OggyBug: A Test Automation Tool in Chatbots

Márcio Braga dos Santos
marciobrst@gmail.com
Cesar School
Recife, Brasil

Ana Paula C. Cavalcanti Furtado
Sidney C. Nogueira
anapaula.furtado@ufrpe.br
sidney.nogueira@ufrpe.br
Universidade Federal Rural de Pernambuco
Recife, Brasil

Diogo Dantas Moreira
diogo.moreira@ifpb.edu.br
Instituto Federal da Paraiba
Cajazeiras, Brasil

ABSTRACT

Context: Motivated by the reduction in operating costs, the use of chatbots to automate customer service has been growing. Chatbots have evolved a lot in terms of technologies used as well as in the different application areas. **Problem**: As it is a recent technology, there is no tool offers to support chatbot test automation with the possibility of testing context information that happens in the dialogue between the chatbot and the human; the existing tools also lack facilities to integrate different data sources that are used during the tests. **Objective**: Propose and evaluate a new framework for chatbot testing that considers context information and allows the integration test between different data sources. **Method**: from the analysis of the lack of existing works reported in the literature, a framework for self-testing of chatbots called OggyBug was proposed, which was used by two chatbots development teams that provided feedback on their use. **Results**: Construction of the framework called OggyBug that allows implementing, manage and report the results of the execution of automatic tests for chatbots, either through an API or through a web interface, with ease of integrating different sources of information within the automation scripts. After collecting the feedback from the teams that used the framework, we can observe the ease in defining scenarios and repeating the execution of the tests. **Conclusion**: Testing in context information proved to be important to verify or define the information of the conversation session. The configuration of integration tests proved to be complex, due to the need to configure web services in the chatbot's actions.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*.

KEYWORDS

Chatbot. Testes de Software. Ferramenta de Automação de Testes

SAST '20, October 20–21, 2020, Natal, Brazil

ACM ISBN 978-1-4503-8755-2/20/09…$15.00
https://doi.org/10.1145/3425174.3425230

ACM Reference Format:
Márcio Braga dos Santos, Ana Paula C. Cavalcanti Furtado, Sidney C. Nogueira, and Diogo Dantas Moreira. 2020. OggyBug: A Test Automation Tool in Chatbots. In *5th Brazilian Symposium on Systematic and Automated Software Testing (SAST '20), October 20–21, 2020, Natal, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3425174.3425230

1 INTRODUÇÃO

Constantemente surgem novos produtos no mercado de tecnologia para atender as demandas da sociedade moderna por serviços mais ágeis, automatizados e com o mínimo de interferência humana. Uma dessas inovações são os robôs de bate papo, mais conhecidos como *chatbots*, que são para prover serviços automatizados de atendimento em linguagem natural [3], seu uso é, em parte, motivado pela redução de custos e tempo de atendimento de clientes nas empresas [25]. Suas aplicações são variadas, indo desde sistemas de reservas até catálogos de mercadorias [13], tendo seu principal uso nas áreas de *call center*, agendamento de consultas, automação das tarefas e verificação de equipamentos de IOT [11].

Assim como em qualquer tipo de software, testes são fundamentais para garantir a qualidade e manter a confiança dos usuários diante de atividades complexas realizadas por *chatbots* [7]. Os cenários de testes nesse contexto são descritos por um par de mensagens (de entrada e de saída), ou um fluxo de troca de mensagens que representa um diálogo [5, 7, 8, 28, 32]. A Figura 1 apresenta um exemplo de um diálogo entre um usuário e um *chatbot*. De forma geral, esse fluxo descreve a ordem em que as mensagens devem ser enviadas pelo usuário ou pelo *chatbot*.

Vários aspectos devem ser considerados no teste de *chatbots*. O trabalho de [20] define 10 características para avaliar sua qualidade: aparência; forma de implementação no *website*; unidade de síntese de voz; base de conhecimento (geral e/ou especializada); apresentação do conhecimento e funções adicionais; habilidade de conversa e sensibilidade ao contexto; personalidade; customização da personalidade; e possibilidade de avaliar o *chatbot* no site.

Algumas ferramentas foram descritas para a automação de alguns tipos de testes em chatbots [5, 7, 8, 28, 32]. No entanto, alguns tipos de testes ainda precisam ser definidos, como testes de reconhecimento de padrões, testes de variáveis de contexto e testes de integração.

Regras para reconhecimento de padrões podem ser definidas, e fazer testes ajudar a identificar conflitos com o crescimento das tarefas de um chatbot. Por exemplo, o trabalho [19] fez uso de vários módulos de conhecimento e criou uma estrutura para facilitar

a adição de novos módulos buscando melhorar a experiência do usuário.

Alguns *chatbot* capturam informações durante o diálogo com o usuário e podem determinar ações específicas, alguns exemplos são: identificação de *cyberbullying* [29], rompimento de negócios [1], e engajamento do usuário [12, 19]. Essas informações podem ser injetadas em variáveis de contexto, o que evidencia a necessidade de testar informações de contexto.

No trabalho de [34] é possível perceber a dificuldade de analisar o fluxo de conversação quando várias consultas através de *webservices* foram adicionadas. Isso está associado às integrações que os *chatbots* podem fazem com outros sistemas. Essas integrações de dados com outros sistemas são comuns em *chatbots* de atendimento a usuários, e daí parte a importância desses testes.

Diante desses casos, esse trabalho propõe uma nova *ferramenta*, chamada **OggyBug** que provê suporte para automação de teste de *chatbots* considerando informações de contexto, integração de dados e reconhecimento de padrões. A *ferramenta* permite implementar, gerenciar e relatar o resultado da execução de testes automáticos para *chatbots*, seja através de uma API ou através de uma interface web, com facilidade de integrar diferentes fontes de informações dentro dos *scripts* de automação. A validação da *ferramenta* foi feita com duas equipes de desenvolvimento de *chatbots* em projetos reais para criar cenários de teste e para a execução de forma automática. Após coletar o *feedback* das equipes, pudemos observar a facilidade em definir cenários e em repetir a execução dos testes. Testar as informações de contexto mostrou-se importante para verificar ou definir informações da sessão de conversação.

O restante deste artigo está organizado da seguinte maneira: na seção seguinte são apresentados os conceitos de automação de testes e *chatbots* utilizados neste trabalho; na seção 4 são apresentados os principais trabalhos relacionados a esta proposta; em seguida, na seção 3 é apresentada a metodologia seguida pelo trabalho desde a pesquisa até a validação; as seções 5 e 6, respectivamente, apresentam a *ferramenta* proposta e sua avaliação em dois projetos de *chatbot*; por fim, a seção 7 apresenta conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta a fundamentação teórica do presente trabalho, expondo uma visão geral sobre **automação de testes** e ***chatbots***, com o intuito de compreender a proposta da ferramenta e sua avaliação

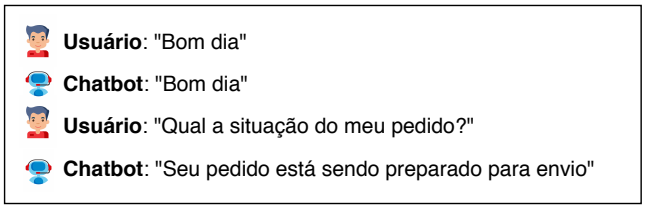

Figure 1: Diálogo entre usuário e um chatbot
Fonte: Elaborado pelos autores (2020)

2.1 Automação de testes

Teste de software é uma das atividades fundamentais no processo de qualidade, tendo como objetivo medir e ajudar a melhorar a confiabilidade de software. A fase de testes é uma das oportunidades para otimizar o custo, a qualidade e o *time to market* de um projeto de software [21]. As atividades de testes devem se iniciar nas fases iniciais do projeto e continuar evoluindo até o fim de seu desenvolvimento [6].

A automação de testes se refere ao uso de ferramentas que controlam a execução de cenários de testes sobre um produto de software [21]. Segundo a ISO/IEC/IEEE 29119-1 [18], cenário de testes é o conjunto de pré-condições, entradas, ações e resultados esperados desenvolvidos para orientar a execução do teste de um item para alcançar objetivos, correções de implementação, identificações de erros, verificações de qualidade e outras informações. Uma ferramenta de automação faz a comparação do resultado gerado pelo produto com o resultado esperado descrito no cenário de teste [14].

Uma arquitetura de automação de testes deve incluir o sistema sob teste, a execução dos testes, ferramentas de monitoração, entradas e saídas, dados armazenados, características dos ambientes, resultado capturado, comparação do resultado, análise da saída e relatórios de resultado. Para [33], uma ferramenta de automação de testes é composta de quatro componentes básicos:

- **Suíte de testes**: conjunto de cenários de testes;
- **Executor**: monitora e controla a execução dos testes;
- **Software sob teste (ou SUT)**: o produto em testes, no caso deste trabalho trata-se de um chatbot; e,
- **Relatório de testes**: informações referentes à execução do cenários de testes, como relatório detalhados ou indicadores.

2.2 Chatbots

Segundo [22], *chatbots* são programas de computador que interagem com seus usuários usando linguagem natural, podendo iniciar, continuar ou tratar interações complexas com parceiros humanos [3]. Um *chatbot* consegue examinar e influenciar o comportamento do usuário fazendo perguntas ou respondendo questões do usuário [2], sendo softwares projetados para criar e reproduzir uma discussão com pessoas usando linguagem natural [25].

Esses sistemas de conversação podem ser categorizados em duas classes [30]: **sistemas de domínio fechado** e **sistemas de domínio aberto**. Os sistemas de domínio fechado focam em responder perguntas de áreas específicas. Enquanto os sistemas de domínio aberto respondem qualquer pergunta. Quanto à tecnologia empregada, os *chatbots* podem ter duas classificações: inteligentes que usam inteligência artificial; e os que usam fluxos pré-definidos de conversação [22]. Os *chatbots* que usam inteligência artificial usam de técnicas de processamento de linguagem natural. Com isso, é possível fazer com que o computador entenda textos como se fossem pessoas [4]. Já os que usam arquivos de regras podem ser criados usando AIML e LSA. AIML é uma linguagem baseada em XML desenvolvida para um computador simular diálogos, enquanto LSA é uma tecnologia usada para definir a semelhança entre palavras e assim sugerir uma resposta [25].

Segundo [9], atualmente existem duas formas de trabalhar com *chatbots*: modelos baseados em retorno e modelos generativos. Os modelos baseados em retorno recuperam uma resposta de um

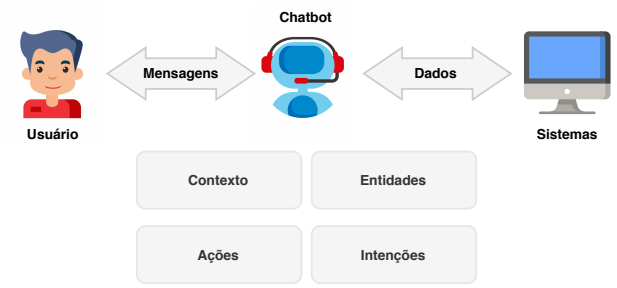

Figure 2: Componentes de chatbots
Fonte: Elaborado pelos autores (2020)

repositório de acordo com um dado contexto. Já os modelos generativos, a resposta que será dada para o usuário será gerada.

Os *chatbots* que tratam de um domínio fechado podem possuir diversos módulos de conhecimento. Esses módulos também podem ser chamados de Habilidades [15] ou Domínios [27]. Um módulo de conhecimento possui alguns componentes: intenções, entidades, variáveis de contexto e ações. A Figura 2 ilustra esses componentes.

Intenções são objetivos ou propósitos expressos na entrada do usuário [15]. Elas são responsáveis por mapear as entradas dos usuários com as respostas [10]. Por exemplo, para um usuário que gostaria de fazer um pedido, poderiam ser definidas duas intenções: "Gostaria de fazer um pedido" ou "Quero comprar um produto".

Entidades são um mecanismo que identifica e extrai dados da entrada do usuário [10]. Elas são informações retiradas da entrada do usuário que auxiliam no propósito da intenção [15]. Por exemplo, na intenção "Gostaria de pedir uma pizza", a palavra "pizza" seria uma entidade que representa o produto que está sendo solicitado pelo usuário.

Ações representam funções executadas para gerar uma resposta [27]. Ações podem ser usadas para fazer integração com outros sistemas. Por exemplo, uma ação pode consultar o preço de um produto ou registrar um pedido em um sistema de uma empresa.

O contexto representa o estado de um diálogo e permite a transmissão de informações de uma intenção para outra [10]. O contexto pode ser atualizado pelas mensagens do usuário ou por ações do *chatbot*. Por exemplo, em uma intenção do usuário "Meu código de acesso é XYZ", o *chatbot* pode executar uma ação de buscar informações para validar a identificação do usuário, e armazenar o resultado em uma variável de contexto.

3 METODOLOGIA

Para [23], uma pesquisa consiste de um problema e as informações necessárias para resolvê-lo. Dessa forma, esta pesquisa objetiva gerar conhecimento à respeito da automação de testes em *chatbots* para auxiliar na melhoria da qualidade e no processo de entrega contínua ligados a seu desenvolvimento. O resumo do protocolo usado nesta pesquisa é apresentado na Tabela 1.

A metodologia utilizada nesse trabalho foi dividida em três fases: revisão bibliográfica; elaboração da proposta; e avaliação, que foi realizada por duas equipes de desenvolvimento.

A fase de revisão da literatura foi realizada usando as principais plataformas de pesquisa nos últimos três anos: **IEEE**, **ACM** e **Google Scholar**. O termo de busca utilizado foi `"chatbot test automation"`. Na busca realizada na base IEEE, esse termo retornou somente 4 resultados, dessa forma, reduzimos o termo para: `"chatbot test"`. Com essa mudança, o número de resultados da busca aumentou em 47. Na ACM e no Google Scholar foram encontrados mais de 23 mil e 2210 artigos, respectivamente.

Os artigos foram selecionados usando como critério a semelhança do título com testes em *chatbots*. Em seguida, foi feita a verificação da introdução e da conclusão dos artigos para selecionar somente os que tratam automação de testes. Após análise desses trabalhos foi possível a definição de uma proposta considerando os tipos de testes que cada trabalho se propunha a tratar.

A avaliação é de natureza qualitativa e de caráter exploratória, tendo sido realizada em duas equipes de desenvolvimento com projetos em plataformas distintas: IBM Watson e Rasa. Os métodos de coleta de dados utilizados foram: entrevista inicial, observação, questionário aplicado ao final do estudo, a análise de artefatos gerados para o sistema e as respostas dadas pelos participantes no questionário final. As etapas da avaliação foram: planejamento, execução, análise dos dados e resultados.

4 TESTE DE CHATBOTS

Após o processo de seleção citado na seção de Metodologia, 5 (cinco) trabalhos relacionados foram selecionados. Uma visão cronológica desses trabalhos é apresentada na Figura 3 e os mesmos estão relacionados abaixo.

(1) **Bottester: Testing Conversational Systems with Simulated Users**, [32] (2017)
(2) **Security Testing for Chatbots**, [8] (2018a)
(3) **BoTest: a Framework to Test the Quality of Conversational Agents Using Divergent Input Examples**, [28] (2018b)
(4) **Botium - The Selenium for Chatbots**, [5] (2018c)
(5) **Chatbot Testing Using AI Planning** [7] (2019)

A primeira referência a automação de testes em *chatbots* aparece em 2017 no trabalho de [32] que apresenta a ferramenta **Bottester** criada com o intuito de testar automaticamente um *chatbot* motivado pelo crescimento do número de cenários. Nesse trabalho, os cenários de testes são compostos de arquivos com as perguntas frequentes e a resposta esperada para cada uma.

Em [8], os autores propõe uma ferramenta para testes de segurança em *chatbots* e descrevem uma proposta de automação para identificar falhas. Definem dois conjuntos de testes para tratar

Table 1: Protocolo de pesquisa.

Natureza da pesquisa	Pesquisa aplicada
Forma de abordagem	Pesquisa qualitativa
Quanto aos objetivos	Pesquisa exploratória
Procedimentos técnicos	Pesquisa bibliográfica

Fonte: Elaborado pelos autores (2020)

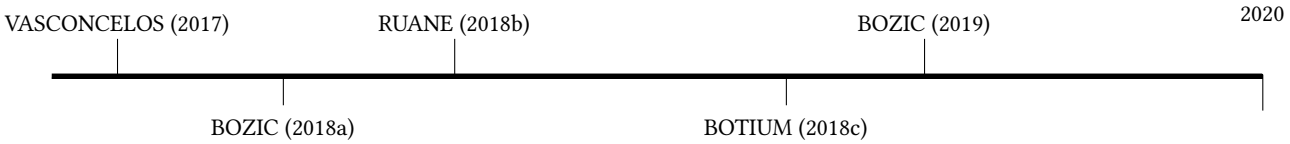

Figure 3: Linha do tempo dos trabalhos relacionados
Fonte: Elaborado pelos autores (2020)

questões de segurança de injeção de SQL e XSS. Neste trabalho, a automatização usa um conjunto de testes, e a avaliação do resultado considera uma interface HTTP disponível do *chatbot*. Testes são feitos com uma base de dados em SQL e uma interface web para interação. Nesse contexto, aspectos relevantes de segurança em *chatbots*, principalmente relacionados à lei geral de proteção de dados, foram considerados.

O trabalho [28] descreve uma ferramenta para a geração de dados divergentes. A partir de um valor de entrada, são gerados valores derivados e submetidos para obter uma resposta. Então, compara-se a resposta obtida com a resposta da entrada inicial. A ideia é avaliar variações sintáticas, semânticas e morfológicas do texto que afetam as respostas dadas pelo *chatbot*.

Botium [5] é uma ferramenta desenvolvida especificamente para testes em *chatbots*. Esta ferramenta consiste atualmente na ferramenta mais completa para automação de testes. Seus principais aspectos são: testes de segurança, testes automatizados, testes de performance, relatórios de testes, entre outros. Permite definir *scripts* de automação que para cenários de testes que simulam o diálogo entre um usuário e o *chatbot*. Além disso, Botium possui uma arquitetura que permite trabalhar com qualquer plataforma disponível, visto que se integra diretamente com a API das plataformas existentes.

Em [7] é proposta uma ferramenta para especificação formal de plano de testes, a geração de cenários e a execução automática.

A tabela 2 apresenta uma comparação entre os trabalhos apresentados. Os seguintes métodos ou técnicas de testes foram usados para a comparação:

- **Testes de Mensagem** (TM): testes simples onde envia-se uma mensagem e espera uma resposta;
- **Testes de Diálogo** (TD): testes com várias mensagens variando de acordo com um contexto;
- **Testes de Entidades** (TE): testes para verificar se os padrões foram reconhecidos adequadamente;
- **Testes de Contexto** (TC): testes que verificam as variáveis de contexto registradas em um diálogo;
- **Testes de Integração** (TI): testes que verificam integrações com outros sistemas;
- **Testes de Carga** (TC): testes que verifica o limite de sessões do chatbot ou a independência do contexto;
- **Testes de Segurança** (TS): testes que verificam aspectos de segurança

Table 2: Comparação dos trabalhos relacionados.

	TM	TD	TE	TC	TI	TC	TS
Vasconcelos (2017)	✓	✗	✗	✗	✗	✓	✗
Bozic (2018)	✓	✗	✗	✗	✗	✗	✓
Ruane (2018)	✓	✗	✗	✗	✗	✗	✗
Botium (2018)	✓	✓	✗	✗	✗	✓	✓
Bozic (2019)	✓	✓	✗	✗	✗	✗	✗

Fonte: Elaborado pelos autores (2020)

5 FERRAMENTA OGGYBUG

As principais partes da ferramenta Oggybug são descritas nas subseções a seguir. A seção 5.1 descreve o módulo para definição de suíte de testes, a seção 5.2 apresenta o gerenciador de execução, a seção 5.3 apresenta o gerador de relatórios do *framework*, e a seção 5.4 apresenta o repositório do código fonte da ferramenta.

5.1 Suíte de Testes

O módulo de suíte de teste do OggyBug é responsável pela documentação dos cenários de testes e pela da definição dos recursos que serão utilizados nos cenários. A ferramenta proposta permite definir cenários para vários tipos de teste, tais como testes de entidades, testes de contexto e testes de integração; além de outros tipos de teste, como testes funcionais, testes de segurança, testes de carga e testes de sessão [5, 7, 8, 28, 32].

Para a documentação dos cenários de testes, uma estrutura em JSON foi elaborada para descrever os dados de execução, dados de integração e as etapas a serem executadas (ver exemplo da Listagem 1). Os dados básicos de um cenário de testes são os campos ***"name"*** e ***"description"***. A plataforma de *chatbot* que será usada na execução é configurada no campo ***"agent"***. A definição de dados de integração é descrita no campo ***"contract"***. O campo ***"steps"*** descrevem as ações que serão executadas nos testes. Essas ações representam pré-condições, entradas ou resultados esperados.

Definimos sete tipos de ações que serão usadas nas etapas de teste, a Tabela 3 classifica e descreve essas ações de acordo com as categorias de testes: diálogo, entidade, contexto, e integração.

5.1.1 Testes de Diálogo. O propósito de um *chatbot* é a troca de mensagens e esta pode ser iniciada pelo usuário ou pelo próprio sistema. Os tipos de etapas `"Enviar mensagem"` e `"Receber mensagem"` são usados para descrever essa troca de mensagens, representando o fluxo básico de uma interação, que podemos chamar de pergunta e resposta.

5.1.2 Testes de Entidades. Quando o usuário interage com um *chatbot*, ou seja, quando envia uma mensagem, esse sistema faz a análise da mesma buscando identificar padrões, também chamados de entidades. Apesar de ser uma tarefa simples, dependendo da quantidade de entidades ou do crescimento das áreas de conhecimento de

um *chatbot*, conflitos podem acontecer. Dessa forma, as entidades devem ser verificadas antes da liberação de uma nova versão do sistema.

5.1.3 Testes de Contexto. Outro ponto importante no desenvolvimento de um *chatbot* são as variáveis de contexto. As variáveis de contexto são informações armazenadas pelo sistema durante uma sessão de conversação com o usuário. Por exemplo, entidades reconhecidas pelo *chatbot*, identificação do usuário, informações do sentimento do usuário, entre outras. Além disso, o contexto pode interferir no propósito de uma intenção, e dessa forma, cria uma variação no fluxo do diálogo.

As variáveis de contexto podem ser provenientes das mensagens trocadas com o usuário, vindas de outros sistemas via integração, ou de ações do próprio *chatbot*. Para tratar esses cenários foram definidas dois tipos de etapas "Atribuir variável de contexto" e "Verificar variável de contexto". A etapa "Atribuir variável de contexto" visa injetar um valor de contexto para ser enviado para o sistema via eventos ou junto com a mensagem de entrada.

5.1.4 Testes de Integração. Os sistemas de *chatbot*, em algumas situações, precisam buscar informações através de integrações com outros sistemas. Essas integrações funcionam como um complemento, principalmente em *chatbots* orientados a tarefas com domínio fechado. As informações provenientes das integrações podem ser armazenadas em variáveis de contexto ou usadas para montar uma resposta do usuário. Já as informações capturadas nas intenções dos usuários podem ser utilizadas para atualizar outros sistemas.

Para atender integrações nos cenários de testes, dois tipos de etapas foram descritos: "Atribuir informação de integração" que popula dados em um recurso REST; e "Verificar informação de integração" que verifica as informações contidas em um recurso REST.

A Listagem 1 apresenta um exemplo de representação de um cenário de testes com várias etapas definidas. Nas linhas 2 a 9 é definido um contrato de recurso para ser usados nas etapas de integração. Nas linhas 11 a 14 é definida uma etapa para popular o recurso de integração com dados iniciais. Nas linhas 15 a 18, uma variável de contexto que define o nome do usuário da sessão é carregado no *chatbot*, e dessa forma, a execução não precisa passar pela tarefa de identificação de usuário. Nas linhas 19 a 22, o executor enviar uma mensagem para o *chatbot* simulando um usuário. Nas linhas 23 a 26, é verificado o reconhecimento da entidade `"PedidoInformado"`. E nas linhas 32 a 34, é verificado se o *chatbot* fez a atualização do status do pedido no recurso de integração definido.

Table 3: Tipos de testes.

Categoria	Tipo
Teste de Diálogo	Enviar mensagem
Teste de Diálogo	Receber mensagem
Teste de Entidade	Verificar entidade
Teste de Contexto	Atribuir variável de contexto
Teste de Contexto	Verificar variável de contexto
Teste de Integração	Atribuir informação de integração
Teste de Integração	Verificar informação de integração

Fonte: Elaborado pelos autores (2020)

Listing 1: Exemplo de cenário

```
1  {
2    "name":"Teste de definição de dados de integração",
3    "contract": {
4      "name": "Pedidos",
5      "fields": [
6        {"name": "code", "type": "text", "key": "true"},
7        {"name": "status", "type": "text", "key": "false"}
8      ]
9    },
10   "steps":[
11     {
12       "type":"Set Data",
13       "data": {"contract":"Pedidos", "code":"PED-10","
             status":"Pendente"}
14     },
15     {
16       "type":"Set Context",
17       "data": { "name":"UsuarioIdentificado", "value":"
             Fulano" }
18     },
19     {
20       "type":"Send Message",
21       "data": {"value":"Gostaria de cancelar meu PED-10"}
22     },
23     {
24       "type":"Check Entity",
25       "data": { "name":"PedidoInformado", "value":"PED
             -10" }
26     },
27     {
28       "type":"Receive Message",
29       "data": {"value":"Seu pedido PED-10 foi cancelado"}
30     },
31     {
32       "type":"Get Data",
33       "data": {"contract":"Pedidos", "code":"PED-10", "
             status":"Cancelado"}
34     },
35   ]
36 }
```

5.2 Gerenciador de Execução

O gerenciador de execução é responsável por preparar o ambiente para execução de um ou mais cenários de testes através de um *drive* específico, e um *token* de autenticação para a configuração do *chatbot*.

Através do gerenciador, é possível definir três fases para o fluxo de execução dos cenários: criação da sessão, execução das etapas do cenário e encerramento da sessão. A Figura 4 ilustra o fluxo de execução do *driver*.

A fase de criação de sessão cria um identificador único da execução. Esse identificador é usado para evitar conflitos durante a execução paralela dos cenários tanto das variáveis de contexto

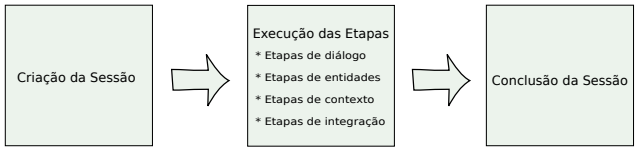

Figure 4: Fluxo de execução do Driver
Fonte: Elaborado pelos autores (2020)

quanto dos dados dos recursos da API REST. A fase de execução realiza as etapas conforme definido no cenário dos casos de testes. A fase de encerramento destrói a sessão no *chatbot* e também registra o resultado da execução para a geração de relatórios.

A implementação da execução dessas fases varia de acordo com a plataforma utilizada. Por exemplo, a criação do identificador da sessão ou o tratamento das variáveis de contexto variam no IBM Watson e no Rasa. Para tratar essa variação foi definido um protocolo, o qual deve ser implementado para cada plataforma na qual se deseja executar os testes.

Outro aspecto importante do módulo executor é a criação dinâmica da API REST para os recursos definidos nos cenários da ferramenta. O módulo cria uma API com o nome do recurso com as operações básicas do REST (GET, POST, PUT, DELETE) e usa a sessão como parâmetro para filtrar os dados correspondentes a uma execução. Essa API deve ser configurada nas ações do *chatbot* com um *token* de autenticação.

5.3 Relatórios de Execução

O módulo de relatórios permite a visualização e exportação dos relatórios de execução dos cenários, e é responsável pela geração de indicadores de execução. Os relatórios de execução mantêm informações sobre as etapas executadas. Para cada etapa, ele exibe o valor esperado, o valor recebido e o resultado da execução da etapa. O relatório pode ser gerado por cenário ou por ciclo de execução. Os indicadores armazenam informações agrupadas por cenários ou por ciclos. Alguns indicadores são: quantidade por etapas, quantidade de etapas por cenário, quantidades de falhas, quantidades de sucesso e duração de execução.

5.4 Instalação e configuração da ferramenta

O código fonte da ferramenta, juntamente com as instruções para configuração e uso podem ser encontrados no repositório do Github[1].

6 AVALIAÇÃO

Conforme explicado anteriormente, na seção 3, a avaliação foi realizada em dois projetos diferentes seguindo as mesmas etapas de avaliação e métodos de coleta de dados. A seguir explicamos detalhes da avaliação.

6.1 Planejamento

A avaliação foi realizada em duas empresas com objetivos e plataformas de *chatbots* diferentes. A primeira empresa, identificada aqui por Empresa X, é um escritório de advocacia cujo projeto de *chatbot* tem o intuito de gerenciar as atividades diárias do escritório, provendo informações sobre as audiências e diligências marcadas. A outra empresa, aqui identificada por Empresa Y, é um escritório de software. O projeto de *chatbot* desta empresa tem por objetivo prover informações sobre eventos de projetos desenvolvidos pela empresa para os gerentes. O Quadro 4 resume o ambiente onde o estudo de caso foi realizado.

Table 4: Ambiente do estudo de caso.

Identificador	Empresa X	Empresa Y
Tipo	Escritório de Advocacia	Escritório de Software
Porte da empresa	Médio	Grande
Objetivo do*chatbot*	Prover informações sobre audiências e diligências	Prover informações sobre projetos da empresa

Elaborado pelos autores (2020)

Table 5: Planejamento inicial da equipe de testes.

Identificador	Empresa X	Empresa Y
Plataforma	IBM WATSON	RASA
Participantes	1	5
Testes	Testes exploratórios usando ferramenta fornecida pela plataforma	Testes automáticos usando TestLink e pyTest

Fonte: Elaborado pelos autores (2020)

Na Empresa X, apenas um programador era responsável pelo desenvolvimento e testes do *chatbot* usando plataforma IBM WATSON. Os testes eram feitos de forma manual usando uma ferramenta fornecida pela plataforma. E na empresa Y, a equipe de testes é composta por 5 pessoas, e a ideia inicial era fazer uso de outras ferramentas para organizar e executar os testes, como TestLink ou pyTest. O Quadro 5 apresenta essas informações de forma sumarizada.

A arquitetura da solução da Empresa X possui os seguintes componentes: IBM Watson Assistant [17], IBM Cloud Functions [16], Nodered [24] e Telegram [31]. Watson Assistant é a plataforma de *chatbot* da IBM. O Nodered faz orquestração das mensagens para integrar o IBM Watson Assistant com o Telegram, e o IBM Assistant consome os serviços definidos no IBM Cloud Functions para gerar respostas para o usuário.

Já o *chatbot* da Empresa Y foi desenvolvido com a plataforma RASA; esse sistema busca informações de projetos da empresa armazenadas em banco de dados.

Em ambas as empresas as seguintes etapas foram realizadas: Levantamento do escopo do projeto, Definição da equipe responsável pelos testes, Apresentação da proposta de automação de testes, Entrega da documentação da ferramenta, Definição dos cenários de testes, Execução e geração dos relatórios, e Aplicação de questionário final.

[1] https://github.com/marciobrst/oggybug.

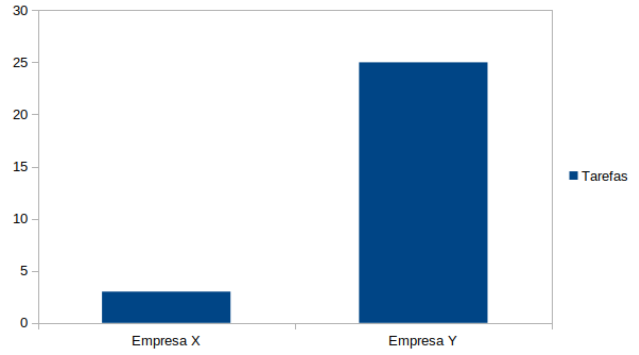

Figure 5: Quantidade de tarefas do *chatbot*
Elaborado pelos autores (2020)

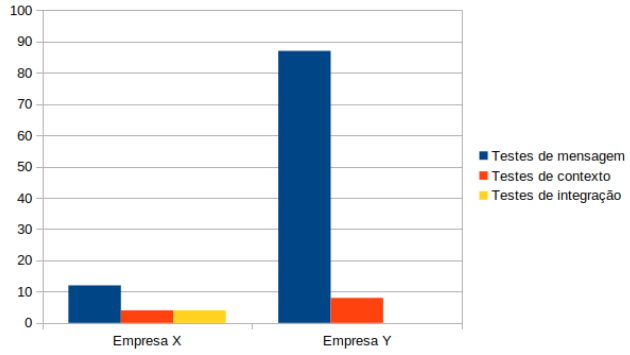

Figure 6: Categorias de etapas definidas nos cenários
Elaborado pelos autores (2020)

6.2 Execução

Inicialmente o escopo dos projetos de *chatbot* e a equipe do projeto foram levantados. Em seguida, a apresentação da proposta foi feita e a documentação de uso da ferramenta foi entregue. Após essas etapas, foram feitas observações durante a definição dos cenários de testes e uso da ferramenta.

Durante a observação, foram encontrados alguns pontos de melhoria da ferramenta para facilitar o trabalho dos participantes: permitir múltiplas respostas esperadas, tratamento de formulários, criação de chaves de acesso para a integração, e uso do módulo de integração contínua.

A possibilidade de uma mesma mensagem de entrada do usuário gerar mais de uma resposta gerou a necessidade de registrar mais de uma resposta esperada na etapa de recebimento de mensagens. Isso é resultado de uma características dos *chatbots* permitirem o uso de variações de expressões (do inglês, *utterance*), por exemplo, "Oi" e "Olá" podem representar variações de uma mesma expressão de cumprimento.

Formulários são um padrão de conversação que coleta fragmentos de informações ordenadas com finalidade de fornecer um tratamento específico [26]. Esse recurso trata o fluxo de coleta de variáveis de contexto de forma específica e gerou a necessidade de atualização no *driver* correspondente do RASA.

Outra melhoria levantada foi em relação aos testes de integração. Foi necessário a criação de um novo módulo para a geração de chaves de acesso usadas na integração, uma vez que os testes estavam acessando o serviços web de integração sem nenhum mecanismo de segurança.

Por fim, foi necessário a criação de um módulo local para fazer uso da API do módulo de integração contínua, que permitiu a execução local dos cenários de testes.

6.3 Análise dos dados

6.3.1 Tarefas por projeto. A Figura 5 apresenta a distribuição de tarefas realizadas pelo *chatbot* de cada empresa.

As tarefas realizadas pelo *chatbot* da Empresa X são: Cumprimentos iniciais, Resumo do dia e Lista de atividades por usuário. Para cada tarefa foram definidos 4 cenários de testes, totalizando 12 cenários de testes. Outros componentes da solução reduziram a quantidade de tarefas do*chatbot*, por exemplo, a identificação do usuário é feita por um módulo fora do*chatbot*.

A Empresa Y definiu 25 tarefas para o*chatbot*, mas não foram criados cenários para todas as tarefas por questões de confidencialidade das informações.

6.3.2 Etapas utilizadas. A Figura 6 apresenta um gráfico com as categorias de etapas usadas nos cenários de testes de ambos os projetos.

O fluxo básico de perguntas e respostas do *chatbot* foi o mais utilizado nos cenários definidos pelas equipes. No entanto, alguns cenários usaram a verificação de variáveis de contexto e integração.

Na Empresa X, foram usados cenários com etapas de contexto de integração. Os cenários de contexto foram usados por conta de variáveis de controle usados pelo *chatbot*, como variáveis de controle de usuário. Já as etapas de integração foram usadas para simular informações dos *webservices* de usuários e atividades da empresa.

Na Empresa Y, nenhum cenário fez uso de testes de integração visto que a ferramenta fazia integração direta com banco de dados e não por serviços web. Entretanto, foram utilizados testes em variáveis de contexto por conta de formulários do RASA.

6.3.3 Execuções registradas. Foram registradas 133 execuções. Os casos de teste da Empresa X foram executados mais de uma vez. Isso foi resultado da fase de exploração da ferramenta pelo participante. Na Empresa Y, a quantidade de execuções registradas é inferior a quantidade de cenários porque os usuário fizeram a exclusão de algumas execuções que possuíam informações confidenciais.

6.3.4 Tempo de execução do projeto. Em virtude das escalas de tempos de execução da Empresa X e da Empresa Y serem diferentes. Optou-se por fazer uma análise dos dados separada.

Os registros de execução da Empresa X tem as seguintes durações: mediana de 136 segundos, duração mínima de 11 segundos e duração máxima de 511 segundos. A maior parte dos valores de duração ficam entre 70 a 231 segundos, representando o 1º e 3º quartil respectivamente.

Na Empresa Y, os registros de execução têm as seguintes durações: mediana de 462 milissegundos, duração mínima de 65 milissegundos e duração máxima de 899 milissegundos. A maior parte

dos valores de duração ficam entre 449 a 471 segundos, representando o 1º e 3º quartil respectivamente.

Os valores de duração das execuções na Empresa Y tem um duração muito inferior porque os cenários foram executados localmente (na máquina do usuário). Na Empresa X, os cenários foram executados acessando um servidor na nuvem. Algumas variações de tempo podem ter sido causadas por *timeouts* durante o acesso a API das plataformas.

6.3.5 Benefícios do uso. A partir das respostas dos participantes, classificamos os benefícios em quatro categorias: velocidade, usabilidade, resultado e credibilidade.

6.3.6 Melhorias propostas. Algumas melhorias foram sugeridas: criar novas formas de visualização, geração de gráficos, criação de mais indicadores para *dashboard*, e adição de mais informações nos relatórios.

6.4 Considerações finais

Com a avaliação foi possível verificar que a ferramenta foi aplicada com sucesso em ambos os projetos. A duração de execução gerou uma velocidade na repetição de testes. E a definição de cenários permitiu a criação e alteração de cenários de forma simples.

6.4.1 Limitações e ameaças a validade. Algumas limitações encontradas durante a avaliação foram:

- O ambiente em que o estudo foi realizado restringiu-se a apenas dois projeto e a duas plataformas, e as conclusões foram baseadas de acordo com as perspectivas dos participantes.
- O escritório de projetos possui informações de caráter confidencial; isso limitou a quantidade de cenários de testes registrados na ferramenta.
- Os testes de integração foram aplicados em somente um dos estudos.
- Uma maior quantidade de ciclos de testes e a aplicação em um processo de entrega contínua pode gerar mais resultados quanto ao uso da ferramenta.

7 CONCLUSÕES

Este trabalho propôs uma *ferramenta* de automação de testes em *chatbots*, que descreve um modelo de documentação de cenários de testes para *chatbots*, incorpora tipos de testes para serem automatizados e descreve uma API para ser usada em entrega contínua e/ou integração contínua.

Os novos tipos de testes propostos foram: testes de reconhecimento de padrões, testes de variáveis de contexto e testes de integração. Os testes de reconhecimento permitem a evolução da base de conhecimento dos *chatbots* e identificam conflitos entre os módulos. Os testes de variáveis de contexto fazem verificações no contexto de diálogos feitos pelo *chatbot*. Os testes de integração tratam da troca de dados com outros sistemas.

A ferramenta facilitou a definição de cenários além da execução de cenários e visualização de relatórios usando uma interface *web*. A ferramenta também implementou a API para a integração contínua e/ou entrega contínua de *chatbots*. Essa função de integração contínua ainda será avaliadas em trabalhos futuros.

A avaliação do trabalho foi realizada com o uso da ferramenta em dois projetos de desenvolvimento de *chatbots*. Na avaliação foi possível a identificação de novas funcionalidades, com a criação de chaves de acesso para a automação de testes. E a identificação de características inerente de cada plataforma, principalmente, em relação ao controle de variáveis de contexto. As equipes desses projetos enalteceram a facilidade de criar cenários, e a produtividade ao repetir testes. As equipe destacaram a necessidade de mais relatórios e melhoria na usabilidade das telas de forma geral.

Devido às limitações impostas pelas as empresas em relação ao acesso a dados, ainda há a necessidade de executar outras experimentações da ferramenta.

Alguns pontos importantes como geração de documentação, versionamento dos cenários e gerenciamento de ciclos de testes ainda serão tratado em trabalhos futuros.

REFERENCES

[1] Christian Abbet, Meryem M'hamdi, Athanasios Giannakopoulos, Robert West, Andreea Hossmann, Michael Baeriswyl, and Claudiu Musat. 2018. Churn Intent Detection in Multilingual Chatbot Conversations and Social Media. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Association for Computational Linguistics, Brussels, Belgium, 161–170. https://doi.org/10.18653/v1/K18-1016

[2] S. A. Abdul-Kader and J. Woods. 2017. Question answer system for online feedable new born Chatbot. In *2017 Intelligent Systems Conference (IntelliSys)*. 863–869. https://doi.org/10.1109/IntelliSys.2017.8324231

[3] A. Argal, S. Gupta, A. Modi, P. Pandey, S. Shim, and C. Choo. 2018. Intelligent travel chatbot for predictive recommendation in echo platform. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. 176–183. https://doi.org/10.1109/CCWC.2018.8301732

[4] C. J. Baby, F. A. Khan, and J. N. Swathi. 2017. Home automation using IoT and a chatbot using natural language processing. In *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*. 1–6. https://doi.org/10.1109/IPACT.2017.8245185

[5] Botium. 2019. Botium - The Selenium for Chatbots. https://botium.atlassian.net/wiki/spaces/BOTIUM/overview

[6] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3rd ed.). IEEE Computer Society Press, Washington, DC, USA.

[7] J. Bozic, O. A. Tazl, and F. Wotawa. 2019. Chatbot Testing Using AI Planning. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. 37–44. https://doi.org/10.1109/AITest.2019.00-10

[8] Josip Bozic and Franz Wotawa. 2018. Security Testing for Chatbots. In *Testing Software and Systems*, Inmaculada Medina-Bulo, Mercedes G Merayo, and Robert Hierons (Eds.). Springer International Publishing, Cham, 33–38.

[9] Pin-Jung Chen, I-Hung Hsu, Yi-Yao Huang, and Hung-Yi Lee. 2017. Mitigating the impact of speech recognition errors on chatbot using sequence-to-sequence model. 497–503. https://doi.org/10.1109/ASRU.2017.8268977

[10] Dialogflow. 2019. Dialogflow - Conceitos. https://cloud.google.com/dialogflow/docs/concepts?hl=pt-br

[11] Gartner. 2017. 4 Uses for Chatbots in the Enterprise. https://www.gartner.com/smarterwithgartner/4-uses-for-chatbots-in-the-enterprise/

[12] Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazare, and Jason Weston. 2019. Learning from Dialogue after Deployment: Feed Yourself, Chatbot!. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 3667–3684. https://doi.org/10.18653/v1/P19-1358

[13] Hill, W. Ford, and Ingrid Farreras. 2015. Real conversations with artificial intelligence: A comparison between human-human conversations and human-chatbot conversations. *Computers in Human Behavior* 49 (01 2015), 245–250.

[14] Douglas Hoffman. 1999. Test Automation Architectures: Planning for Test Automation. *Copyright* (01 1999).

[15] IBM. 2019. IBM Cloud Documentos - Assistente do Watson. https://cloud.ibm.com/docs/assistant?topic=assistant-intents&locale=pt-BR

[16] IBM. 2019. IBM Cloud Functions - Plataforma Functions-as-a-Service (FaaS) com base no Apache OpenWhisk. https://cloud.ibm.com/functions/

[17] IBM. 2019. Watson Assistant - Build an assistant that your customers actually want to use. https://www.ibm.com/cloud/watson-assistant/

[18] ISO. 2013. ISO/IEC/IEEE 29119-1: Software and systems engineering – Software testing–Part 1: Concepts and definitions.

[19] Jieming Ji, Qingyun Wang, Zev Battad, Jiashun Gou, Jingfei Zhou, Rahul Divekar, Craig Carlson, and Mei Si. 2017. A Two-Layer Dialogue Framework For Authoring Social Bots. *Alexa Prize Proceedings* (2017).

[20] Karolina Kuligowska. 2015. Commercial Chatbot: Performance Evaluation, Usability Metrics and Quality Standards of Embodied Conversational Agents. *Professionals Center for Business Research* 2 (01 2015), 1–16. https://doi.org/10.18483/PCBR.22

[21] Divya Kumar and Krishn Mishra. 2016. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia Computer Science* 79 (12 2016), 8–15. https://doi.org/10.1016/j.procs.2016.03.003

[22] D. Madhu, C. J. N. Jain, E. Sebastain, S. Shaji, and A. Ajayakumar. 2017. A novel approach for medical assistance using trained chatbot. In *2017 International Conference on Inventive Communication and Computational Technologies (ICICCT)*. 243–246. https://doi.org/10.1109/ICICCT.2017.7975195

[23] Eduardo Moresi. 2003. Metodologia da Pesquisa.

[24] Node-RED. 2019. Node-RED - Low-code programming for event-driven applications. https://nodered.org/

[25] B. R. Ranoliya, N. Raghuwanshi, and S. Singh. 2017. Chatbot for university related FAQs. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 1525–1530. https://doi.org/10.1109/ICACCI.2017.8126057

[26] RASA. 2019. RASA - Forms. https://rasa.com/docs/rasa/core/forms/

[27] RASA. 2019. The Rasa Core Dialogue Engine. https://rasa.com/docs/rasa/core/about/

[28] Elayne Ruane, Théo Faure, Ross Smith, Dan Bean, Julie Carson-Berndsen, and Anthony Ventresque. 2018. BoTest: a Framework to Test the Quality of Conversational Agents Using Divergent Input Examples. 1–2. https://doi.org/10.1145/3180308.3180373

[29] V. Selvi, S. Saranya, K. Chidida, and R. Abarna. 2019. Chatbot and bullyfree Chat. In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*. 1–5. https://doi.org/10.1109/ICSCAN.2019.8878779

[30] M. Su, C. Wu, K. Huang, Q. Hong, and H. Wang. 2017. A chatbot using LSTM-based multi-layer embedding for elderly care. In *2017 International Conference on Orange Technologies (ICOT)*. 70–74. https://doi.org/10.1109/ICOT.2017.8336091

[31] Telegram. 2019. Telegram - a new era of messaging. https://telegram.org/

[32] Marisa Vasconcelos, Heloisa Candello, Claudio Pinhanez, and Thiago dos Santos. 2017. Bottester: Testing Conversational Systems with Simulated Users. https://doi.org/10.1145/3160504.3160584

[33] Dietmar Winkler, Reinhard Hametner, Thomas Östreicher, and Stefan Biffl. 2010. A framework for automated testing of automation systems. *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 1 – 4. https://doi.org/10.1109/ETFA.2010.5641264

[34] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. 2016. Building a Chatbot with Serverless Computing. 1–4. https://doi.org/10.1145/3007203.3007217