Automating the Measurement of Heterogeneous Chatbot Designs

Pablo C. Cañizares
Universidad Autónoma de Madrid
Madrid, Spain
pablo.cerro@uam.es

Sara Pérez-Soler
Universidad Autónoma de Madrid
Madrid, Spain
sara.perezs@uam.es

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
esther.guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
juan.delara@uam.es

ABSTRACT

Chatbots are being increasingly used to provide a natural language interface to all kinds of software services. However, while there are many platforms and tools for chatbot development, they typically lack support to statically measure properties of the designed chatbots, as indicators of their size, complexity, quality or usability, and facilitating comparison.

To attack this problem, in this paper we propose a suite of 20 metrics for chatbot designs. The metrics are defined on a neutral chatbot design language, becoming independent of the implementation platform. We have developed a tool, called Asymob, which supports the translation of chatbots defined in several platforms into this neutral format to perform the measurements. As a proof-of-concept, we evaluate the metrics over a collection of Dialogflow and Rasa chatbots from several sources and open-source repositories. Our metrics helped detecting quality issues statically, and served as a basis for comparing chatbots from different origins and built using different technologies.

CCS CONCEPTS

• **Human-centered computing** → **Natural language interfaces**; • **General and reference** → **Metrics**; • **Social and professional topics** → **Quality assurance**;

KEYWORDS

Chatbot design, metrics, quality assurance

ACM Reference Format:
Pablo C. Cañizares, Sara Pérez-Soler, Esther Guerra, and Juan de Lara. 2022. Automating the Measurement of Heterogeneous Chatbot Designs. In *Proceedings of ACM SAC Conference (SAC'22).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

SAC'22, April 25 - April 29, 2022, Brno, Czech Republic

ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Chatbots are becoming popular to access all sorts of services (e.g., banking, shopping, tourism, health) using conversation in natural language [36], and they are being increasingly used to assist in software engineering activities [17]. For this reason, many platforms are available for their construction [29], like Google's Dialogflow [12], Amazon Lex [18], IBM's Watson [38], the Microsoft bot framework [21], and many others by smaller companies and communities like Rasa [32], Pandorabots [26], or FlowXO [15].

While there are many chatbot implementation platforms, their support for chatbot quality assurance is limited [29]. Since chatbots are a kind of software, their construction should follow sound engineering principles. Some recent approaches [4, 5, 13] and tools [3] propose methods for testing chatbots. Dynamic testing is essential to ensure the quality of the resulting chatbot, but it requires having a functional, deployed chatbot; it demands high effort for creating testing phrases and oracles; and it is time-consuming.

In this paper, we propose the use of metrics as a tool to guide and control the quality of the chatbot throughout its development, becoming a complement to dynamic testing. Metrics are an accepted mechanism for assessing and controlling properties of software products and processes [14]. They are complementary to dynamic testing as they can be used at design time, even when the chatbot is not yet functional. They can discover issues (e.g., regarding the design complexity and size) which are not the target of dynamic testing, and can be used to trigger recommendations for the improvement of the chatbot design. However, to our knowledge, there is hardly any proposal for the practical application of metrics to chatbot designs.

We argue that static metrics for chatbots can be useful to detect potential problems related to user experience (e.g., complex conversation flows, hard-to-read chatbot answers); as indicators of chatbot complexity; to compare properties of heterogeneous chatbots; to discover chatbot commonalities and cluster similar chatbots; and to understand how different implementation platforms can impact on the chatbot design. Ultimately, the availability of metrics may have a notable impact on current bot development practices and tools, helping to increase the quality of chatbots.

To pursue this goal, we propose a suite of 20 static metrics for chatbot designs, and an accompanying tool called Asymob that supports their evaluation over heterogeneous chatbot implementations.

To avoid reimplementing the metrics for every chatbot implementation platform, Asymob defines the metrics over a neutral design notation called Conga [28], and provides importers from several platforms into Conga. We report on an evaluation applying the metrics over Dialogflow and Rasa chatbots from public repositories, and over predefined chatbots provided by the implementation platforms. Our experiment reveals quality issues in some chatbots, and shows that the metrics can serve as a basis for comparing chatbots from different sources and built using different technologies.

The rest of the paper is organized as follows. Section 2 explains the basics about chatbots. Section 3 revises related work. Section 4 proposes a suite of chatbot metrics over a neutral chatbot design notation. Section 5 describes tool support, and Section 6 evaluates the metrics over chatbots from several sources (predefined, open-source repositories) and technologies (Rasa, Dialogflow). Finally, Section 7 concludes and outlines lines for future work.

2 AN OVERVIEW OF CHATBOTS

Chatbots are conversational software systems with a natural language interface to existing services, like those in banking or shopping. Figure 1 shows a diagram with their typical working scheme. Normally, the user starts the interaction by providing an *utterance* – a phrase in natural language – (step 1) which the chatbot processes to give a proper response (step 6). The interaction can be using text (e.g., if the chatbot is embedded in a social network like Telegram[1]) or voice (e.g., if the chatbot is deployed on smart speakers like Amazon echo[2]). The processing of the user utterance involves a number of steps, which we detail next.

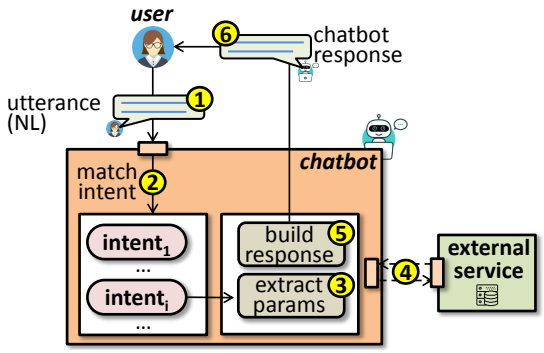

Figure 1: Chatbot working scheme.

Most chatbots are designed around a set of *intents*. These are conversation topics the chatbot aims at recognizing (step 2 in the figure), related to the offered functionality. Depending on the implementation platform, intents are defined either using *regular expressions* (e.g., as in Pandorabots [26]) or with *training phrases* that become interpreted using natural language processing (NLP). Additionally, intents can include *parameters* identifying relevant information pieces to be extracted from user utterances (step 3).

As an example, a chatbot for a cafeteria would define an intent to recognize the users' orders. This intent would be matched by phrases like *"I'd like a medium cappuccino"*, from which the chatbot would extract two parameters: the type of drink (*cappuccino*) and its size (*medium*). Parameters are typed by *entities*, which can be pre-existing in the platform (e.g., dates or numbers) or user-defined for a specific domain (e.g., the type and size of drinks). User-defined entities declare a list of literals (e.g., *medium* and *large* for the size of drinks) with their synonyms.

Upon receiving a user utterance, the chatbot matches the more likely intent, performs some predefined actions to handle the intent, such as accessing an external service (step 4), and composes a text/voice response (step 5) which may incorporate media elements (e.g., images, links) or widgets supported by the social network (e.g., buttons in Telegram). For example, the cafeteria chatbot may need to access an information system to check the availability of discounts and annotate the order, and the response may report the final price. If the chatbot lacks a matching intent for a user utterance, it can trigger a *fallback* intent to ask for clarification.

Typically, conversations are structured into *flows* that intertwine user utterances and chatbot responses. As an example, upon the user utterance *"I'd like a medium cappuccino"*, the chatbot may answer *"Would you like something to eat?"*, leading to a new user interaction (e.g., *"No thanks"*), and so on, according to the defined conversation flow within the chatbot design.

Moore and Arar [23] propose a classification of chatbots depending on their conversation style. *System-centric* chatbots answer user queries or interpret commands by means of 2-turn conversations (i.e., each user turn starts a new conversation and the chatbot lacks state). *Content-centric* chatbots act as an interface for FAQs, typically providing long document-like responses that may not be appropriate for voice-based interfaces or mobile devices with small screens. *Visual-centric* chatbots present buttons and other widgets to facilitate user interaction, in a style borrowed from mobile phones. Finally, *conversation-centric* chatbots mimic human conversations, offering conversation management utterances (e.g., "*What do you mean?*") and short responses. This latter type of chatbots are normally preferred because their conversation style suits a wider variety of devices and engages better in natural conversations.

3 RELATED WORKS ON CHATBOT QUALITY ASSESSMENT

Since the early days of conversational systems [39], researchers have proposed ways for evaluating their quality. For example, PARADISE [37] is an early framework based on the correlation of performance and user satisfaction.

Recently, the popularity of chatbots has raised concerns on proper conversational design. For example, IBM's Natural Conversation Framework [24] proposes conversation patterns [25] and design principles [23, 34]. The latter include guidelines like *recipient design* (i.e., allow multiple conversation paths for different user types), *minimization* (i.e., use concise chatbot answers), and *repair* (i.e., provide support for clarifications). In this line, *Chatbottest* [9] defines guidelines for chatbot design issues in categories like answering, error management, intelligence, navigation, personality and understanding. However, the burden is on the developer to manually test whether the chatbot fulfils the guidelines.

Literature reviews [27, 31] have also identified chatbot quality properties and ways to assess them. Radziwill and Benton align

[1] https://telegram.org/
[2] https://en.wikipedia.org/wiki/Amazon_Echo

bot quality attributes with the ISO-9241 notion of usability [1] (efficiency, effectiveness and satisfaction), while Peras [27] adds further categories (e.g., information retrieval, affect). Generally, the assessment of these quality properties relies on the dynamic execution of the chatbot, on collecting statistical data, or on subjective evaluations [10, 16, 22, 33]. Instead, our goal is to provide metrics that can be calculated automatically and statically on chatbot designs.

Other approaches assess quality via testing [7]. For instance, tools like Botium [3] or OggyBug [13] support test automation. Still, the developer has to provide a set of user utterances and expected chatbot answers within the envisioned conversation flows. To alleviate this burden, some works focus on the generation of challenging test user utterances [4, 5], e.g., via mutation of the training phrases defined for the intents. ChatEval [35] targets testing readability, which can be done statically by applying metrics (e.g., BLEU2 and average cosine similarity [19]) to the chatbot responses, and interactively by requiring the user to complete evaluation tasks. Instead, our goal is to provide complementary assessment mechanisms to testing in the form of metrics, which can be applied prior to deploying the chatbot and can reveal defects in the chatbot design.

As we will see in Section 4.2, some of our metrics profit from the work of the NLP community, which has developed useful readability metrics [19, 30]. For example, Pitler and Nenkova [30] combine lexical, syntactic and discourse features in a highly predictive model of human judgements of text readability. In this model, several linguistic features correlate best with readability judgments. In particular, the average number of verb phrases per sentence, the number of words in the text, and the vocabulary, among others, are associated with human assessments of how well a text is written. More specific to chatbots, Liu et al. [19] identify some weaknesses of metrics for chatbot responses, and provide recommendations for future chatbot evaluation systems.

Overall, we observe a lack of metrics to evaluate statically and automatically quality aspects of chatbot designs, independently of the chatbot implementation platform. Our goal is to fill this gap.

4 CHATBOT DESIGN METRICS

In order to provide a suite of metrics independent from the chatbot implementation technology, we propose using a neutral design notation to represent chatbots, over which the metrics can be computed. In this section, firstly, Section 4.1 introduces the chatbot design notation, and then, Section 4.2 details our proposed metrics.

4.1 A neutral notation for chatbot designs

Since our aim is to develop metrics for chatbot designs, we need a concrete notation over which to define the metrics. For this purpose, we rely on the chatbot neutral notation we proposed in [28], called Conga. We opt for this neutral notation because, as reported in [28], its definition is based on a thorough revision of 15 widely used chatbot development platforms. This means that the design concepts in Conga can be mapped from and to all these platforms. Hence, by defining the metrics over Conga, they become platform-agnostic as well as significant for many chatbot development platforms. As we will show in Section 5, another practical implication is that one can build importers from different platforms into Conga to perform the measurement of existing chatbots.

Figure 2 depicts the meta-model of Conga. It permits representing a chatbot by a Chatbot object, which contains a set of Intents, Entities, Actions performed by the bot, and conversation Flows. The notation supports multi-language chatbots, and so, each intent can declare a number of TrainingPhrases per definition language. The phrases may refer to Parameters, which are defined at the level of the intent. Parameters are typed either by predefined entities (enumeration PredefinedEntity) or user-defined Entity objects. Entities can be Simple, Regex (regular expressions, a change in this new version of the meta-model) or Composite, and for each language (EntityLanguage), they declare the set of literals and synonyms making up the entity. For example, a chatbot can declare a simple entity for drink sizes with literals small, medium and large in English, and additionally define synonyms regular for medium and big for large.

A chatbot can define one or more Actions of type Text, Image, HttpRequest, HttpResponse and Empty. The two first types are used to compose responses combining text and images. HttpRequest and HttpResponse allow configuring the communication of the chatbot with external services in the backend. The last action type Empty is a wildcard for other platform-specific actions, added in this new version of the meta-model to facilitate transformation between platform-specific definition into Conga (explained in Section 5.2).

Finally, the conversation flow between the chatbot and the users is modelled by Flow objects consisting of user and bot turns (classes BotInteraction and UserInteraction). The user turn refers to the intent to be recognized in the interaction (reference UserInteraction.intent). The bot turn specifies the actions that the bot has to perform (reference BotInteraction.actions).

4.2 A metrics suite for chatbot designs

We propose the suite of metrics for chatbot designs that Table 1 shows. All metrics measure internal attributes of chatbots. We considered three sources when designing the metrics:

- Some of them, like INT (the number of intents) or ENT (the number of user-defined entities), are calculated by taking statistics of concepts from the meta-model in Figure 2. According to [28], these concepts are common in chatbot development frameworks.
- Some other metrics have been adapted from the NLP literature [19, 30] to assess the readability of the chatbot responses or the complexity of the expected user utterances.
- Finally, we use the conversation design principles proposed in [23, 34], and Moore and Arar's classification of chatbots [23], to interpret the value of some metrics such as PATH (the number of conversation paths), FLOW (the number of conversation entry points) and WPOP (the number of words per bot output phrase).

The fourth column of Table 1 classifies the potential impact of the metrics on usability (as defined in the ISO 9241-11) [1] in terms of Effectiveness (i.e., accuracy and completeness with which users achieve their goals), efficiency (i.e., time and resources that users expend to achieve their goals) and Satisfaction (i.e., comfort and acceptability of use). We also classify metrics based on their target: either global design properties, or specific aspects of intents, entities or conversation flows. Non-global metrics can be computed per element (intent, entity, flow) or averaged for all elements of a kind.

4.2.1 Global metrics. We start introducing *global metrics*. These measure the number of intents (INT), entities (ENT) and flows (FLOW,

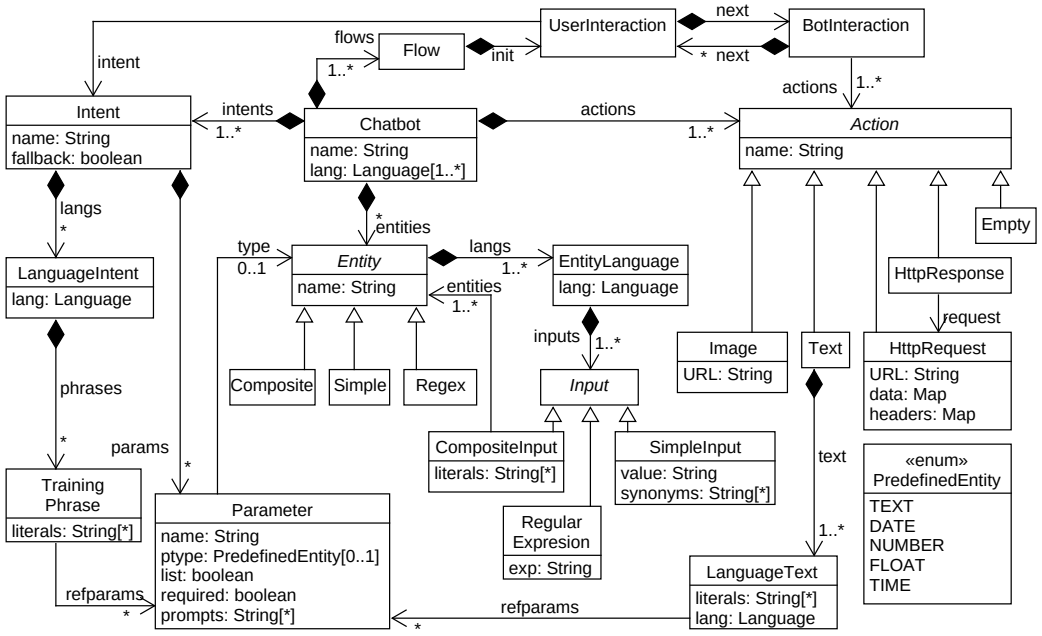

Figure 2: Meta-model for chatbot design (adapted from [28]).

Table 1: Metrics for chatbot designs. Column dimension uses abbreviations for Effectiveness, efficiencY and Satisfaction.

Metric	Description	Type	Dim
Global metrics			
INT	# intents	design size	E
ENT	# user-defined entities	vocabulary size	S
FLOW	# conversation entry points	conversation diversity	E
PATH	# different conversation flow paths	conversation complexity	S,Y
CNF	# confusing phrases [8]	bot understanding	E,S
SNT	# positive, neutral, negative output phrases [33]	user experience	S
Intent metrics			
TPI	# training phrases per intent	topic complexity	E,S
WPTP	# words per training phrase	topic complexity	Y
VPTP	# verbs per training phrase	topic complexity	S,Y
PPTP	# parameters per training phrase	topic complexity	E
WPOP	# words per output phrase	readability	S,Y
VPOP	# verbs per output phrase	readability	S
CPOP	# characters per output phrase	readability	S,Y
READ	reading time of the output phrases [6]	readability	Y
Entity metrics			
LPE	# literals per entity	vocabulary complexity	S
SPL	# synonyms per literal	vocabulary complexity	S
WL	word length	readability	Y,S
Flow metrics			
FACT	# actions per flow	bot response complexity	E,S
FPATH	# conversation flow paths	conversation complexity	S,Y
CL	conversation length	conversation complexity	Y

PATH), and include understanding and user experience metrics (CNF, SNT).

INT is an indicator of design size and functionality, since each intent contributes to functionality offered to the user. The larger INT is, the more functionality the bot offers, potentially impacting effectiveness. ENT measures the size of the chatbot vocabulary and the conversation topic diversity, which may affect satisfaction.

FLOW counts the number of conversation entry points for users, being an indicator of conversation diversity. Since each entry point might correspond to a functionality, FLOW may impact effectiveness. PATH measures conversation complexity. If PATH=FLOW, all conversations are linear, while if PATH>FLOW, some conversation splits into several paths. As an example, Figure 3 shows two small excerpts of chatbot designs conformant to the Conga meta-model. The chatbot design (a) depicts a linear flow (i.e., FLOW=PATH=1). Linear flows enable simple conversations, typically request/reply, which may indicate a system-centric chatbot [23]. The chatbot design (b) shows a conversation flow that splits after the bot interaction (FLOW=1, PATH=2). This kind of flows permits non-linear conversations with multiple turns and dialogue alternatives, typical of conversation-centric chatbots [23].

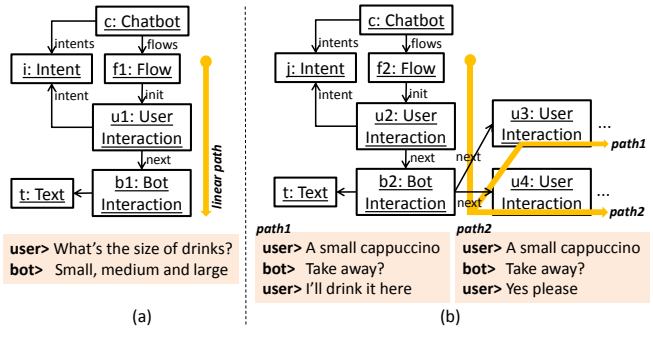

Figure 3: Chatbot design excerpts illustrating (a) a linear conversation flow, (b) a forked conversation flow.

The combined use of FLOW and PATH can help detecting deviations of some design principles. The *recipient* principle [34] advices to design for the target users, from experts (who may give all information at once) to novices (where the bot needs to prompt for more information). In turn, the *repair* principle [34] recommends

supporting clarifications in the conversation, and multiple paths may be an indication of this. Moreover, having several paths per flow potentially results in more natural conversations (impacting satisfaction) but less predictable for the user (likely impacting the user effort or efficiency).

The CNF metric measures the semantic distance between the training phrases of different intents, identifying similar phrases that may confuse the bot to make it identify a wrong intent [8]. Since this may cause errors, the metric is related to effectiveness and satisfaction.

Finally, SNT measures the sentiment of the chatbot output phrases, classifying them into positive, negative and neutral. This is related to satisfaction, since a bot that outputs mostly negative phrases may cause a negative user experience [33].

4.2.2 Intent metrics. Intent metrics measure quality properties of each intent with respect to the expected user utterances and the bot output phrases.

Related to user utterances, TPI counts the number of training phrases in the intent definition. The larger TPI is, the more precise the intent recognition might be, but this also may indicate a complex intent. WPTP measures the length of the training phrases in words. Long phrases are not adequate or even possible in social networks (e.g., Twitter restricts message length), and so, large WPTP values might be problematic. VPTP measures the number of verbs per training phrase. This is an indication of interaction complexity, since composite phrases with several verbs can be more difficult to elaborate for the user [30]. PPTP measures the number of information items (i.e., parameters) the user needs to provide, and the larger PPTP is, the more complex is the intent domain concept.

Regarding chatbot outputs, WPOP measures the number of words per bot output phrase. According to the *minimization* principle [23, 34], the bot answers should be concise. Large phrases are more difficult to understand and can be problematic in social networks. The latter is more concretely targeted by CPOP, as high values may require scrolling (e.g., in mobile devices) and long reading times (with the risk that the user does not complete the reading [23]). Long outputs are especially problematic for voice-based chatbots, since speaking takes longer than reading [23]. Hence, large CPOP values may decrease user satisfaction and efficiency. Similarly, VPOP is another indicator of the complexity of the chatbot responses, given by the number of verbs per output phrase. Finally, READ measures the expected reading time of the bot output responses (a metric related to efficiency). This is calculated as the ratio between the number of words per output phrase, and the number of words that an average person can read per minute [6].

4.2.3 Entity metrics. Entity metrics target user-defined entities representing domain concepts. LPE and SPL are indicators of the complexity of the concepts managed by a chatbot, impacting satisfaction. High LPE and SPL values signal elaborate concepts, but since SPL counts synonyms, a large number may improve recognition in user utterances (better satisfaction). A narrow vocabulary (low SPL) may constrain the way users communicate with the chatbot, and may lead to frustration if the chatbot does not recognize important parameters within user utterances. WL measures the length of words, and similar to CPOP, it contributes to readability and may impact user satisfaction and efficiency.

4.2.4 Flow metrics. Flow metrics consider features of the conversation flows. FACT measures the bot actions (presenting images, text, calling backends) in each conversation flow. The more actions, the more sophisticated tasks can be achieved. Moreover, rich controls help to reduce the user cognitive load and speed up the completion of the intended task. Hence, FACT may impact effectiveness and satisfaction. FPATH measures the number of possible paths per conversation flow. High values signal complex conversations (i.e., more natural-sounding but less predictable). The PATH global metric is calculated by adding up FPATH for each flow. Finally, CL measures the length of each path within a flow, as the number of bot and user turns. This is an indicator of conversation complexity. Longer paths require more time to complete – which affects efficiency – and are typical of conversation-centric chatbots [23].

5 ARCHITECTURE AND TOOL SUPPORT

We have developed a tool called Asymob supporting the automatic measurement of chatbot designs specified with Conga. Next, Section 5.1 presents the architecture of Asymob, including its main features, underlying technologies, and steps required to compute the metrics. Then, Section 5.2 details the conversion of chatbots implemented in two mainstream platforms into Conga.

5.1 Overview of Asymob

We have built a Java framework called Asymob for measuring chatbot designs. The framework is available at https://github.com/ASYM0B/tool. Asymob has a modular architecture that facilitates adding new metrics in multiple programming languages, like Java, Python and Perl. To support chatbots from different platforms, it relies on the neutral chatbot design notation Conga, introduced in Section 4.1. Hence, Asymob computes the metrics on Conga models, independently of any chatbot implementation platform. To measure chatbots from a specific platform, an importer from the platform into Conga must be provided. Currently, Asymob has importers from Dialogflow and Rasa. Section 5.2 will provide more details about these two importers.

To simplify the implementation of new metrics, Asymob supports third-party technologies such as Stanford CoreNLP [20], TensorFlow [2] and Deep Java Learning [11]. Stanford CoreNLP is an NLP library that Asymob uses to perform sentiment and syntactic analysis of the chatbot training and output phrases. The implementation of metrics SNT and VPTP make use of this library. Asymob relies on Deep Java Learning and TensorFlow to detect confusing phrases between intents, using the cosine similarity algorithm. The CNF metric is based on this algorithm.

Figure 4 shows the architecture of Asymob, and the steps to measure a chatbot design. First, the user selects a set of metrics (label 1) and a chatbot (label 2). Then, the Asymob core configures the *metrics database* with the selected metrics, and converts the provided chatbot into a Conga model (label 3, see Section 5.2). Next, the *metric engine* applies the selected metrics to the Conga model, and stores the results in a meta-data file (label 4). On request (label 5), the user can obtain a report with the results in several formats like plain text, Excel and LaTeX (label 6).

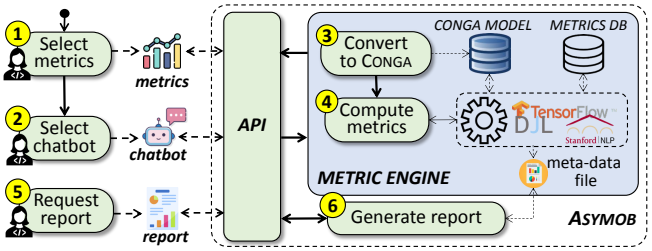

Figure 4: Architecture of Asymob.

5.2 Importing chatbots into Conga

In the following, we provide details of the importers that we have built to convert chatbots from two representative and widely used chatbot platforms (Dialogflow and Rasa) into Conga.

5.2.1 From Dialogflow to Conga. Dialogflow is a low-code development platform to create chatbots using a graphical interface within the browser. Chatbots so defined can be exported as JSON files, which our importer is able to convert into Conga models.

In the JSON-based representation of a Dialogflow chatbot, the file *Agent.json* describes global chatbot features, like its name, definition languages, or connection data to external services (the *webhook*). The latter include details such as the URL, headers, and authentication credentials. Our importer creates a Conga Chatbot object using the agent name and languages, and an HttpRequest action with the webhook data.

Entities in Dialogflow can be predefined or user-defined. The latter are described either by a regular expression, a list of literals with synonyms, or a composite entity. Each user-defined entity becomes exported as a JSON file containing the entity name and configuration information (if it is a regular expression or a composite entity), and one file per definition language with the corresponding literals. Our importer converts these files into Conga Entity objects.

Intents in Dialogflow have a name, training phrases, responses, parameters, and an indication of whether they are fallback or enable a webhook, among other features. Intents are exported into JSON files. For each intent definition file, our importer creates a Conga Intent object with its Parameters and TrainingPhrases, as well as the necessary Actions to compose each response. We currently support text and image responses, and convert other custom responses into Empty actions. Anyhow, this does not affect the defined metrics.

Finally, Dialogflow controls the conversation flow via contexts. These can be input/output to intents, and can store relevant conversation state. Our importer uses the contexts and the responses of the related intents to generate Conga Flow objects.

5.2.2 From Rasa to Conga. Rasa is a framework to develop chatbots using Python, markdown and YAML. The definition of a Rasa chatbot comprises several files. The *config.yml* file defines configuration properties, like the chatbot language or the used NL prediction model. The *data/nlu.md* file contains training data to identify the intents correctly, with its entities and synonyms or regular expressions. As an example, the *data/nlu.md* file in Listing 1 defines an intent called order (lines 1–4). The parameters in the training phrases can be defined within brackets and followed by the entity name in parenthesis (e.g., [cappuccino](type)), or with curly brackets (e.g., [medium]{"entity": "size", "value": "medium"}).

```
1  ## intent:order
2  - I'd like a [medium]{"entity": "size", "value": "medium"} [cappuccino](type)
3  - I want a [small]{"entity": "size", "value": "small"} [latte](type)
4  - Can I order a [large]{"entity": "size", "value": "large"} [black](type) coffee?
5  ## synonym:small
6  - little
7  - short
8  ## synonym:medium
9  - regular
10 - median
11 ## synonym:large
12 - big
13 - extra
```

Listing 1: Example of data/nlu.md Rasa file

```
1  ## story1
2  * order
3      - utter_confirm_order
```

Listing 2: Example flow from data/stories.md Rasa file

The listing also declares synonyms for literals small (lines 5–7), medium (8–10) and large (11–13).

The file *domain.yml* defines the chatbot intents, entities and actions. Actions can be text, images, buttons, or custom actions defined in the Python file *actions.py*. Finally, the file *data/stories.md* specifies the conversation flows. Listing 2 shows a flow example, by which matching the intent order triggers the response utter_confirm_order.

We have built an importer that reads the chatbot language from the *config.yml* file and creates Conga intents and entities from the *data/nlu.md* file, Conga actions from the *domain.yml* file, and Conga flows from the *data/stories.md* file. As in the case of Dialogflow, our importer from Rasa supports text and image responses, and converts Rasa custom actions into Conga empty actions.

6 EVALUATION

We have used Asymob to perform an empirical study to assess the suitability of our metrics to detect quality issues and compare bots. We aim at answering the following research questions (RQs):

RQ1 Can the defined metrics detect quality issues in real chatbots?
RQ2 Can the defined metrics be used to compare heterogeneous chatbots?

Next, Section 6.1 describes the experiment setting, Sections 6.2 and 6.3 answer the RQs, and Section 6.4 discusses threats to validity.

6.1 Experiment setting

We have analysed 6 Dialogflow chatbots and 6 Rasa chatbots built by third parties, available at https://github.com/ASYM0B/evaluation. Table 2 shows the metric results, with some extreme values marked in bold. Chatbots are categorised depending on their implementation platform (**D**ialog**f**low or **R**as**a**) and their source (**G**ithub or **P**redefined natively on the platform). We used Asymob to import the chatbots into the Conga format (cf. Section 5.2) and obtain the metrics.

6.2 RQ1: Detection of quality issues

Some metric values reveal design issues. The CPOP of the FAQ-RASA-NLU chatbot is 285 characters. This indicates poor accessibility and

Table 2: Summary of the evaluation. Columns use abbreviations for Dialogflow (DF), Rasa (RS), Github (G) and Predefined (P).

Chatbot			Global metrics						Intent metrics								Entity metrics			Flow metrics		
Name	Plat.	Src	INT	ENT	FLOW	PATH	CNF	SNT (%)	TPI	WPTP	VPTP	PPTP	WPOP	VPOP	CPOP	READ	LPE	SPL	WL	FACT	FPATH	CL
bikeShop	DF	G	5	1	4	4	3	38 50 12	2.60	3.48	0.81	0.60	14.00	2.60	55.20	12.00	2.00	5.50	6.64	2.00	1.00	2
googleChallenge	DF	G	32	34	32	32	**1442**	19 59 22	6.94	9.62	1.76	0.94	19.81	3.49	105.16	16.00	3.15	4.54	11.37	1.00	1.00	1
mysteryAnimal	DF	G	62	37	62	62	770	0 0 0	6.52	4.34	1.30	2.27	0.00	0.00	0.00	0.00	163.84	3.89	9.21	3.00	1.00	1
Car	DF	P	77	14	61	117	**4188**	0 0 0	9.70	6.80	1.29	2.25	0.00	0.00	0.00	0.00	14.93	3.60	11.41	1.00	1.92	2
Dining-Out	DF	P	9	15	4	14	148	20 61 19	94.67	3.81	0.76	8.33	8.50	2.56	31.44	7.00	**1255.00**	2.39	11.36	1.25	3.50	3
Easter-Eggs	DF	P	6	0	6	6	0	10 51 **39**	7.17	6.23	1.31	0.00	7.46	1.63	37.54	6.00	0.00	0.00	0.00	1.00	1.00	1
05_event_bot	RS	G	17	0	1	20	167	40 60 0	3.82	2.05	0.12	0.00	15.00	2.75	87.00	12.00	0.00	0.00	0.00	1.02	**20.00**	6
FAQ-RASA-NLU	RS	G	8	0	7	7	4	15 34 **51**	3.38	4.67	1.00	0.00	54.56	3.56	**285.56**	46.00	0.00	0.00	0.00	1.00	1.00	1
small-talk-rasa-stack	RS	G	87	0	86	92	**8467**	24 62 14	22.51	3.82	1.13	0.00	6.66	1.81	28.48	5.00	0.00	0.00	0.00	1.00	1.07	**16**
concertbot	RS	P	6	2	1	1	0	25 75 0	0.00	0.00	0.00	0.00	2.25	0.25	12.25	1.00	0.00	0.00	0.00	3.50	1.00	2
formbot	RS	P	8	1	2	9	37	17 83 0	35.63	4.25	0.92	1.25	5.00	1.50	22.50	4.00	0.00	0.00	0.00	1.16	4.50	7
moodbot	RS	P	6	0	2	4	22	20 80 0	10.50	2.10	0.44	0.00	3.00	1.00	11.25	2.00	0.00	0.00	0.00	1.10	2.00	3

readability, as large answers require scrolling in mobile devices, long reading times (46 seconds for this bot), and cannot be fully displayed on social networks like Twitter due to their message length constraints. As an example, Fig. 5 shows a chatbot response deployed on Telegram using a mobile phone, which requires scrolling as the response has more than 30 lines. This is an example of a content-centric chatbot to access a FAQ. However, according to [23], conversation-centric chatbots with short answers and a natural conversation style are usable in more platforms. The googleChallenge chatbot has the same problem to a lesser extent (CPOP is 105, READ is 16).

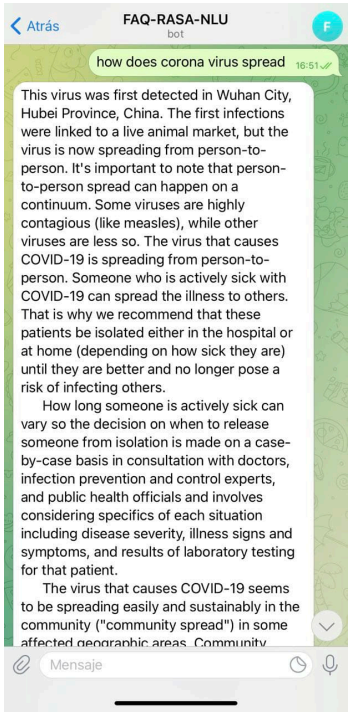

Figure 5: Large response from FAQ-RASA-NLU in a mobile in Telegram.

The flow metrics reveal some complex conversations. The bot 05_event_bot has a single FLOW with 20 paths (PATH and FPATH are 20). Another indicator of conversation complexity is the conversation length CL. The chatbot with the highest CL value (16) is small-talk-rasa-stack.

The sentiment of the bot responses may affect the user experience. In this respect, FAQ-RASA-NLU and Easter-Eggs have 51% and 39% of negative responses (third value of column SNT). Also related to user experience, the high CNF values in bots small-talk-rasa-stack, Car and googleChallenge (8467, 4188 and 1442) may signal chatbot understanding problems due to the existence of similar training phrases in different intents, which may confuse the bots. For example, Car has similar training phrases in different intents, such as *"turn down the heater for each seat in the car"* and *"turn off the heating in my car"*. Other bots with confusing training phrases are small-talk-rasa-stack (*"I am very bored"* / *"I'm bored of you"*), googleChallenge (*"What is the time duration for completing Masters in Artificial Intelligence?"* / *"Completion period for masters in AI?"*), Dining-Out (*"now cafe"* / *"find cafe"*), and bikeShop (*"Can you fix my road bike?"* / *"Can you service my bike?"*). Hence, CNF provides useful information to detect intents that a chatbot may mismatch, without resorting to intensive dynamic testing.

Overall, we can answer RQ1 positively, since our metrics could detect issues regarding readability (CPOP), conversation complexity (FLOW, CL), user experience (SNT) and bot understanding (CNF).

6.3 RQ2: Comparing chatbots

Metrics also serve to compare or classify chatbots based on their design style [23]. For instance, some chatbots like Car, mysteryAnimal and small-talk-rasa-stack are very detailed and complex according to their number of intents (INT), flows (FLOW) and paths (PATH). Instead, others like bikeShop and moodbot are simpler.

Interestingly, two chatbots have no output phrases, one for being a predefined template bot that the developer needs to complete (Car), and the other because a backend API generates the output dynamically (mysteryAnimal). Likewise, chatbots Easter-Eggs, 05_event_bot, small-talk-rasa-stack and FAQ-RASA-NLU lack a domain-specific vocabulary, since ENT is 0. This might be explained as being general-purpose (e.g., for small talk) or simple bots (e.g., 05_event_bot).

Regarding conversations, some chatbots have linear conversations where FLOW=PATH (e.g., FAQ-RASA-NLU, concertbot), while others support complex conversations where FLOW<PATH (e.g., Car, Dining-Out, 05_event_bot). Additionally, the conversation length of some bots is limited to one user-bot interaction (CL=1), and hence, they can be classified as system-centric [23]. Within this set, bots providing long responses (like FAQ-RASA-NLU) are likely content-centric. Other bots allow longer, more elaborate conversations (CL>1). Bots with non-linear conversations (FLOW<PATH) and multiple turns (CL>1) can be classified as conversation-centric [23].

Metrics are also helpful to compare implementation platforms. First, all entity metrics of the analysed Rasa chatbots have value 0. This is so as entities in Rasa are not defined explicitly, but via a Python method that returns whether an entity accepts a given String. The concertbot bot has 0 training phrases because Rasa bots

can be trained interactively. We observe that bots in Rasa define fewer entities (ENT) than in Dialogflow. In general, the analysed Dialogflow bots are more detailed in terms of functionality (INT), vocabulary (ENT) and intent recognition (TPI). Conversations in the Dialogflow bots tend to be linear (PATH=FLOW) while in Rasa they split in several paths (PATH>FLOW), denoting less predictability.

Finally, metrics can be used to compare open-source and predefined bots. In Rasa, the predefined bots are simpler than the Github ones, reflected on lower values of INT, FLOW and PATH. This does not happen with the Dialogflow bots.

Overall, we can answer RQ2 affirmatively. Our metrics permit comparing chatbot complexity and size regarding intents, flows and paths; enable classification of chatbots along Moore and Arar's taxonomy [23]; and – being defined over Conga – they can be applied to different chatbot technologies and chatbot sources.

6.4 Threats to validity

Given the limited size of the experiment, we cannot claim differences or similarities between implementation platforms or predefined/open-source bots, for which we would need a larger scale experiment. Instead, our goal was to hint at the usefulness of the defined static chatbot metrics.

Another limitation of our evaluation is that it relies on custom-made importers from existing platforms into Conga. Since Rasa is a framework, it permits programming some aspects of chatbots in different ways. For example, one may train the model on the fly instead of using training phrases, or even change the conversation flow using Python. All these variants may affect the metric values.

7 CONCLUSIONS AND FUTURE WORK

The increasing relevance of chatbots demands support for assessing their quality prior to testing. With this aim, we have proposed a suite of metrics that can be evaluated statically on chatbot designs, independently of their implementation platform. We have demonstrated the feasibility of our proposal by building the Asymob tool, which we have used to evaluate existing heterogeneous chatbots.

In the future, we plan to extend our evaluation to get a panorama of the features of open-source chatbots and derive metric thresholds. Our metrics could be correlated with development metrics like effort, and validated with usability metrics collected dynamically. We plan to extend our tool to cluster chatbots by similarity, and enable semantic clustering by representing chatbots using a bag-of-words model. The latter can be useful to provide a search facility over chatbot repositories. Technically, we aim at embedding Asymob as a web service to let the community profit from its metrics.

ACKNOWLEDGMENTS

Work funded by the Spanish Ministry of Science (RTI2018-095255-B-I00) and the R&D programme of Madrid (P2018/TCS-4314).

REFERENCES

[1] ISO 9241-11. 1998. Ergonomic requirements for office work with visual display terminals (VDTs). Part II guidance on usability. (1998).
[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Gordon Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*. USENIX Association, 265–283.
[3] Botium. [n. d.]. `https://www.botium.ai/`. ([n. d.]). last access in 2021.
[4] J. Bozic and F. Wotawa. 2019. Testing chatbots using metamorphic relations. In *ICTSS (LNCS)*, Vol. 11812. Springer, 41–55.
[5] S. Bravo-Santos, E. Guerra, and J. de Lara. 2020. Testing chatbots with Charm. In *QUATIC (CCIS)*, Vol. 1266. Springer, 426–438.
[6] M. Brysbaert. 2019. How many words do we read per minute? A review and meta-analysis of reading rate. *J. of Memory and Language* 109 (2019), 104047.
[7] J. Cabot, L. Burgueño, R. Clarisó, G. Daniel, J. Perianez-Pascual, and R. Rodríguez-Echeverría. 2021. Testing challenges for NLP-intensive bots. In *BotSE*. IEEE, 31–34.
[8] D. Cer, Y. Yang, S.-Y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018), 7.
[9] Chatbottest. [n. d.]. https://chatbottest.com/. ([n. d.]). last access in 2021.
[10] D. Coniam. 2014. The linguistic accuracy of chatbots: usability from an ESL perspective. *Text & Talk* 34, 5 (2014), 545–567.
[11] Deep Java Library. [n. d.]. `https://djl.ai/`. ([n. d.]). last access in 2021.
[12] Dialogflow. [n. d.]. https://dialogflow.com/. ([n. d.]). last access in 2021.
[13] M. B. dos Santos, A. P. C. C. Furtado, S. C. Nogueira, and D. D. Moreira. 2020. OggyBug: A test automation tool in chatbots. In *SAST*. ACM, 79–87.
[14] N. E. Fenton and S. Lawrence Pfleeger. 1996. *Software metrics - a practical and rigorous approach (2. ed.)*. International Thomson.
[15] FlowXO. [n. d.]. https://flowxo.com/. ([n. d.]). last access in 2021.
[16] J. Jiang and N. Ahuja. 2020. Response quality in human-chatbot collaborative systems. In *SIGIR*. ACM, 1545–1548.
[17] C. Lebeuf, M.-A. D. Storey, and A. Zagalsky. 2018. Software bots. *IEEE Softw.* 35, 1 (2018), 18–23.
[18] Lex. [n. d.]. https://aws.amazon.com/en/lex/. ([n. d.]). last access in 2021.
[19] C.-W. Liu, R. Lowe, I. Serban, M. Noseworthy, L. Charlin, and J. Pineau. 2016. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP*. ACL, 2122–2132.
[20] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL: System Demonstrations*. 55–60.
[21] Microsoft Bot Framework. [n. d.]. https://dev.botframework.com/. ([n. d.]). last access in 2021.
[22] S. Möller, R. Englert, K.-P. Engelbrecht, V. V. Hafner, A. Jameson, A. Oulasvirta, A. Raake, and N. Reithinger. 2006. Memo: Towards automatic usability evaluation of spoken dialogue services by user error simulations. In *ICSLP*. ISCA.
[23] R. J. Moore and R. Arar. 2018. Conversational UX Design: An Introduction. In *Studies in Conversational UX Design*. Springer, 1–16.
[24] R. J. Moore and R. Arar. 2019. *Conversational UX Design: A Practitioner's Guide to the Natural Conversation Framework*. ACM, New York, NY, USA.
[25] R. J. Moore, E. Young Liu, S. Mishra, and G.-J. Ren. 2020. Design systems for conversational UX. In *CUI*. ACM, 45:1–45:4.
[26] Pandorabots. [n. d.]. https://home.pandorabots.com/. ([n. d.]). last access in 2021.
[27] D. Peras. 2018. Chatbot evaluation metrics: Review paper. In *ESD*. Varazdin Development and Entrepreneurship Agency, 89–97.
[28] S. Pérez-Soler, E. Guerra, and J. de Lara. 2020. Model-driven chatbot development. In *ER (LNCS)*, Vol. 12400. Springer, 207–222.
[29] S. Pérez-Soler, S. Juárez-Puerta, E. Guerra, and J. de Lara. 2021. Choosing a chatbot development tool. *IEEE Software* 38, 4 (2021), 94–103.
[30] E. Pitler and A. Nenkova. 2008. Revisiting readability: A unified framework for predicting text quality. In *EMNLP*. ACL, 186–195.
[31] N. M. Radziwill and M. C. Benton. 2017. Evaluating quality of chatbots and intelligent conversational agents. (2017). http://arxiv.org/abs/1704.04579
[32] Rasa. [n. d.]. https://rasa.com/. ([n. d.]). last access in 2021.
[33] R. Ren, J. W. Castro, S. T. Acuña, and J. de Lara. 2019. Evaluation techniques for chatbot usability: A systematic mapping study. *Int. J. Softw. Eng. Knowl. Eng.* 29, 11&12 (2019), 1673–1702.
[34] E. A. Schegloff. 2007. *Sequence Organization in Interaction*. Cambridge University Press.
[35] J. Sedoc, D. Ippolito, A. Kirubarajan, J. Thirani, L. Ungar, and C. Callison-Burch. 2019. Chateval: A tool for chatbot evaluation. In *NAACL-HLT (Demonstrations)*. ACL, 60–65.
[36] A. Shevat. 2017. *Designing bots: Creating conversational experiences*. O'Reilly.
[37] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella. 1997. PARADISE: A framework for evaluating spoken dialogue agents. In *ACL/EACL*. Morgan Kaufmann Publishers / ACL, 271–280.
[38] Watson. [n. d.]. https://www.ibm.com/cloud/watson-assistant/. ([n. d.]). last access in 2021.
[39] J. Weizenbaum. 1966. ELIZA - A computer program for the study of natural language communication between man and machine. *Commun. ACM* 9, 1 (1966), 36–45.