered 4 of the generated test utterances out, as they were meaningless. From the remaining test cases, all chatbots failed some (5th column), so we trained the chatbots with the utterances of the failed cases, after which all tests succeeded (6th column). Next, we applied the precision test with new mutated utterances, obtaining fewer errors than in the first cycle (7th and 8th columns).

5.3 Discussion

Overall, we can answer RQ1 and RQ2 positively: CHARM produced tests that revealed faults, and also helped in improving the quality of the chatbots. However, we need to perform further experiments with more complex chatbots to strengthen this assessment. We observed that CHARM synthesized convo files to test the context in chatbot RoomService. While CHARM can generate tests that detect chatbot defects, there is still a manual step to filter meaningless utterances in precision tests. For instance, in our experiment, we had to remove around 3% of the automatically generated utterances.

To get an intuition of the synthesized tests, Table 4 shows some of the utterances generated by CHARM, together with the mutation operator that produced them. The last column of the table indicates whether the generated utterance was manually discarded or not. For example, we removed the last utterance shown for chatbot Nutrition, as the translation chain produced a sentence with almost opposite meaning to the original.

Table 4: Sample of generated utterances.

Chatbot	**Utterance**	**Mutated utterance**	**Operator**	**Discarded?**
Baseline	just going to say hi	just come to say hello	translation-chain	no
Baseline	when are the meetings?	When is the meetings?	translation-chain	no
Baseline	good luck	good muck	swap-char-close	no
Nutrition	nutrition analysis	food analysis	word-to-synonym	no
Nutrition	calories in 4 oz of steak	calories in four oz of steak	number-to-word	no
Nutrition	how many calories in one big mac	how many calories in 1 big mac	word-to-number	no
Nutrition	does a kiwi contain vitamin A	Not one kiwi fruit contains vitamin a	translation-chain	yes
RoomService	is there any room free tomorrow?	Any rooms free	translation-chain	no
RoomService	Do you have rooms for this monday?	Do you have rooms for thismonday?	delete-char	no

6 Related work

While there are many tools for chatbot development, their support for testing is scarce. Most development platforms (like Dialogflow, Lex or Watson) provide a web chat console that permits informal, manual testing of the chatbots. Approaches based on programming languages – like Rasa[10], which is built atop Python – can rely on the debugging and testing support offered by the programming language itself. Only a few platforms, like Dialogflow, offer debugging facilities to inspect the matched intent and related information. In addition, Dialogflow includes checks of the chatbot quality, like detecting intents with similar training phrases.

Some companies have developed their own chatbot testing tools. For example, haptik.ai provides a testing tool[11] that automates the interaction with the chatbot via simple scripts, and can be integrated with automation servers such as Jenkins. BOTIUM can also be integrated in testing flows using Jenkins. However, these tools require manual building or extension of the test suites, which our work aims to automate.

Regarding academic proposals, in [1], the authors use AI planning techniques to generate tests traversing the conversation flow. More similar to us, the metamorphic chatbot testing approach in [3] applies mutation operators (e.g., replacing a word by a synonym, or a number by another one) to a set of utterances to produce follow-up test cases, which should match the same intent. In a similar vein, BoTest [8] creates divergent inputs (word order errors, incorrect verb tense, synonyms) from an initial utterance set. We also rely on mutation, but in addition, we classify our mutation operators to obtain different types of tests (to test either the robustness of the NLP engine or the precision of the intent definitions), provide automation on top of BOTIUM, and a methodology for chatbot improvement.

To reduce the human cost of chatbot testing, *Bottester* [11] simulates users who interact with chatbots, and collects some interaction metrics like the answer frequency, the response time or the precision of the intent recognition. While *Bottester* targets chatbots created with in-house technology, CHARM is based on BOTIUM and so can test chatbots for the major chatbot creation platforms. Moreover, our testing process covers different chatbot aspects and provides a cycle of chatbot improvement.

CHARM is focused on testing the NL aspect of the chatbot, but other (non-functional) aspects need to be tested as well, like the communication with external services or the chatbot security [2]. For example, *Alma*[12] is a chatbot that helps in evaluating Messenger and Telegram bots across seven categories: personality, onboarding, understanding, navigation, error management, intelligence and response time. While *Alma* is based on questions to the chatbot users, we support automated testing. One of the decisive aspects for chatbot acceptance is

[10] `https://rasa.com/`
[11] `https://haptik.ai/tech/automating-bot-testing/`
[12] `http://chatbottest.com`

their usability. Some heuristics for bot usability have been proposed[13], but more actionable usability patterns – possibly integrated within chatbot development tools – and automated means for usability evaluation are needed [7].

7 Conclusions and future work

The increasing use of chatbots for varying activities makes necessary techniques to ensure their quality. This paper contributes to solve this need by proposing a set of techniques for automated chatbot test synthesis, a methodology supporting three different types of tests, and a supporting tool that uses Botium for test automation.

In the future, we would like to extend our set of mutation operators (for example, to enable adversarial text generation [5]), support new types of tests, improve the functionality of the Charm service, and enable the integration of Charm with continuous testing and integration workflows.

Acknowledgments. We would like to thank the anonymous reviewers for their comments. This work has been partially funded by the Spanish Ministry of Science (project MASSIVE, RTI2018-095255-B-I00) and the R&D programme of Madrid (project FORTE, P2018/TCS-4314).

References

1. Bozic, J., Tazl, O.A., Wotawa, F.: Chatbot testing using AI planning. In: AITest. pp. 37–44. IEEE (2019)
2. Bozic, J., Wotawa, F.: Security testing for chatbots. In: ICTSS. LNCS, vol. 11146, pp. 33–38. Springer (2018)
3. Bozic, J., Wotawa, F.: Testing chatbots using metamorphic relations. In: ICTSS. LNCS, vol. 11812, pp. 41–55. Springer (2019)
4. Erlenhov, L., de Oliveira Neto, F.G., Scandariato, R., Leitner, P.: Current and future bots in software development. In: Proc. the 1st International Workshop on Bots in Software Engineering, BotSE@ICSE. pp. 7–11. IEEE / ACM (2019)
5. Jin, D., Jin, Z., Zhou, J.T., Szolovits, P.: Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In: AAAI (2020)
6. Pérez-Soler, S., Guerra, E., de Lara, J.: Collaborative modeling and group decision making using chatbots in social networks. IEEE Softw. **35**(6), 48–54 (2018)
7. Ren, R., Castro, J.W., Acuña, S.T., de Lara, J.: Evaluation techniques for chatbot usability: A systematic mapping study. International Journal of Software Engineering and Knowledge Engineering **29**(11&12), 1673–1702 (2019)
8. Ruane, E., Faure, T., Smith, R., Bean, D., Carson-Berndsen, J., Ventresque, A.: Botest: A framework to test the quality of conversational agents using divergent input examples. In: IUI Companion. ACM (2018)
9. Shevat, A.: Designing bots: Creating conversational experiences. O'Reilly (2017)
10. Solís, C., Wang, X.: A study of the characteristics of behaviour driven development. In: 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA. pp. 383–387. IEEE Computer Society (2011)

[13] `https://haptik.ai/blog/usability-heuristics-chatbots/`

11. Vasconcelos, M., Candello, H., Pinhanez, C., dos Santos, T.: Bottester: Testing conversational systems with simulated users. In: IHC. pp. 73:1–73:4. ACM (2017)
12. Zeller, A., Gopinath, R., Böhme, M., Fraser, G., Holler, C.: Mutation-based fuzzing. In: The Fuzzing Book. Saarland University (2019), `https://www.fuzzingbook.org/html/MutationFuzzer.html`, retrieved June 2020