

Integrating Static Quality Assurance in CI Chatbot Development Workflows

Jesús Sánchez Cuadrado , University of Murcia

Daniel Ávila , Sara Pérez-Soler , Pablo C. Cañizares , Esther Guerra , and Juan de Lara , University Autónoma of Madrid

// To fill a gap in proposals to integrate automated quality assurance mechanisms into the chatbot development workflow, we present a continuous integration workflow for chatbot development, implemented as GitHub actions, and show its usefulness by its application to open source chatbots. //



©SHUTTERSTOCK.COM/PRODUCTION PERIG

CHATBOTS ARE INCREASINGLY present in our lives as a natural language interface to software services. They are popular because they enable a natural human–computer interaction, and can be deployed on multiple channels like websites, social networks usable from mobile devices, or intelligent speakers.

Chatbots are classified as *open-domain* if they can converse on arbitrary topics, as is the case of OpenAI’s ChatGPT (<https://openai.com/chatgpt>) or Google’s Gemini (<https://gemini.google.com/>). Instead, *task-oriented* chatbots target particular tasks, like booking flights or ordering food. They are the default option when enterprises grant conversational access to their services, and so they must be carefully designed, programmed, and tested. However, currently, there are scarce quality assurance (QA) techniques designed to help in the chatbot development process, especially in the early phases, when the chatbot is not fully functional yet.¹ Static analyses can be applied in early development stages, and for general programming languages, they serve to uncover problems and as indicators of expected prerelease defects, helping to inform decisions on code inspection, testing or redesigns.² However, conventional static analysis methods are not suitable for chatbots due to their unique design focused on concepts like intents, training phrases, and conversation flows rather than on traditional programming constructs. Thus, there is a lack of static analyses targeting structural design issues in chatbots, with potential functional impact.

Part of the problem is the plethora of available technologies (e.g., Google’s Dialogflow,³ Amazon Lex,⁴ and Rasa⁵) for developing task-oriented chatbots. This causes technology lock-in since developers can only use the

QA mechanisms that the selected chatbot development platform provides.¹ Moreover, the abundance of technologies makes it difficult to develop QA techniques—especially static ones—applicable to all of them.

To alleviate this problem, we propose chatbot QA techniques executable as part of continuous integration (CI) workflows via a ready-to-use GitHub action. Our proposal is technology-independent since our QA techniques are applicable to several chatbot platforms and versions by the use of an intermediate chatbot representation.⁶ For instance, the same QA workflow can be executed on a chatbot implemented in Rasa 2.0, on its evolution to Rasa 3.0, or on a Dialogflow chatbot. Our workflow supports the extraction of the chatbot design, its graphical visualization, its static analysis (e.g., to detect issues like poorly trained chatbot intents, or defects in the designed conversations), and its measurement (e.g., to compare design aspects, like size or complexity, against thresholds established by the development organization).

This article describes our proposal and evaluates its usefulness for the QA of open source chatbots built with heterogeneous technologies.

Task-Oriented Chatbots

Open-domain chatbots, like ChatGPT or Gemini, rely on *large-language models* (LLMs). These are deep-learning architectures trained on vast amounts of data and able to generate text upon user prompts. Our interest is on chatbots that perform specific tasks like booking a flight on an airline information system. While open-domain chatbots could be fine-tuned for the task, and prompts could be designed to instruct the LLM to complete the task, the technology to achieve reliable, robust,

trustworthy task-oriented chatbots using LLMs is still in the making.⁷

Instead, task-oriented chatbots are designed around the *intents*, or tasks, that the chatbots must address. Prominent technologies for developing these chatbots are Google’s Dialogflow,³ IBM Watson,⁸ Amazon Lex,⁴ or Rasa.⁵ Each intent defines *training phrases* that exemplify how to express the user’s intention (e.g., “I’d like to book a flight to London” if the intent is booking a flight). As Figure 1 depicts, the user interacts with the chatbot through a channel, e.g., a website or a social network like Slack or Telegram. When the user produces an *utterance* (step 1 in the figure), the chatbot matches the most similar intent with a confidence

level (step 2). If the confidence is below a threshold, then a *fallback* intent is selected (if one is available), which informs that the user utterance was not understood (step 3a).

Intents may also declare *parameters*, which are pieces of information that the chatbot needs to extract from the utterances (step 3b). For example, in the flight booking intent, the chatbot needs the origin, destination, and date of the trip. Parameters are typed by *entities*, which can be predefined (e.g., numbers, dates) or domain-specific (e.g., airport codes). Parameters may also be optional or mandatory. Whenever the user does not provide a mandatory parameter, the chatbot will prompt the user for it. After extracting the parameters,

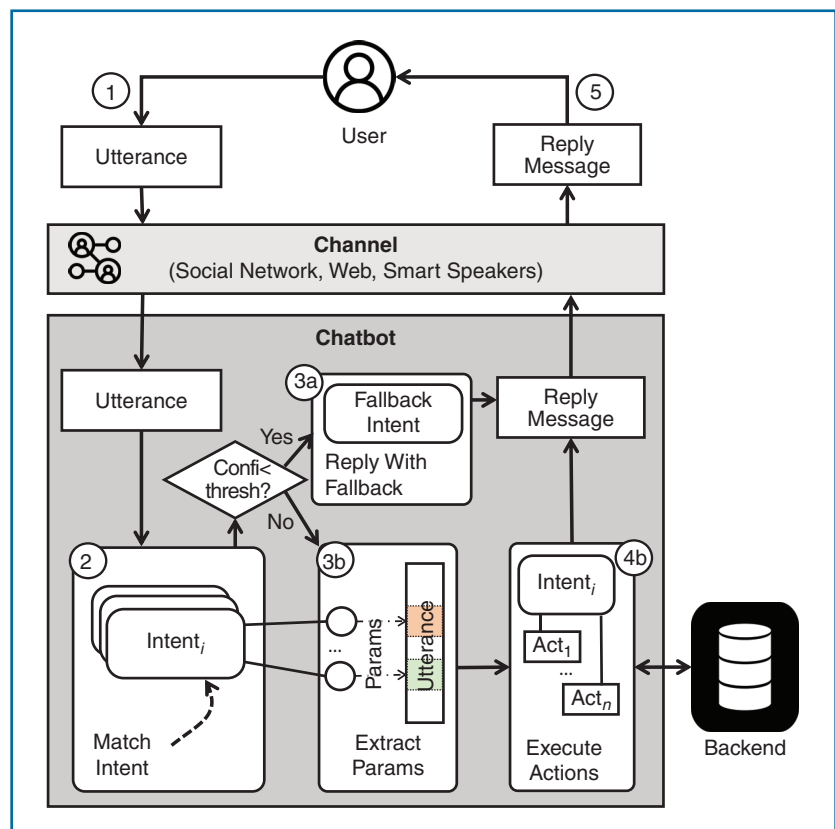


FIGURE 1. Working scheme of intent-based chatbots.

the chatbot will execute the *actions* associated to the intent, like querying an external application programming interface (e.g., to retrieve the available flights) or replying to the user (steps 4b and 5).

GitHub action (<https://github.com/marketplace/actions/rasa-train-test-model-github-action>) facilitates the training and testing of Rasa chatbots on GitHub. Overall, the applicability and usefulness of chatbot testing is well

phrases) effortlessly and without requiring a functional version of the chatbot. Our chatbot design metrics and static analyzer are defined on a neutral chatbot design language, called CONGA.⁶ This encompasses the features of 15 of the most widely used intent-based chatbot construction platforms¹ which enables mapping the design concepts of CONGA from and to these platforms. This way, the metrics and analyses applied to CONGA designs can be considered technology-agnostic, being applicable to many chatbot platforms.

Our proposal is technology-independent since our QA techniques are applicable to several chatbot platforms and versions by the use of an intermediate chatbot representation.

Finally, intents can be organized into conversations, whose entry points are called *flows*. Each conversation flow may then bifurcate into several *paths*, depending on the user responses.

Chatbot QA

Since chatbots are a kind of software, analyzing their quality using dedicated tools is a fundamental task in their development cycle. Three main approaches can be used for this purpose: testing, metrics, and static design validation.

Testing is a widespread technique for checking the correctness of software systems, recently applied to chatbots.^{9,10,11} In industry, Botium (<https://cyara.com/products/botium/>) permits defining test cases (scenarios of expected user-chatbot interactions) and executing them against chatbots. Likewise, Rasa provides a continuous integration/continuous deployment (CI/CD) pipeline (<https://rasa.com/docs/rasa/setting-up-ci-cd>) with mechanisms for testing the chatbot message processing and dialogue management. In particular, the train-test

recognized. However, testing requires having a functional chatbot (precluding its use in early development phases), defining test scenarios of user-chatbot interactions, and it may consume considerable time and resources.

As a complement to testing, researchers have investigated ways to detect potential defects in chatbots earlier, at the design level. For example, *Chatbottest*¹² outlines guidelines for identifying chatbot design issues in categories like answering, error management, intelligence, navigation, personality, and understanding. However, the burden of assessing whether a chatbot meets the guidelines is on the developer.

Also at design level, we propose metrics and static analysis as inexpensive mechanisms for detecting chatbot design issues automatically. The goal is being able to uncover common errors related to user experience (e.g., chatbots issuing mostly messages with negative sentiment), chatbot comprehensibility (e.g., wordy or long conversations) or intent quality (e.g., insufficient training

A CI Workflow for Chatbot Development

To help improve the chatbot development, release, and maintenance process, we have defined the CI workflow depicted in Figure 2. It comprises a GitHub action (see <https://github.com/satori-chatbots/chatbots-actions/>) that triggers automated static quality assurance (SQA) checks whenever a change to a chatbot is pushed into the repository. Our action assumes that the chatbot is in the repository, so for platforms like Dialogflow, the chatbot needs to be exported from the platform and then imported into the repository.

To achieve technology independence, the action first transforms the chatbot into the CONGA language. Then, it produces a graphical visualization of the chatbot design; computes metrics of the chatbot design, comparing their value against predefined thresholds; and performs static analysis of the chatbot design to detect potential problems. Finally, it reports the workflow results to the developer. Compared to the Rasa CI/CD workflow, we focus on static analyses—rather than on testing—and remain technology-independent. Next, we detail the workflow steps.

Transformation into CONGA: In previous work,⁶ we created the CONGA language using model-driven engineering. Its abstract syntax is defined by a metamodel, and its concrete syntax is textual. Its architecture is modular and extensible, enabling the provision of parsers and code generators from/to different chatbot tools. This makes our CI workflow independent of the chatbot technology since adding a parser from a specific technology to CONGA makes the action available for that technology. Currently, Rasa (versions 1.10 to 3.0) and Dialogflow are supported.

Design visualization: To facilitate comprehension, our action produces a graphical visualization of the chatbot design that abstracts away the accidental complexity that specific chatbot technologies may introduce. For example, Rasa chatbots are defined in multiple files and use Python programming, while Dialogflow chatbots are defined via web forms, making it difficult to understand the design underlying the implementation. Instead, the produced visualization represents

the chatbot design as a state machine, where messages in arrows correspond to user utterances, and states correspond to chatbot actions.

Design metrics: In previous work,¹³ we developed a suite of metrics (cf. Table 1) measuring internal quality properties of chatbot designs about their *size* (number of intents, supported languages, flows, paths), *intent quality* (number and complexity of training phrases, similarity of intents), *output phrases* (length, reading time, complexity, readability, sentiment), *vocabulary* (number of entities, complexity of entity literals), and *conversations* (length, paths, actions).

Metrics can help uncovering potential problems, like intents with similar training phrases (so the chatbot may get confused and not recognize the intention behind a user utterance), low number of training phrases, too long conversations (difficult for users to complete), or lengthy chatbot responses (tedious to read, to listen to in case of voice chatbots, or even impossible to deploy in certain channels like X/Twitter).

Our action enables measuring designs and defining thresholds to ensure adherence to internal company guidelines or quality standards, like setting a minimum number of training phrases per intent (e.g., 10 are recommended in Abdellatif et al.¹⁴) or a maximum output length (e.g., when targeting restricted channels).

Design validation: Our static analysis performs checks on the chatbot design to detect issues. These checks complement metrics, detecting well-formedness problems (e.g., duplicate identifiers, several conversation flows starting with the same intent, malformed regular expressions), unused elements (e.g., unused intents), nonterminating conversation loops, repeated training phrases, or lack of a fallback intent, among others. We support both technology-agnostic validations that any chatbot design should fulfill, and technology-specific ones. For example, Rasa does not support multiple fallback intents or multilanguage chatbots, and Dialogflow can issue at most one HTTP request in every conversation turn. The issues are

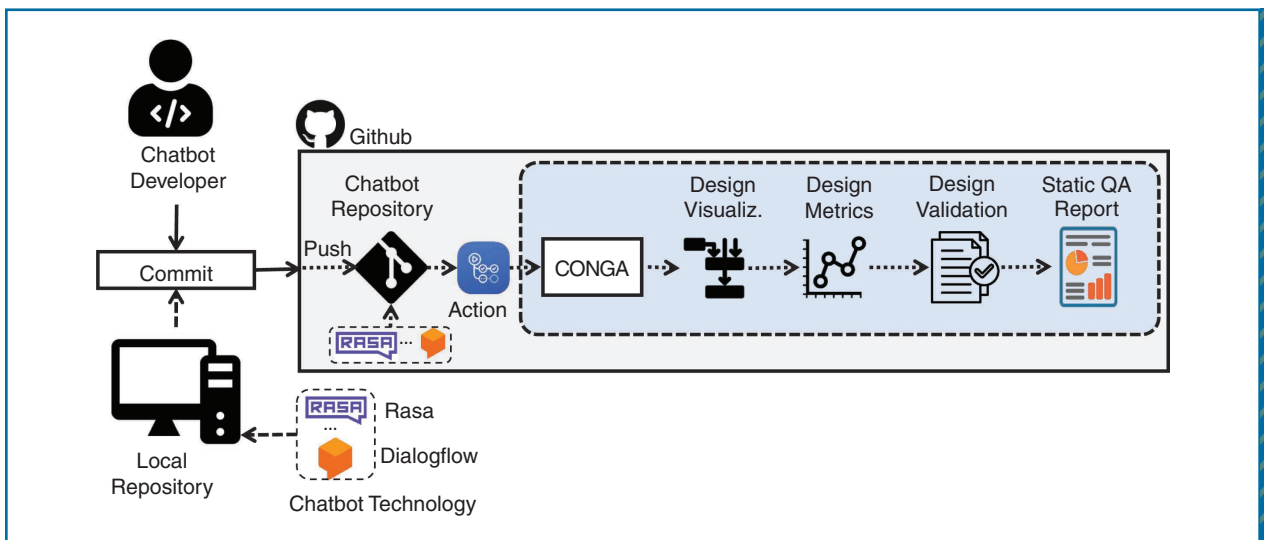


FIGURE 2. Scheme of our CI workflow for chatbots.

classified into errors (i.e., the design is malformed, so the chatbot will not compile or will fail at run-time) and warnings (i.e., quality problems that may make the chatbot not work as expected). We currently support 20 technology-agnostic validations (eight errors and 12 warnings), two

specific validations for Dialogflow, and three for Rasa.

We designed our validations considering several sources. Errors and technology-specific checks stem from an analysis of existing platforms.¹ The remaining ones reflect our experience on developing chatbots, the analysis of open source chatbots,¹³ and guidelines and recommendations from the literature (e.g., few training phrases in intents).¹⁴

Results report: Figure 3 shows a screenshot of the results of the action execution, triggered upon pushing into the repository. The action is lightweight, typically completed in seconds (29 s in the figure) since it does not require executing the chatbot. This allows an inexpensive, early assessment of the chatbot quality even when the chatbot is not fully functional. The results comprise: 1) a diagram (generated with PlantUML) of the chatbot design represented as a state machine; 2) a table with the metric values, and indicators of their compliance to the thresholds established within the project; and 3) a table with the problems found. The metrics to calculate and their thresholds are configurable. In the figure, the developer filtered out some metrics and established thresholds based on her experience and knowledge about the expected chatbot usage. Empirical studies about chatbot usage could be used to identify metric outliers (too low or too high values) and fine tune such thresholds.¹³ In this example, the metrics identify potential issues due to intents with few training phrases and entities with no literals. Moreover, the static analysis detects unused intents (i.e., intents that are not part of any conversation) and intents with a low number of training phrases (<3).

Table 1. Chatbot design metrics.

Metric	Description	Type
Global chatbot size		
INT	Number of intents	Design size
NL	Number of supported languages	Internationalization
FLOW	Number of conversation entry points	Conversation diversity
PATH	Number of conversation paths	Conversation complexity
Intent quality		
TPI	Number of training phrases per intent	Topic complexity
CNF	Number of confusing phrases	Bot understanding
WPTP	Number of words per training phrase	Topic complexity
VPTP	Number of verbs per training phrase	Topic complexity
PPTP	Number of parameters per training phrase	Topic complexity
Chatbot output phrases		
WPO	Number of words per output	Readability
CPO	Number of characters per output	Readability
VPOP	Number of verbs per output phrase	Readability
READ	Reading time of the output phrases	Readability
OPRE	Output phrase readability	Readability
SNT	Number of positive, neutral, negative output phrases	User experience
Chatbot vocabulary		
ENT	Number of user-defined entities	Vocabulary size
LPE	Number of literals per entity	Vocabulary complexity
SPL	Number of synonyms per literal	Vocabulary complexity
WL	Word length	Readability
Conversation		
FACT	Number of actions per flow	Bot response complexity
FPATH	Number of conversation paths per flow	Conversation complexity
CL	Conversation length	Conversation complexity

rdavilao / test

test-satori-validation-action

rdavilao is checking chatbots with SATORI #331

Summary

Jobs

- check-chatbot

Run details

- Usage
- Workflow file

Triggered via push 3 minutes ago

Status: Success

Total duration: 29s

Artifacts: -

test.yaml

```

check-chatbot 21s

```

check-chatbot summary

Chatbot conversation flow

```

graph TD
    Start(( )) --> Describe-Room
    Describe-Room --> Write-Note
    Write-Note --> Read-Notes
    Read-Notes --> Default-Fallback-Intent-Game
    Default-Fallback-Intent-Game --> Default-Welcome-Intent-Game
    Default-Welcome-Intent-Game --> End(( ))

```

This diagram is built thanks to: [PlantUML](#)

Chatbot Metrics

RESULT	METRIC	VALUE	RANGE	MESSAGE
✓	ENT	2	[0, 9]	2 entities is fine!
✓	INT	14	[3, 29]	14 intents is fine!
⚠	NL	1	[1, 10]	1 supported language is OK (but in the lower limit)
✓	FLOW	5	[2, 21]	5 flows are fine!
✓	PATH	5	[2, 27]	5 paths are fine!
✗	LPE	0	[3, 13]	0 literals/entity is too low: at least 3 required
⚠	SPL	0	[0, 7]	0 synonyms/literal is OK (but in the lower limit)
✗	WL	0	[3, 18]	0 chars/literal is too low: at least 3 required
⚠	CL	1	[1, 5]	1 steps/conversation is OK (but in the lower limit)
⚠	FPATH	1	[1, 6]	1 path/flow is OK (but in the lower limit)
✓	FACT	2	[1, 5]	2 actions/flow is fine!
✗	TPI	4	[8, 30]	4 phrases/intent is too low: at least 8 required
✓	WPTP	3	[1, 7]	3 words/phrase is fine!
⚠	PPTP	0	[0, 3]	0 parameters/phrase is OK (but in the lower limit)
✓	CPOP	23	[8, 250]	23 chars/output is fine!

For more information on the interpretation of these metrics, please visit: [asymb](#)

Conga Validation

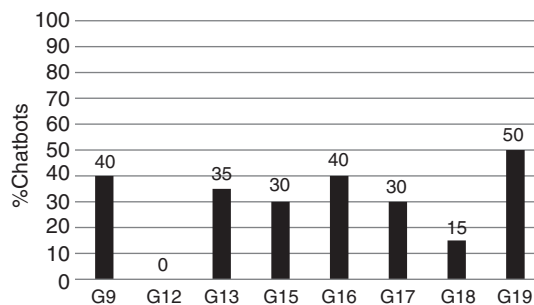
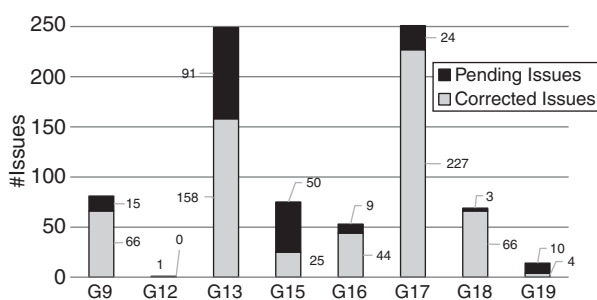
RESULT	ID	PROBLEM
⚠	G13	The intent Take-Note is never used in a flow. Intents should be used in some flow
⚠	G13	The intent Greet-Character-Game is never used in a flow. Intents should be used in some flow
⚠	G13	The intent Move-Room is never used in a flow. Intents should be used in some flow
⚠	G13	The intent Interact-Game-Object is never used in a flow. Intents should be used in some flow
⚠	G13	The intent See_Map is never used in a flow. Intents should be used in some flow
⚠	G13	The intent Info-Character-Game is never used in a flow. Intents should be used in some flow
⚠	G13	The intent help is never used in a flow. Intents should be used in some flow
⚠	G13	The intent Select-Player is never used in a flow. Intents should be used in some flow
⚠	G13	The intent Check-Inventory is never used in a flow. Intents should be used in some flow
⚠	G15	The intent Greet-Character-Game must contain at least tree training phrases in English. The intent must contain at least three training phrases for each language
⚠	G15	The intent Describe-Room must contain at least tree training phrases in English. The intent must contain at least three training phrases for each language
⚠	G15	The intent help must contain at least tree training phrases in English. The intent must contain at least three training phrases for each language
⚠	G15	The intent Select-Player must contain at least tree training phrases in English. The intent must contain at least three training phrases for each language

Summary: Total 13 problems, 0 errors and 13 warnings

Job summary generated at run-time

FIGURE 3. Screenshot of the execution of the SQA action.

Technology	Name	Commits (Total)	Total Issues	Final Issues	Avg Commits to Resolution	Remaining Issue Types
Rasa 2.8	agent-bot-new	36 (73)	27	2	19,8	G16, G19
Rasa 2.6	agents-bot	68 (102)	51	7	17	G9, G13, G18, G19
Rasa 2.0	chatbot	91 (388)	37	10	112,8	G13, G15, G16, G16
Rasa 2.1	chatbot w conv.repa	87 (299)	116	0	50	
Rasa 2.0	chloe	65 (80)	51	12	14	G9, G15, G17, G19
Rasa 3.0	Cinebot	6 (7)	14	13	4,6	G9, G16, G17, G18
Dialogflow	Conf-Chatbot	7 (42)	80	65	22	G13, G15
Rasa 2.8	Cooking Assistant	15 (208)	18	1	21,4	G9
Rasa 2.5	CS310Project	42 (62)	65	0	18,3	
Rasa 2.8	faster-Sharon	50 (65)	56	13	16,3	G15, G17
Rasa 3.0	helpdesk-assistant	13 (198)	12	0	39,8	
Rasa 2.0	IEEE_Chatbot_v2	44 (76)	18	7	39,9	G9, G13, G16, G17
Dialogflow	ISU-Jovo-v2	9 (23)	15	14	11,9	G9, G15, G18
Rasa 3.0	Knowl. based chatbot	25 (65)	13	1	16,5	G19
Rasa 2.0	rasa-cfs	22 (74)	14	7	28,1	G13, G16, G17, G19
Rasa 2.6	rasa-chatbot	40 (87)	38	27	4,2	G9, G13, G17
Rasa 2.0	rasa_project	38 (80)	95	2	22	G16, G19
Rasa 2.0	rhit_IRPA_2023	77 (545)	12	1	59,1	G19
Rasa 2.0	STDs_Bot	37 (60)	32	16	23,8	G9, G13, G16, G19
Rasa 2.6	team saga	31 (144)	31	4	51	G15, G16, G19



Legend

- G9 = The chatbot supports LANGUAGE, but the INTENT does not have an input in this language.
- G12 = ENTITY should be used by some parameter.
- G13 = The INTENT is never used in a flow. Intents should be used in some flow.
- G15 = The INTENT must contain at least three training phrases for each language.
- G16 = Confusing intents: two intents are in the start of a flow and contain equal training phrases.
- G17 = Repeated training phrases for the INTENT. Two training phrases cannot be equal in the same intent.
- G18 = The INTENT contains a training phrase with only a text parameter. Training phrases should contain something more than a text parameter.
- G19 = The chatbot should contain at least one fallback intent.

FIGURE 4. Summary of results of the SQA action on the selected chatbots.

Evaluation

To assess the usefulness of our proposal, we analyzed the commit history of open source chatbots with the goal of answering the research question (RQ): “Could our SQA action have helped detecting potential issues committed during the chatbot development process?”.

The top of Figure 4 shows the analyzed chatbots. We selected them by crawling GitHub to identify relevant repositories containing Rasa or Dialogflow chatbots, as well as a history with at least six commits modifying the chatbot specification (i.e., not only changing the backend). We filtered out non-English chatbots using a language identification service and then kept the chatbots with more commits. Overall, we selected 20 chatbots, 18 built with Rasa, and two with Dialogflow. The latter chatbots are much less common in GitHub since they must be exported from Google’s Dialogflow platform before pushing them into GitHub, e.g., along with their backend.

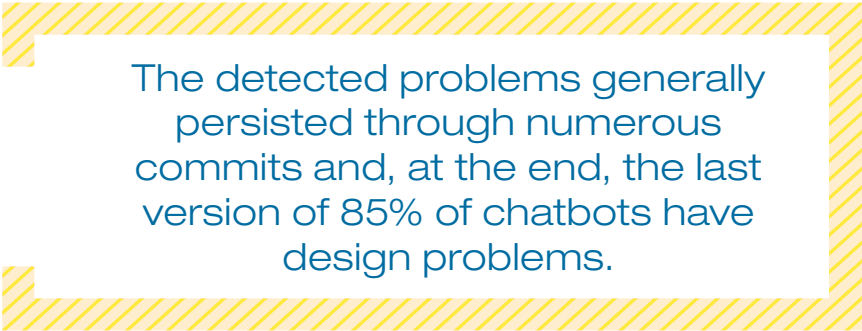
Then, we applied our SQA action to each commit modifying the chatbot. Figure 4 summarizes the results (full data at <https://github.com/satori-chatbots/chatbots-actions-experiments>). The table shows the number of commits (modifying the chatbot and total), different issues throughout the commit history, remaining issues after the last commit, average number of total commits until an issue is resolved, and types of pending issues. The types of identified issues are G9 (intent without training phrases in one of the chatbot languages), G12 (unused entity), G13 (intent not used in any conversation), G15 (intent poorly trained, with less than three training phrases), G16 (two intents starting different conversation flows have

common training phrases), G17 (intent with duplicate training phrases), G18 (improper use of text parameter in training phrase), and G19 (missing fallback intent). After the last commit, only three chatbots were free of defects.

Graph (a) in Figure 4 shows the number of issues corrected across commits, and the remaining ones. From the eight issue types detected, all except G12 are present in the last version of some chatbot, and globally, 25.5% of issues remain. Graph

not affect behavior but gives the false impression of high-quality training. This issue is present in 30% of chatbots. Since training phrases in Rasa are defined in text files, this is prone to copy–paste mishaps.

However, some encountered issues interfere with the proper chatbot operation and become errors which should be fixed before releasing the chatbot. For instance, Cinebot features issue G16 from the second to its last version. Specifically, its intents “book_tickets” and



The detected problems generally persisted through numerous commits and, at the end, the last version of 85% of chatbots have design problems.

(b) shows the percentage of chatbots with a given issue in their last version. Overall, 85% of chatbots have some issue in their final version. Moreover, the average number of commits before an issue is resolved ranges between 4.6 and 112.8. This suggests that the projects could have benefited from our QA action.

Not all detected issues affect the chatbot behavior. For example, Conf-Chatbot has many unused intents (G13), and many of their intents lack at least three training phrases (G15). However since Conf-Chatbot does not use these intents in any conversation flow, their presence does not affect the chatbot behavior. However, it includes unnecessary data in the chatbot definition—akin to “dead code.” Similarly, issue G17 (an intent has duplicate training phrases) does

“collect_tickets” define the same training phrase “tickets please.” Since these intents are entry points for two conversation flows, there is a conflict. Actually, if a user says this phrase to Cinebot, its natural language understanding model favors the “book_tickets” intent (with a confidence of 0.9091, against 0.0890 for intent “collect_tickets”). In practice, this precludes starting the conversation to collect tickets using this phrase. Additionally, both Dialogflow chatbots have few training phrases (G15, metric TPI), and rather short (metric WPTP), which may hinder these chatbots from recognizing the user intention, producing incorrect outcomes. Notably, 50% of chatbots have issue G19 (missing fallback intent). This means that the chatbot will not reply when



JESÚS SÁNCHEZ CUADRADO is an associate professor at the Languages and Systems Department of the University of Murcia, 30100 Murcia, Spain. His research interests include model driven engineering topics, notably model transformation languages, metamodeling, and domain specific languages and recently in the application of AI to software modeling. Caudrado received his Ph.D. in computer science from the University of Murcia. Contact him at <http://sanchezcuadrado.es> or jesusc@um.es.



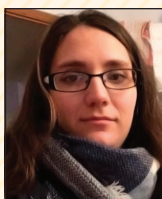
PABLO C. CAÑIZARES is an assistant professor at the University Autónoma of Madrid, 28049 Madrid, Spain. His research interests include software testing, model-driven engineering, distributed systems, and chatbots. Cañizares received his Ph.D. in computer science from the Complutense University of Madrid. Contact him at pablo.cerro@uam.es.



DANIEL ÁVILA is a research project associate with the University Autónoma of Madrid (UAM), 28049 Madrid, Spain. His research interests include the development of computational intelligence, sentiment analysis, machine learning, model-based engineering, and chatbots. Ávila received his master's in research and innovation in computational intelligence and interactive systems from UAM. Contact him at rdavilao@outlook.com.



ESTHER GUERRA is a full professor in the Computer Science Department, University Autónoma of Madrid (UAM), 28049 Madrid, Spain. Her research interests include model-driven engineering, flexible modeling, domain-specific languages, and chatbots. Guerra received her Ph.D. in computer science from UAM. Contact her at esher.guerra@uam.es.



SARA PÉREZ-SOLER is an assistant professor at the University Autónoma of Madrid (UAM), 28049 Madrid, Spain. Her research interests include model-driven engineering, conversational agents, and domain-specific languages. Pérez-Soler received her Ph.D. in computer science from UAM. Contact her at sara.perezS@uam.es.



JUAN DE LARA is a full professor in the Computer Science Department, University Autónoma of Madrid (UAM), 28049 Madrid, Spain. His research interests include automated software development, model-driven engineering, domain-specific languages, and chatbots. De Lara received his Ph.D. in computer science from UAM. Contact him at juan.delara@uam.es.

it does not understand the intent of the user. Finally, 40% of chatbots have intents without training phrases (G9), which makes the chatbots unable to recognize the user intention in those cases. For example, the welcome intent of ISU-Jovo-v2 lacks phrases, so the chatbot does not

recognize when the user starts a conversation by greeting.

Regarding performance, the execution time is in the order of seconds for the whole process (between 30 and 90 s for the analyzed chatbots), which is reasonably fast for the number of analyses performed.

Overall, the detected problems generally persisted through numerous commits and, at the end, the last version of 85% of chatbots have design problems. Even the three chatbots without final issues had a substantial number of them during development, which remained in

many commits. Hence, we can answer the RQ affirmatively: Our SQA action has detected these problems, which is a first step toward their resolution. Actually, to assess the resolvability of the detected problems, we successfully fixed by hand those present in chatbot Cinebot. As a validity threat, these results are specific to the analyzed chatbots and cannot be generalized, i.e., other open source chatbots may have other problems or lack problems. To mitigate any possible bias, we tried to select quality chatbots.

Chatbots should be developed following sound software engineering principles. We claim that CI can help to achieve this aim, as is the case for other types of software. CI automates the integration of code changes by multiple contributors into a common repository, after asserting the correctness of the new code. Our proposed CI workflow comprises a GitHub action covering design visualization, measurement, and static analysis for chatbots. We challenge the community to provide further chatbot quality assessments, which can be integrated as part of CI workflows. In this respect, we are currently working on a technology-independent unit testing action, using Botium as a basis.

Our SQA action is technology-independent. We used it to analyze the history of 20 open source Rasa and Dialogflow chatbots, detecting problems in 85% of them, which suggests the usefulness of these techniques. While we focused on intent-based chatbots, emerging chatbot construction paradigms based on LLMs, like LangChain (<https://www.langchain.com/>), will demand QA mechanisms, likely integrated into CI workflows.

Finally, we foresee the need to migrate intent-based into LLM-based chatbots, and to make both chatbot types interoperable. A technology-independent approach like Conga can help in this respect. ☯

Acknowledgment

This work was supported by the Spanish Ministry with the following grants: TED2021-129381B-C21 and TED2021-129381B-C22 (MICIU/AEI/10.13039/501100011033 and UE/NextGenerationEU), PID2022-140109NB-I00 (MICIU/AEI/10.13039/501100011033 and FEDER/UE), PID2021-122270OB-I00 (MICIU/AEI/10.13039/501100011033 and FEDER/UE), and RED2022-134647-T (MICIU/AEI/10.13039/501100011033).

References

1. S. Pérez-Soler, S. Juárez-Puerta, E. Guerra, and J. de Lara, "Choosing a chatbot development tool," *IEEE Softw.*, vol. 38, no. 4, pp. 94–103, Jul./Aug. 2021, doi: [10.1109/MS.2020.3030198](https://doi.org/10.1109/MS.2020.3030198).
2. N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Proc. 27th Int. Conf. Softw. Eng. (ICSE)*, New York, NY, USA: ACM, 2005, pp. 580–586, doi: [10.1145/1062455.1062558](https://doi.org/10.1145/1062455.1062558).
3. Dialogflow. Accessed: May 2024. [Online]. Available: <https://dialogflow.com/>
4. Lex. Accessed: May 2024. [Online]. Available: <https://aws.amazon.com/en/lex/>
5. Rasa. Accessed: May 2023. [Online]. Available: <https://rasa.com/>
6. S. Pérez-Soler, E. Guerra, and J. de Lara, "Model-driven chatbot development," in *Proc. 39th Int. Conf. Conceptual Modeling*, vol. 12400, Cham, Switzerland: Springer-Verlag, 2020, pp. 207–222, doi: [10.1007/978-3-030-62522-1_15](https://doi.org/10.1007/978-3-030-62522-1_15).
7. J. D. Zamfirescu-Pereira et al., "Herding AI cats: Lessons from designing a chatbot by prompting GPT-3," in *Proc. ACM Designing Interactive Syst. Conf.*, New York, NY, USA: ACM, 2023, pp. 2206–2220, doi: [10.1145/3563657.3596138](https://doi.org/10.1145/3563657.3596138).
8. "Watson." IBM. Accessed: May 2024. [Online]. Available: <https://www.ibm.com/cloud/watson-assistant/>
9. S. Bravo-Santos, E. Guerra, and J. de Lara, "Testing chatbots with CHARM," in *Proc. 13th Int. Conf. Qual. Inf. Commun. Technol.*, vol. 1266, Cham, Switzerland: Springer-Verlag, 2020, pp. 426–438, doi: [10.1007/978-3-030-58793-2_34](https://doi.org/10.1007/978-3-030-58793-2_34).
10. J. Bozic and F. Wotawa, "Testing chatbots using metamorphic relations," in *Proc. 31st IFIP WG 6.1 Int. Conf. Testing Softw. Syst.*, vol. 11812, Cham, Switzerland: Springer-Verlag, 2019, pp. 41–55, doi: [10.1007/978-3-030-31280-0_3](https://doi.org/10.1007/978-3-030-31280-0_3).
11. M. B. dos Santos, A. P. C. C. Furtado, S. C. Nogueira, and D. D. Moreira, "OggyBug: A test automation tool in chatbots," in *Proc. 5th Brazilian Symp. Systematic Autom. Softw. Testing*, New York, NY, USA: ACM, 2020, pp. 79–87, doi: [10.1145/3425174.3425230](https://doi.org/10.1145/3425174.3425230).
12. Chatbottest. Accessed: May 2024. [Online]. Available: <https://chatbottest.com/>
13. P. C. Cañizares, J.-M. López-Morales, S. Pérez-Soler, E. Guerra, and J. de Lara, "Measuring and clustering heterogeneous chatbot designs," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 4, pp. 1–43, 2023, doi: [10.1145/3637228](https://doi.org/10.1145/3637228).
14. A. Abdellatif, K. Badran, D. Costa, and E. Shihab, "A comparison of natural language understanding platforms for chatbots in software engineering," *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 3087–3102, Aug. 2022, doi: [10.1109/TSE.2021.3078384](https://doi.org/10.1109/TSE.2021.3078384).